Visual C++ & MSWindows

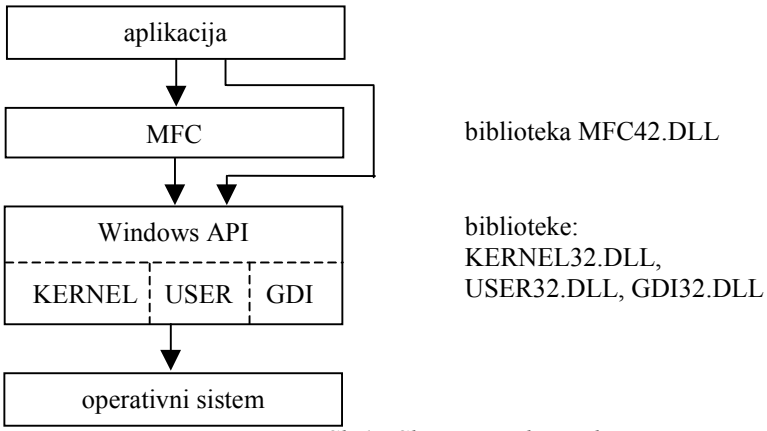
Sadržaj

1. Struktura i organizacija Windows programa

Operativni sistem Windows nastao je iz težnje da se unificira i pojednostavi programiranje u grafičkom okruženju, kao i da se standardizuje izgled korisničkog interfejsa softvera. MSDOS aplikacije su tipično pravljene tako da je svaka firma kreirala svoj izgled korisničkog interfejsa, sa raznim specifičnim komandama koje je korisnik morao da nauči da bi mogao uopšte da koristi aplikaciju. Korišćenje svakog novog softverskog paketa zahtevalo je da korisnik odvoji jedan dobar deo vremena za obuku. Sa druge strane, Windows aplikacije koriste objekte grafičkog okruženja koje su deo operativnog sistema (meniji, prozori, dijalozi, ...) i zato izgledaju isto u svim aplikacijama. Drugi motiv za prelazak na Windows je bio mogućnost multitaskinga, koji nije bio podržan u MSDOS-u. Osim ovoga, Windows je ponudio jedinstven način upravljanja memorijom, kao i interfejs prema hardveru. Ovo poslednje je naročito značajno, jer je pristup periferijskim uređajima u MSDOS-u zahtevao da isporučilac softvera zajedno sa softverom pravi i veliki broj drajvera za npr. različite grafičke karte ili različite štampače. U operativnom sistemu Windows za drajver je zadužen proizvođač hardvera (ukoliko već ne postoji u skupu Windowsovih standardnih drajvera), koji onda koriste sve aplikacije koje zahtevaju dati uređaj. Ovo značajno pojednostavljuje programiranje aplikacije, jer se uređaju pristupa uvek na standardan način, preko operativnog sistema, a ne direktno.

Sam operativni sistem Windows postepeno je evoluirao iz grafičke nadgradnje MSDOS-a: prve verzije su predstavljale samo GUI (graphical user interface), a tek je prelazak na sa 16-bitne na 32-bitnu varijantu dao Windows-u sve atribute pravog operativnog sistema: sopstveno upravljanje procesima (multitasking), upravljanje memorijom (uvedeno je 32-bitno adresiranje i "flat" memorijski model umesto starog segmentiranog način korišćenja memorije koje je predstavljalo noćnu moru za programere), sopstveni fajl sistem itd. Windows se danas isporučuje u tri varijante: (1) Windows 9x (95, 95OSR2, 98), (2) Windows NT (3.51, 4.0, 2000) i (3) Windows CE. Postoje značajne razlike u funkcionisanju "ispod haube" ove tri varijante: njih pokreću praktično tri različita kernela (jezgra) operativnog sistema; njih su razvijala nezavisno tri Microsoft-ova tima programera. Njima je jedino isti spoljašnji izgled i - način korišćenja API-ja (dok su implementacije API-ja, normalno, različite). Njihove razlike su posledica toga što su namenjeni za različita tržišta, koja normalno imaju različite zahteve: NT serija je predviđena za poslovne primene gde se zahteva pre svega pouzdanost; 9x serija je namenjena za kućnu primenu, gde je bitna kompatibilnost sa 16-bitnim Windows-ima i grafička podrška; CE je optimizovan za uređaje sa ograničenom potrošnjom, kao što su prenosivi računari.

API (application programming interface) predstavlja skup sistemskih poziva - usluga (servisa) operativnog sistema na najnižem nivou, koji je predviđen da se koristi od strane aplikativnog softvera. Kako je prikazano na slici 1, aplikacija poziva API kada god treba da pristupi nekom resursu operativnog sistema. Ne može se pristupiti nekom delu operativnog sistema bez korišćenja API-ja.



Sl. 1 - Slojevi Windows okruženja.

Windows API se sastoji od tri celine, KERNEL-a, USER-a, i GDI-ja, koji se nalaze u tri sistemske dinamičke biblioteke (DLL-a). KERNEL predstavlja jezgro operativnog sistema; on je zadužen za rad

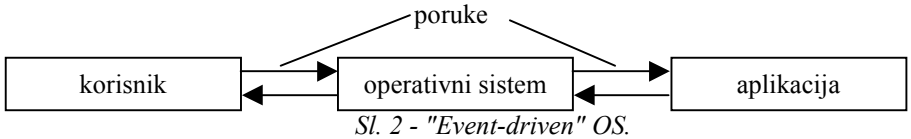
sa sistemskim objektima - procesima, nitima, ulazom i izlazom, memorijom. USER biblioteka omogućava rad sa objektima korisničkog interfejsa: prozorima, menijima, dijalozima, kontrolama, kurzorima, ikonama, itd. GDI bibliteka omogućava crtanje elemenata GUI - bilo da je u pitanju tekst, ili slike (podržava rastersku grafiku - bitmape, kao i vektorsku grafiku - metafajlove).

Način na koji funkcioniše Windows aplikacija obično u početku zbunjuje ljude koji su navikli da programiraju "konzolno". Uporedimo ova dva pristupa. Tipičan konzolni program u MSDOS-u izgleda nekako ovako:

```
#include <stdio.h>
int main(int argc, char *argv[])
{   /* ulaz */
    ... scanf ...

    /* obrada */
    ...

    /* izlaz */
    ... printf ...
}
```

Ulaz se obezbeđuje ili iz komandne linije (argumenti funkcije main), ili sa standardnog ulaza (to može biti konzola ili fajl, ukoliko je urađena redirekcija). Program procesira podatke i izlaz ispisuje na standardni izlaz (opet konzolu ili fajl). Eventualno može da se vrati kod greške u komandno okruženje i na taj način omogući pravljenje BATCH procedura.

Za razliku od MSDOS-a, Windows je "even driven" operativni sistem - vođen događajima (slika 2). Rad Windows aplikacije se zasniva na obradi događaja koji stižu od korisnika (pokretanje miša, pritisak tastera, itd.). Šematski prikazano to izgleda ovako:



Sl. 2 - "Event-driven" OS.

Aplikacija predstavlja "produžetak" operativnog sistema. Naime, sve poruke koje stižu od korisnika obrađuje OS. Kada kreiramo sopstvenu aplikaciju koja ima svoj prozor, unutar tog prozora odgovornost za pristigle poruke preuzima sama aplikacija. Windows program se, u najjednostavnijoj varijanti, sastoji iz dve celine: ulazne funkcije aplikacije WinMain, i funkcije koja obrađuje poruke WndProc. Ulazna funkcija aplikacije WinMain poziva se prilikom startovanja aplikacije. Njen zadatak je da obavi inicijalizaciju aplikacije, kreiranje glavnog prozora, i da u jednoj beskonačnoj petlji, tzv. petlji poruka (message loop) prosleđuje poruke pristigle od strane OS funkciji za obradu poruka WndProc. Iz petlje se izlazi kada stigne jedna specifična poruka, WM_QUIT, koja označava da je korisnik zahtevao gašenje aplikacije. Argumenti funkcije WinMain imaju sledeće značenje: hinstExe označava hendl (identifikacioni broj) aplikacije dodeljen od strane OS pri kreiranju novog procesa za aplikaciju; hinstPrev se ne koristi u 32-bitnim Windows-ima i predstavlja ostatak iz Windows-a 3.11; pszCmdLine je string u kome se nalazi poziv aplikacije iz komandne linije; nCmdShow je skup flegova koji određuju inicijalni izgled prozora aplikacije (da li je minimizovan, maksimizovan, standardan, ...).

```
int WINAPI WinMain(HINSTANCE hinstExe, HINSTANCE hinstPrev, PSTR
pszCmdLine, int nCmdShow)
{   /* inicijalizacija aplikacije */
    ...

    /* kreiranje prozora */
    ...

    /* petlja poruka - "message loop" */
    ...
```

```
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam)
{   ...
    /* obrada poruka u prozoru */
    switch(message)
    {   case WM_CREATE:
            /* inicijalizacija prozora */
            break;
        case WM_LBUTTONDOWN:
            /* akcija na pritisak na levi taster misa */
            ...
            break;
        case WM_MOUSEMOVE:
            /* akcija na pomeranje misa */
            ...
            break;
        case WM_PAINT:
            /* osvezavanje prozora */
            ...
            break;
        case WM_DESTROY:
            /* posalji poruku WM_QUIT */
            break;
    }
    /* poziv default obrade poruka */
}
```

Funkcija za obradu poruka WndProc poziva se implicitno (znači ne od strane programera) kada stigne neka poruka aplikaciji od strane OS. Pristiglu poruku funkcija DispatchMessage iz petlje poruka u funkciji WinMain prosleđuje prozorskoj funkciji WndProc. Prozorska funkcija na osnovu broja poruke, u switch-case strukturi, prelazi na izvršavanje dela programa zaduženog za obradu pristigle poruke. Treba primetiti da se sav "koristan" kod aplikacije nalazi u prozorskoj funkciji WndProc, dok ulazna funkcija WinMain samo obavlja režijske poslove. Neke tipične prozorske poruke su:
WM_CREATE: OS šalje ovu poruku prozoru kada ga kreira. Kod koji obrađuje ovu poruku treba da obavi neku dodatnu inicijalizaciju prozora.
WM_SIZE, WM_MOVE: Prozor dobija ove poruke kada se reskalira, odnosno pomera.
WM_PAINT: Kada stigne ova poruka to znači da se zahteva ponovno iscrtavanje sadržaja prozora, bilo iz razloga da je sadržaj prozora oštećen (npr. zaklonio ga je neki drugi prozor), bilo da su podaci u prozoru zastareli, pa ih treba osvežiti. U ovom delu koda treba da bude skoncentrisan sav izlaz, tj. sva iscrtavanja i ispisivanja.
WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_KEYDOWN: Ove poruke stižu kada korisnik nešto radi u oblasti prozora: pomera miša, pritiska levi taster na mišu, ili pritiska neki taster na tastaturi. U kodu koji obrađuje ove poruke treba da se nalazi sav ulaz. Ovaj deo koda predstavlja praktično korisnički interfejs prozora.
WM_DESTROY: Prozor dobija ovu poruku kada korisnik klikne na dugme x u gornjem desnom uglu title bar-a. To znači da se zahteva uništavanje prozora. Kod koji se izvršava nakon pristizanja ove poruke treba da obavi neka dodatna čišćenja (npr. nekih dinamički alociranih promenljivih), ukoliko postoji potreba za tim, pre gašenja prozora. Ukoliko je to i glavni prozor aplikacije, potrebno je poslati poruku WM_QUIT, čime se u funkciji WinMain iskače iz petlje poruka i završava izvršavanje aplikacije.
Nakon ove switch-case strukture sledi poziv default funkcije za obradu poruka - u njoj će se poruke obraditi "na standardan način" koji zahteva Windows, čak i one poruke koje nisu eksplicitno navedene u switch-case strukturi.

Jedan vrlo koristan primer, "HelloWin" dat je u knjizi [1]. U njemu se može videti kako u stvarnosti izgleda konkretan kod (koji se može iskompajlirati i startovati) ovakve najjednostavnije Windows aplikacije.

Ovaj bazični način progamiranja aplikacije pozivanjem API funkcija iz C programa naziva se još i SDK programiranjem (prema Software Development Kit-u). Kod većih aplikacija koje direktno koriste Windows API kod programa se znatno komplikuje i veoma brzo postaje nečitljiv. To je zbog toga što jezik C nije "prirodan" jezik za opisivanje event-driven aplikacija. Mnogo prirodniji pristup bi bio u jeziku C++ napraviti objekte "prozor", "aplikacija", itd. i izmenjivati poruke između njih. Microsoft je ponudio jedan način da se ovo uradi, preko svoje MFC biblioteke (Microsoft Foundation Classes). MFC biblioteka predstavlja skup klasa koje definišu osnovne Windows objekte i njima pridružene operacije. MFC klase se u potpunosti baziraju na Windows API-ju, i njihovim korišćenjem se u suštini pozivaju odgovarajuće API funkcije koje vrše kreiranje, manipulaciju i uništavanje tih objekata (slika 1). MFC znatno uprošćava kod, i kod većih aplikacija postaje nezamenjiv alat. Na slikama 3a-3c prikazana je hijererhija MFC-a. MFC ne pokriva kompletan Windows API, već samo oko dve trećine funkcija. To su one najvažnije funkcije, za pravljenje kostura aplikacije i korisničkog interfejsa. Osim toga što predstavlja "oblandu" za API, MFC nudi i niz izvedenih klasa kojima se dobija veća funkcionalnost u odnosu na API.

U MFC-u nije uobičajeno da se ponuđene klase koriste direktno, tj. da se instanciraju objekti baš tih klasa. Standardan način je da se iz potrebne klase izvede sopstvena klasa, sa specifičnostima koje trebaju našoj aplikaciji, a da se zatim instancira objekat te izvedene klase. MFC aplikacija počiva na dve osnovne klase: CWinApp i CMDIFrameWnd (u slučaju single-document-interface-a, klasa CFrameWnd). Iz njih se izvode klase C_ime_aplikacije_App (pošto se primer koji ćemo stalno koristiti i nadograđivati zove Prvi, ova klasa će biti CPrviApp) i CMainFrame, i objekti te dve klase praktično predstavljaju aplikaciju. Najvažniji delovi koda MFC aplikacije izgledaju otprilike ovako:

```
CPrviApp theApp; // objekat aplikacije (klasa CPrviApp je izvedena iz
                 // klase CWinApp)

int AFXAPI AfxWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPTSTR lpCmdLine, int nCmdShow)
{   ...
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    // Inicijalizacija
    // AfxInit ...
    pApp->InitApplication();
    pApp->InitInstance();
    ...
    // Kraj programa
    pApp->ExitInstance();
}

BOOL CPrviApp::InitInstance()
{   // Standardna inicijalizacija objekta aplikacije
    ...
    // Ucitavanje podesavanja iz sistemskog registra
    ...
    // Registracija dokumenta
    ...
    // Kreiranje glavnog prozora aplikacije
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME)) return FALSE;
    m_pMainWnd = pMainFrame;
    // Parsiranje komandne linije
    ...
    // Prikazivanje glavnog prozora aplikacije
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();
    return TRUE;
}
```

Iz klase CWinApp izveli smo klasu CPrviApp, i kreirali globalni objekat te klase theApp. Ulazna tačka programa je globalna funkcija AfxWinMain, koja redom obavlja razne inicijalizacije objekta aplikacije. Osnovna klasa CWinApp obezbeđuje petlju poruka u funkciji CWinApp::Run(), a takođe i member

varijablu m_pMainWnd - pointer na objekat glavnog prozora (CMainFrame klase). Funkcija CPrviApp::InitInstance() je mesto od kog program počinje "da živi" sa stanovišta programera. U tu funkciju treba staviti sav kod koji vrši inicijalizaciju aplikacije. Ova funkcija ima svoj par, funkciju CPrviApp::ExitInstance(), koja se poziva prilikom gašenja aplikacije, i koja je predviđena da obavlja sva dodatna čišćenja pre nego što se aplikacija zatvori. Inicijalizacija se kod MFC klasa koje realizuju elemente grafičkog korisničkog interfejsa retko radi u konstruktoru klase, već skoro uvek u funkciji koja je za to predviđena. Kako je već rečeno, inicijalizacija aplikacije se radi u InitInstance(), a takođe se npr. inicijalizacija dijaloga radi u funkciji OnInitDialog() a ne u konstruktoru dijaloga, itd. Razlog tome je to što u se vreme izvršavanja konstruktora ne garantuje da su svi potrebni objekti - elementi korisničkog interfejsa kreirani.

U funkciji InitInstance se, pored drugih stvari, vrši i kreiranje glavnog prozora aplikacije. Glavni prozor aplikacije (main frame) je odgovoran za kreiranje ostatka korisničkog interfejsa - status bara, tulbarova, drugih prozora, dijaloga, itd.

Opis elemenata korisničkog interfejsa čuva se u jednoj običnoj tekst datoteci, tzv. resursima. Visual C++ poseduje grafički editor ovih elemenata, u kojem se oni crtaju. Tek u fazi prevođenja programa iz ove tekst datoteke se čitaju podaci o dimenzijama, poziciji tih elemenata i ugrađuju u izvršni kod. Elementima korisničkog interfejsa su dodeljeni jednoznačni resursni identifikatori (ID_NESTO), pomoću kojih se oni povezuju sa kodom. Pogledajmo kako to funkcioniše na primeru menija. Meni opcija Help>About predviđena je da aktivira tzv. About dijalog, koji sadrži osnovne podatke o programu: ime, verziju, ime autora, eventualno kontakt adresu. Resursni identifikator ove opcije je ID_APP_ABOUT (on ima neku konkretnu vrednost koja nas ne zanima). U kodu klase koja će koristiti ovaj resurs, a u slučaju ovog dijaloga to je CPrviApp, Visual C++ generiše poziv makroa tzv. mape poruka (message map), koja vrši mapiranje (preslikavanje) resursnih identifikatora u konkretne pozive funkcija:

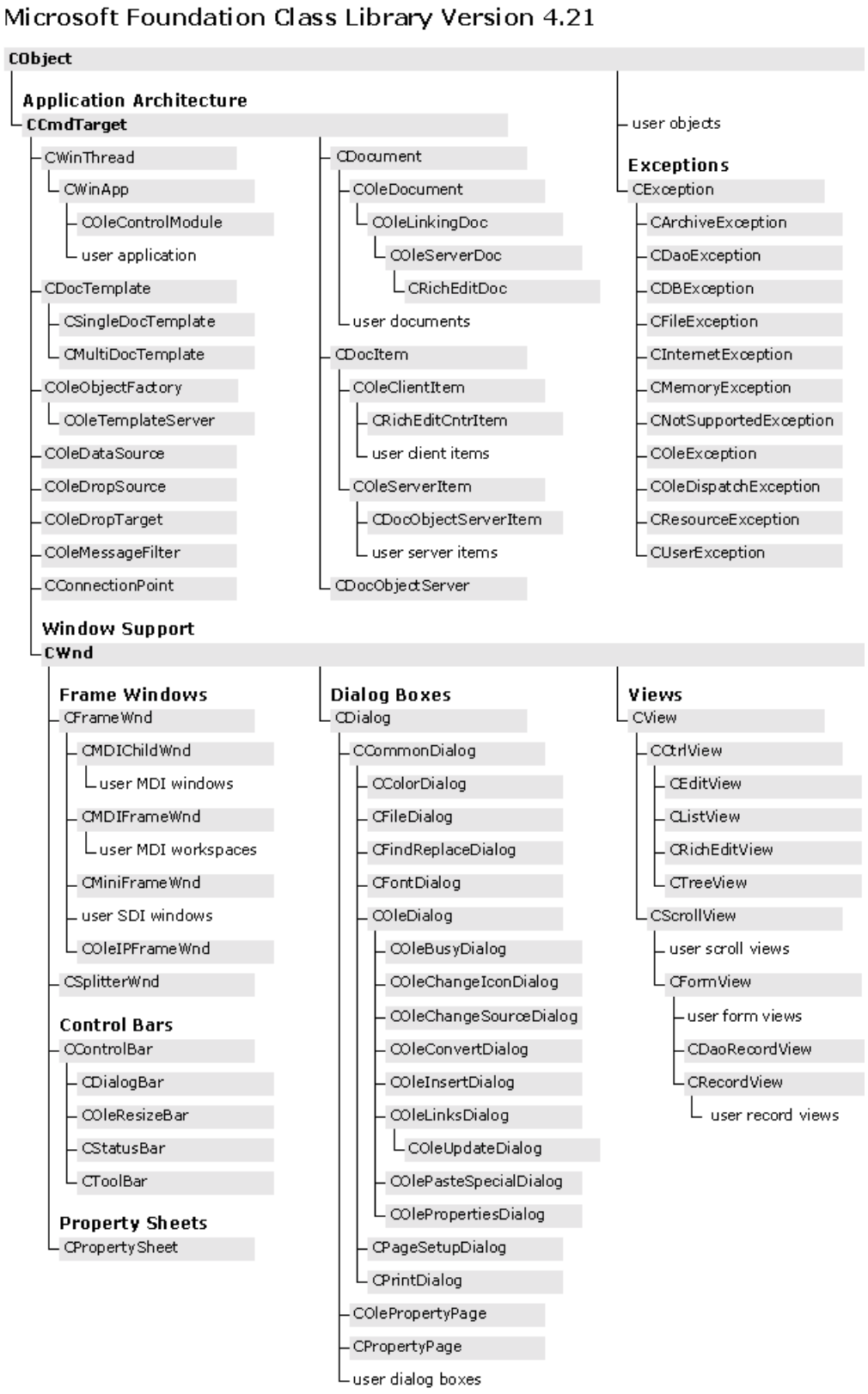
```
BEGIN_MESSAGE_MAP(CPrviApp, CWinApp)
    //{{AFX_MSG_MAP(CPrviApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // NOTE-the ClassWizard will add and remove mapping macros here.
    // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
```

Ovo praktično znači da će aktiviranje ove opcije izazvati poziv hendlera (funkcije) CPrviApp::OnAppAbout. U ovoj funkciji se dalje vrši kreiranje About dijaloga. Napomena: ne treba mešati resursni identifikator meni opcije koja startuje dijalog (ID_APP_ABOUT) i resursni identifikator samog dijaloga (IDD_ABOUTBOX).

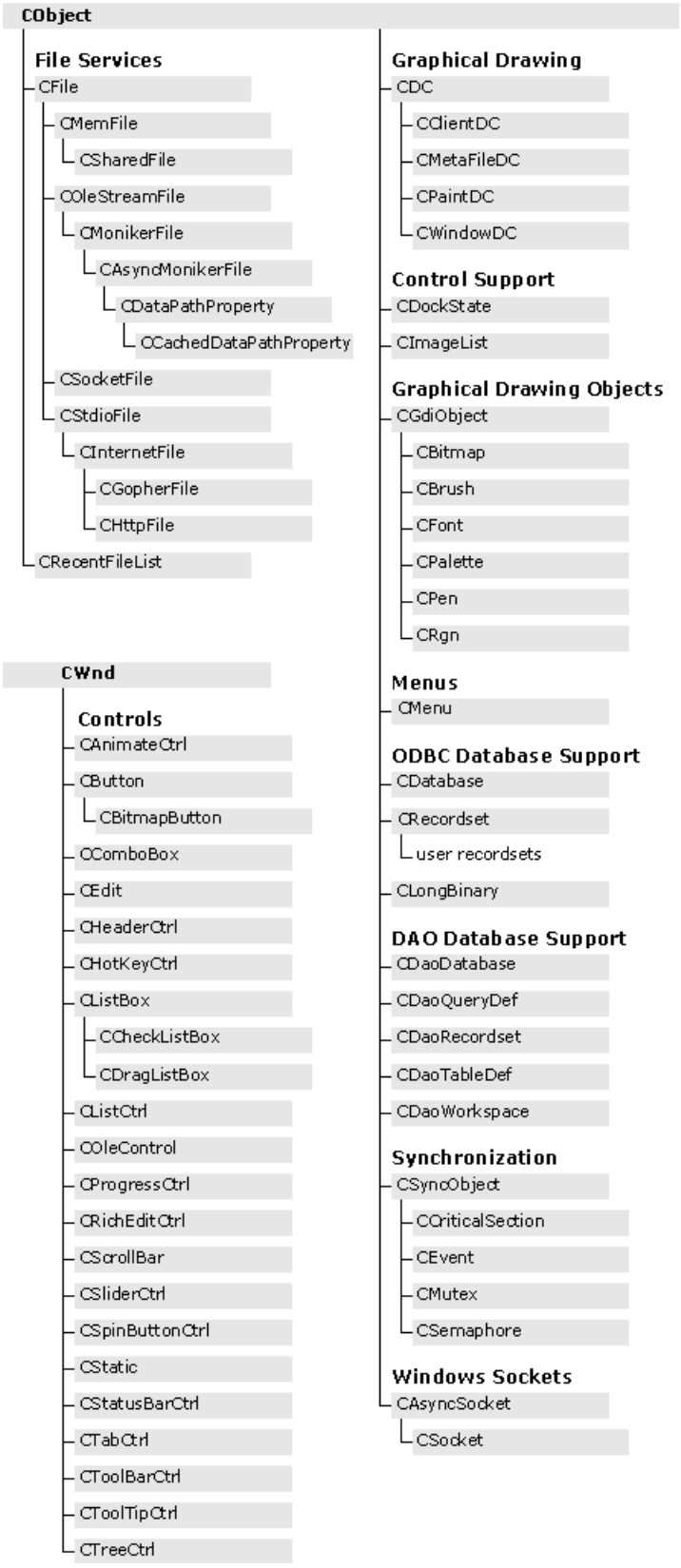
Kod aplikacije bazirane na API-ju, obradu poruka koje stižu u prozor vršili smo u prozorskoj funkciji WndProc, u switch-case naredbi. U MFC aplikaciji svakoj poruci se dodeljuje posebna funkcija, tzv. hendler, koji sadrži kod koji se izvršava po pristizanju te poruke. Kako kostur aplikacije zna da treba da pozove baš tu funkciju kada stigne data poruka? Pa, isto preko mape poruka. Pomoću jednog pomoćnog programa iz Visual C++ integrisanog okruženja, Class Wizard-a, svakoj poruci može da se dodeli hendler u datoj klasi. Class Wizard upisuje makro u mapu poruka kojim se povezuje poruka i hendler. Na primer, za poruku WM_MOUSEMOVE, Class Wizard će u mapu poruka dodati makro ON_WM_MOUSEMOVE(), a hendler za obradu ove poruke će biti Ime_Klase::OnMouseMove(UINT nFlags, CPoint point). Napomena: osim u izuzetnim slučajevima, mapu poruka ne treba ručno editovati, već isključivo preko Class Wizard-a. Naročito ne treba modifikovati deo koji se nalazi unutar AFX_MSG_MAP o čemu Visual C++ generiše komentar sa upozorenjem (u Visual C++ editoru ti delovi koda koji su zabranjeni za ručno editovanje su obojeni sivom bojom).

Domaći zadaci.

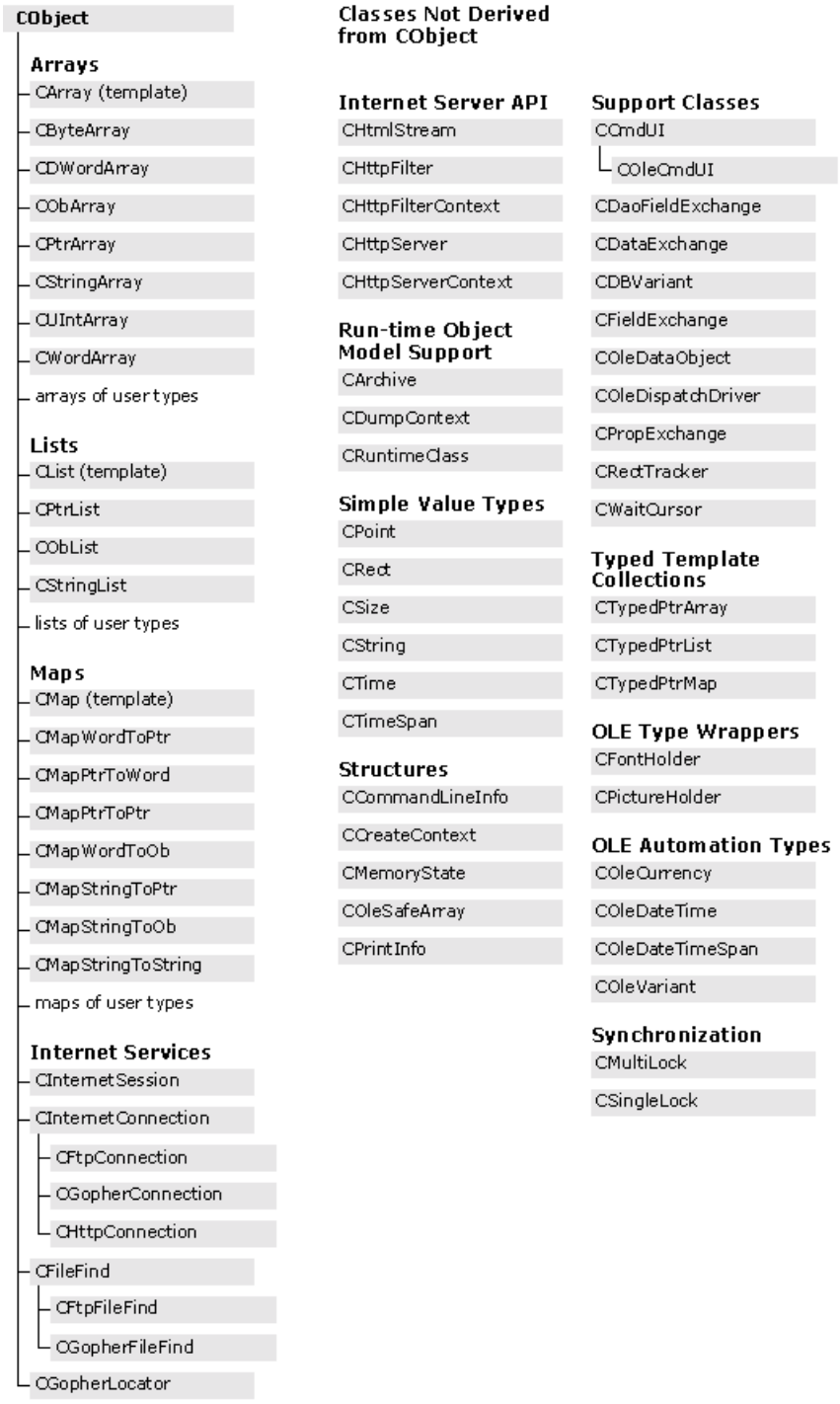
1. Pomoću Application Wizard-a, kreirati nov projekat (MFC AppWizard EXE / Multiple Documents / MFC as a shared DLL). Nazvati ga Prvi. Ovu aplikaciju ćemo dalje koristiti u nastavku kursa i nadograđivati je raznim elementima.
2. Videti kod koji kompajler generiše po defaultu, za "praznu" aplikaciju. Obratiti pažnju na spomenute klase - CPrviApp i CMainFrame.
3. Iskompajlirati praznu aplikaciju u režimima Debug i Release (Build>Set Active Configuration). Uporediti dužine dobijenih EXE fajlova za oba režima.
4. Isprobati funkcionalnost "prazne" aplikacije.



Sl. 3a - Hijerarhija MFC klasa (prvi deo).



Sl. 3b - Hijerarhija MFC klasa (drugi deo).



Sl. 3c - Hijerarhija MFC klasa (treći deo).

2. Doc/View arhitektura

Veliki broj Windows aplikacija pripada kategoriji aplikacija koje su specijalizovane za obradu jedne iste vrste dokumenata - npr. programi za obradu teksta, za crtanje, za prikaz bitmapa, za tabelarne proračune, itd. (skoro sve aplikacije iz paketa MS Office mogu da se uzmu za primer ovakve vrste aplikacija). Ako posmatramo funkcionisanje takvih programa, vidimo da se one razlikuju pre svega po reprezentaciji podataka i algoritmima za obradu tih podataka, što je i normalno, pošto je fizičko značenje tih podataka raznoliko. Međutim, ono što nazivamo "kostur aplikacije" *(application framework)* je isti (ili bolje reći sličan) kod svih njih. Svaki od njih ima učitavanje, snimanje dokumenta, izlaz na printer, undo, operacije sa klipbordom, razne vrste prikaza podataka (npr. tabelarno, ili kao grafik), on line help sistem, a takođe i funkcionisanje na komande "opšteg tipa" je slično - npr. pita se da li treba snimiti izmenjeni dokument prilikom njegovog zatvaranja, itd.

MFC nudi klase koje u aplikacijama ovog tipa obavljaju režijske poslove i na taj način omogućavaju programeru da se u potpunosti posveti onome što aplikacija treba da radi. To je par klasa CDocument (dokument) i CView (prikaz, view). Korisnik iz ove dve klase izvodi svoje klase sa specifičnim ponašanjem za datu primenu. Dokument treba da sadrži podatke, i operacije sa njima (algoritme). View je zadužen za komunikaciju sa korisnikom - kako za unos i izmenu podataka, tako i za njhov prikaz. On predstavlja korisnički interfejs aplikacije. Jedan dokument može da se veže sa više različitih View-a, što praktično znači da jedna ista vrsta podataka može imati više različitih korisničkih interfejsa (opet spominjemo primer niza brojeva: mogu se prikazati tabelarno, kao grafik ove ili one vrste, itd.).

Klasa dokumenta CDocument je izvedena iz klase CCmdTarget, koja je opet izvedena iz klase CObject. Klasa CView je izvedena iz klase prozora CWnd, koja je izvedena iz CObject. Zbog toga se view ponaša kao klasičan Windows prozor sa još nekim dodatnim osobinama. Korisnički interfejs u view-u funkcioniše na sličan način kao za prozor: na osnovu prozorskih poruka *(window messages, WM)*, pozivaju se funkcije hendleri koje obrađuju događaje vezane za tu poruku (npr. WM_LBUTTONDOWN se šalje kada je pritisnut levi taster miša, što izaziva pozivanje hendlera OnLButtonDown).

Crtanje u view-u se obavlja isto kao i za prozor: u hendler funkciji OnPaint koja je odgovor na poruku WM_PAINT. Ova poruka se šalje prozoru svaki put kad je njegov sadržaj zastareo pa ga treba osvežiti, bilo da je to izazvano spolja (npr. reskaliran je, ili je neki drugi prozor preklopio neki njegov deo), ili iz same aplikacije (npr. zbog promene sadržaja dokumenta).

U kreiranu aplikaciju Prvi ubacićemo kod koji će joj dati osnovnu funkcionalnost Doc/View arhitekture. Radi jednostvnosti, neka nam dokument bude samo jedan podatak tipa int. Hoćemo da ovaj podatak možemo da unesemo, prikažemo, snimo u fajl i učitamo iz fajla. Za prve dve operacije biće zadužen view, pošto je to posao korisničkog interfejsa, a za druge dve dokument, pošto one ne zavise od načina prikaza podataka. Takođe ćemo isprobati način na koji funkcioniše korisnički interfejs u klasi view "hvatanjem" poruka od miša. Hoćemo da, npr. naš view prikazuje i trenutnu poziciju pokazivača miša. Osim toga, uvešćemo snimanje podešavanja aplikacije u sistemski registar (Registry) i čitanje podešavanja iz njega. Parametar za podešavanje biće jedna BOOL promenljiva refresh, koja određuje da li se prikaz osvežava stalno, ili na zahtev korisnika.

Uraditi sledeće:

1. Definisati member varijablu CPrviDoc klase koja će predstavljati dokument: u PrviDoc.h ubaciti public promenljivu int sadrzaj, i u konstruktor dokumenta (u PrviDoc.cpp) ubaciti incijalizaciju ove member varijable sadrzaj=15.
2. U Class Wizard-u (CTRL+W), u klasi CPrviView, dodati hendlere za poruke od miša (sekcija Message Map): WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_RBUTTONDOWN, WM_KEYDOWN. Dodati i hendler za poruku WM_PAINT, za crtanje po prozoru view-a. Svaka poruka WM_NESTO dobiće hendler funkciju CPrviView::OnNesto koja će se pozvati kada stigne ta poruka (koja je izazvana tačno odgovarajućim događajem, npr. pomeranjem miša ili pritiskanjem dugmeta na mišu).

3. Dodati member varijable za korisnički interfejs; u CPrviView.h dodati public member varijable int mousex, int mousey, BOOL refresh. U konstruktor view-a ubaciti incijalizaciju ovih promenljivih: mousex=mousey=0, refresh=TRUE.
4. Napisati hendlere za miša:

```
CPrviView::OnMouseMove(UINT nFlags, CPoint point)
{   mousex=point.x;
    mousey=point.y;
    if(refresh) Invalidate(FALSE);
    ...
}

CPrviView::OnLButtonDown(UINT nFlags, CPoint point)
{   ...
    Invalidate(TRUE);
}

CPrviView::OnRButtonDown(UINT nFlags, CPoint point)
{   ...
    refresh=!refresh;
    ...
}

CPrviView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{   ...
    GetDocument()->sadrzaj=nChar;
    GetDocument()->SetModifiedFlag(TRUE);
    if(refresh) Invalidate(FALSE);
    ...
}
```

Komentar skoro i da nije potreban: OnMouseMove prihvata koordinate miša i upisuje u member varijable view-a mousex i mousey. Ukoliko se zahteva automatsko osvežavanje, sadržaj prozora view-a se proglašava zastarelim pozivom funkcije Invalidate. Argument ove funkcije znači da li se prozor briše posle invalidacije. Stavili smo "ne" zato što često brisanje celog prozora (što bi se dešavalo pri svakom pomeranju miša preko prozora view-a) izaziva vidljivo treperenje. OnLButtonDown odrađuje osvežavanje na zahtev. Tada može da se briše ceo prozor view-a. OnRButtonDown menja stanje logičke promenljive refresh, odnosno menja način osvežavanja view-a. OnKeyDown kod pritisnutog tastera upisuje u dokument i, ukoliko je uključeno automatsko osvežavanje, proglasi view zastarelim. Takođe, proglasi dokument promenjenim (SetModifiedFlag) čime se omogućava framework-u da pri gašenju aplikacije pita korisnika da li da snimi dokument pošto je izmenjen.

5. Napisati hendler za poruku WM_PAINT:

```
CPrviView::OnPaint()
{   ...
    CString izlaz;
    izlaz.Format("Koordinate misa=(%d,%d) sadrzaj=%d       ",
        mousex,mousey,GetDocument()->sadrzaj);
    dc.TextOut(20,20,izlaz);
    ...
}
```

Hendler OnPaint se aktivira svaki put kada treba nanovo iscrtati sadržaj prozora. Jedan način "forsiranog" iscrtavanja, od strane programera, je proglašavanjem prozora nevažećim pozivom funkcije Invalidate.

6. Dodati kod za učitavanje i snimanje dokumenta. Ovo može da se uradi na dva načina: (1) serijalizacijom i (2) pomoću funkcija iz dokumenta OnFileSave, OnSaveDocument i OnOpenDocument Realizovaćemo prvi način. U funkciju CPrviDoc::Serialize, koju Application Wizard generiše pri kreiranju aplikacije, dodaćemo kod koji vrši snimanje i učitavanje:

```
CPrviDoc::Serialize(CArchive &ar)
{   if(ar.IsStoring())
    {   ar<<sadrzaj;
    }
    else
    {   ar>>sadrzaj;
    }
}
```

Ova funkcija se poziva implicitno od strane framework-a kada se zahteva učitavanje ili snimanje. Takođe se (implicitno) poziva standardni Windows File dijalog koji treba da vrati ime fajla i putanju. Podrazumevana ekstenzija fajla se zadaje prilikom kreiranja kostura aplikacije, u Application Wizard-u, u Advanced opcijama. Drugi način podrazumeva da se u funkciji OnFileSave ručno pozove File dijalog, i da se dobijeno ime fajla prosledi funkciji OnSaveDocument u kojoj se nalazi kod za snimanje. Kod učitavanja, CPrviApp::OnFileOpen treba da pozove fajl dijalog i pozive CWinApp::OpenDocumentFile i kao argument prosledi dobijeno ime fajla. Nakon što se kreira novi objekat dokumenta (i njemu pridruženi view), pozvaće se CPrviDoc::OnOpenDocument funkcija u kojoj se nalazi kod za snimanje. Drugi način omogućava direktniju kontrolu nad ovim procesom, i treba ga koristiti kada aplikacija podržava npr. više formata u kojima može da snimi dokument (onda bi se format fajla, odnosno ekstenzija, birala u fajl dijalogu).

7. Dodati snimanje podešavanja (promenljiva refresh) u sistemski registar. Dodati u CPrviApp klasu public member varijablu m_refresh (u fajl Prvi.h). Ova promenljiva će se učitati iz registra prilikom inicijalizacije aplikacije, a snimiti u registar prilikom gašenja aplikacije. U funkciju CPrviApp::InitInstance (fajl Prvi.cpp) dodati kod za učitavanje (default kod koji generiše Application Wizard isključiti stavljanjem pod komentar):

```
CPrviApp::InitInstance()
{   ...
    // SetRegistryKey... (ovo generiše AppWizard)
    SetRegistryKey(_T("VTA"));
    ...
    m_app_refresh=GetProfileInt("Init","refresh",TRUE);
    ...
}
```

U funkciju CPrviApp::ExitInstance dodati kod za snimanje ove promenjive u registar:

```
CPrviApp::ExitInstance()
{   ...
    WriteProfileInt("Init","refresh",m_app_refresh);
    ...
}
```

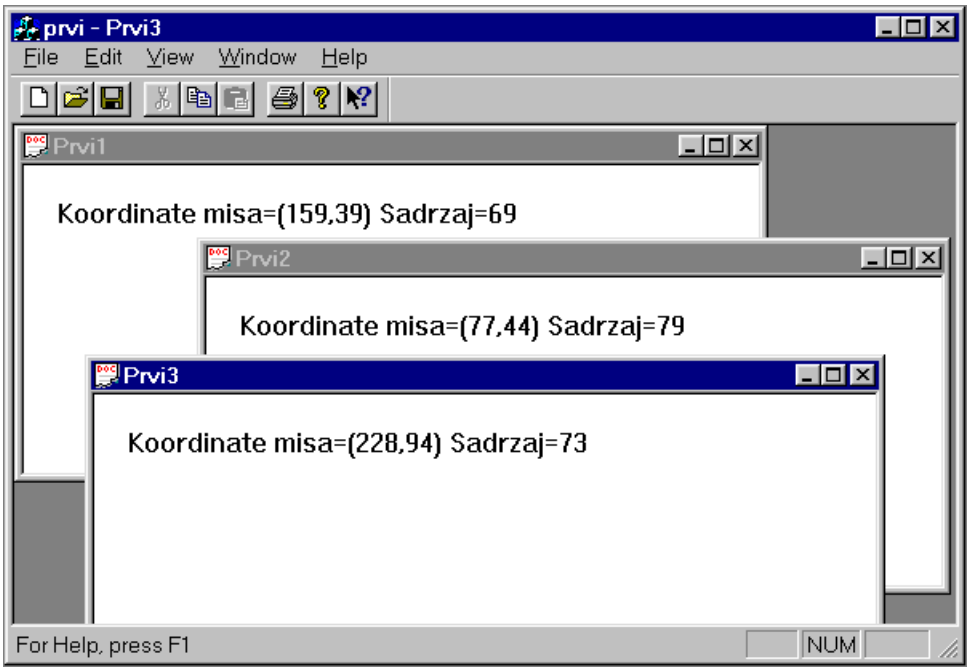
U konstruktor view-a sada treba uneti neke izmene. Umesto postavljanja default vrednosti treba uvesti prepisivanje iz m_app_refresh:

```
CPrviView::CPrviView()
{   //refresh=TRUE; //izbačeno
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    refresh=pApp->m_app_refresh;
}
```

Takođe treba izmeniti destruktor view-a tako što će mu se dodati upisivanje refresh u m_app_refresh:

```
CPrviView::~CPrviView()
{   CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    pApp->m_app_refresh=refresh;
}
```

Kada se program iskompajlira, treba da funkcioniše na sledeći način. Kada se mišem krećemo iznad dokumenta, koordinate će se prikazivati u view-u. Kada pritisnemo neki taster, njegov kod će biti zapamćen kao sadržaj dokumenta i takođe će biti prikazan u view-u. Pritiskom na desni taster miša biramo tip osvežavanja view-a - automatski ili po zahtevu. Drugi način osvežavanja funkcioniše tako što se čeka pritisak levog tastera na mišu da bi se obavilo osvežavanje (dotle će izgled view-a biti nepromenjen iako npr. dokument može da bude izmenjen). Sadržaj dokumenta može da se snimi u fajl, i da se učita nazad u neki drugi dokument. Podešavanja (refresh) se čuvaju u registru, i pamte nakon izlaska iz aplikacije. Izgled aplikacije je dat na slici 4.



Sl. 4 - Izgled aplikacije Prvi.

3. Meni. Undo bafer. Dijalog

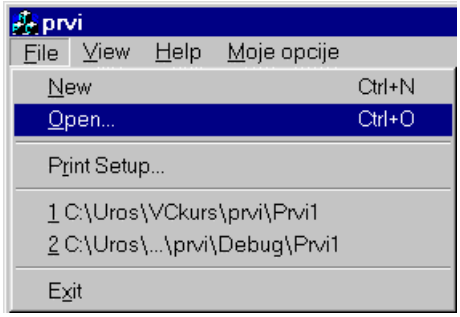
U prethodnoj lekciji upoznali smo se sa Doc/View arhitekturom aplikacije, u kojoj objekat dokumenta čuva podatke, a objekat view-a predstavlja interfejs prema korisniku. Sada ćemo upoznati još neke elemente korisničkog interfejsa, meni i dijalog. Pored toga videćemo kako se Doc/View aplikaciji dodaje mogućnost brisanja poslednje izvršene operacije - Undo.

3.1. Meni

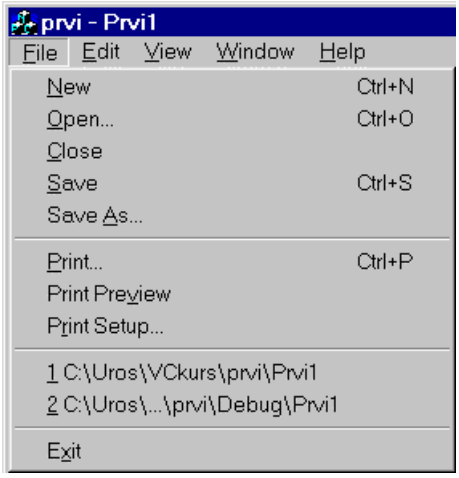
U Windows-u se komande obično mogu zadati trojako: preko menija - biranjem opcije mišem, preko menija - biranjem opcije tastaturom, i preko neke kombinacije tastera, tzv. akceleratorom. Uobičajeno je da se sve opcije mogu izabrati na prva dva načina, a da se akceleratori definišu samo za neke često korišćene operacije kao što su snimanje fajla (CTRL+S), kopiranje na klipbord (CRTL+C), itd. Takođe se često koriste i kontekstni meniji, koje korisnik dobija pritiskom na desni taster miša, i čiji sadržaj zavisi od mesta na koje je korisnik kliknuo - od konteksta. Ovu poslednju grupu nećemo raditi jer zahtevaju dinamičko kreiranje i uništavanje, što nije predmet ovog osnovnog kursa. U knjiizi [3] se može naći primer korišćenja kontekstnih menija. Još jedan način zadavanja komandi je preko tulbara, o čemu će biti reči u lekciji 4.

I obični meniji se mogu kreirati kako statički, tako i dinamički. Statičko kreiranje podrazumeva da se definišu u resursima, što praktično znači da se izgled menija zna u vreme kompajliranja. Kod dinamički kreiranih menija moguće je dodavati meni stavke u kodu, čime se menja njihov izgled u toku izvršavanja programa. Zbog jednostavnosti bavićemo se samo prvom grupom, menijima koji su potpuno definisani pre kompajliranja.

Kada se u Visual C++ resursnom editoru doda nova meni stavka, njoj se pridružuje i resursni identifikator. Kada korisnik u programu izabere neku meni stavku, generiše se poruka programu koju on treba da obradi. U prethodnoj lekciji smo videli kako se "hvataju" poruke: potrebno je definisati hendler funkciju koja sadrži kod za obradu poruke, i povezati je sa resursnim identifikatorom preko mape poruka. Meniji su specifični po tome što nude mogućnost odgovaranja na dva tipa poruka: *command* i *update command*. Prvi tip poruka, command, generiše se kada korisnik izabere meni stavku i on predstavlja zapravo, komandu, odnosno akciju koja treba da se izvrši. Drugi tip poruka, update command, generiše se svaki put kada korisnik otvori meni, a pre nego što bilo šta izabere. Hendler za ovu poruku odgovoran je za ažuriranje izgleda menija. Moguće je neke stavke onemogućiti (disable), štiklirati (check) ili ih prikazivati normalno. Onemogućena stavka se prikazuje sivom bojom i nije je moguće izabrati. Onemogućavanje se primenjuje kada u toku rada programa zbog trenutne situacije u programu izvršenje neke operacije nema smisla. Meni stavke se štikliraju obično kada predstavljaju neki fleg, odnosno Bulovu promenljivu: npr. prikazuje se/ne prikazuje se neki element korisničkog interfejsa.



Sl. 5a - Izgled menija IDR_MAINFRAME.



Sl. 5b - Izgled menija IDR_PRVITYPE.

U Doc/View aplikaciji postoje inicijalno dva menija. Jedan ima resursni identifikator IDR_MAINFRAME, a drugi IDR_PRVITYPE (ovo ime zavisi od imena aplikacije). Prvi meni obrađuje klasa CMainFrame, i on je aktivan kada u aplikaciji ne postoji ni jedan otvoreni dokument. Drugi meni obrađuje klasa CPrviView i on je aktivan kada postoji makar jedan otvoreni dokument. Ova podela je logična, jer su različite operacije moguće u ova dva slučaja. Na slikama 5a, 5b su prikazani ovi meniji.

Alternativni načini aktiviranja komandi su preko tastature i preko akceleratorske tabele. Ako se u nazivu meni stavke postavi znak ampersand (&) ispred nekog slova, to slovo će se u meni stavci pojavljivati podvučeno, što znači da je komandi moguće prići i preko tastature, otvaranjem tog nivoa menija, i kombinacijom tastera ALT+<podvučeno slovo>. Akcelerator za datu komandu se postavlja u resursnom editoru. Prosto se u tabelu akceleratora doda nova kombinacija tastera, zajedno sa identifikatorom komande koju treba da izvrši. Kod postavljanja akceleratora treba paziti da se ne uzme neki već zauzet, kao i da se standardni akceleratori, kao što su CTRL+C, CTRL+V, CTRL+S, itd. ne redefinišu novim značenjima - pomenuto je da je kod Windows aplikacija poželjno da sve aplikacije imaju jedan značajan podskup istih komandi.

3.2. Undo bafer

Kod Doc/View aplikacija uobičajeno postoji opcija Undo - tj. poništavanje poslednje unete izmene dokumenta. Sada ćemo ugraditi tu opciju u naš program. Undo je moguće napraviti na dva načina: da vraća samo poslednju izmenu, i da može da vrati unazad sve izmene od poslednjeg snimanja dokumenta. Mi ćemo, radi jednostavnosti, realizovati prvu varijantu. Treba uraditi sledeće:

1. U CPrviDoc klasu dodati public member varijable: `int undobuf, BOOL undoenabled`. Prva promenljiva pamti staru vrednost dokumenta (a naš dokument se sastoji samo od jedne int promenljive), dok druga predstavlja fleg koji određuje da li je Undo uopšte dozvoljen, odnosno, da li se u undo baferu nalazi neka smislena vrednost. U konstruktor ove klase dodati inicijalizaciju ovih promenljivih: `undobuf=0; undoenabled=FALSE;`
2. U CPrviDoc klasu dodati funkcije koje pristupaju undo baferu: `void NapuniUndoBafer()` i `void IzvrsiUndo()`. Te funkcije izgledaju ovako:

```
void CPrviDoc::NapuniUndoBafer(void)
{   undobuf = sadrzaj;
    undoenabled = TRUE;
}

void CPrviDoc::IzvrsiUndo(void)
{   if(undoenabled) sadrzaj = undobuf;
}
```

3. Funkcija NapuniUndoBafer treba da se poziva svuda u kodu gde dolazi do promene dokumenta, i to neposredno pre te promene. Dodaćemo poziv te funkcije u hendler OnKeyDown u View-u, pošto se tu dokument menja:

```
void CPrviView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{   // TODO: Add your message handler code here and/or call default
    GetDocument()->NapuniUndoBafer();
    ... //ostatak koda
}
```

4. Fleg undoenabled, koji je inicijalno FALSE, postavlja se na TRUE posle prvog upisa u undo bafer, tj. posle prve izmene dokumenta. Kada on treba da se vrati na vrednost FALSE? Prvo, kad se dokument snimi; drugo, kad se izvrši Undo. Prvi slučaj ćemo realizovati sada, a drugi kasnije, kad dodamo hendler za komandu Undo:

```
void CPrviDoc::Serialize(CArchive& ar)
{   if (ar.IsStoring())
    {   // TODO: add storing code here
        ar << sadrzaj;
```

```
        undoenabled=FALSE;
    }
    else
    ... //ostatak koda
}
```

5. Sada ćemo dodati hendlere za meni opciju Edit>Undo, koja već postoji kreirana u meniju IDR_PRVITYPE. Njen resursni identifikator je ID_EDIT_UNDO. U Class Wizard-u (CTRL+W) treba naći taj resursni identifikator, i generisati u klasi CPrviView oba ponuđena hendlera: i COMMAND, i UPDATE_COMMAND_UI. Kada to uradimo, Class Wizard će nam u mapu poruka ubaciti kod koji povezuje resursni ID ove meni opcije sa generisanim hendlerima (u view-u):

```
BEGIN_MESSAGE_MAP(CPrviView, CView)
    //{{AFX_MSG_MAP(CPrviView)
    ...
    ON_COMMAND(ID_EDIT_UNDO, OnEditUndo)
    ON_UPDATE_COMMAND_UI(ID_EDIT_UNDO, OnUpdateEditUndo)
    ...
    //}}AFX_MSG_MAP
    ...
END_MESSAGE_MAP()
```

Ostaje nam samo da "popunimo" prazne hendler funkcije OnEditUndo i OnUpdateEditUndo sa kodom koji obavlja te operacije:

```
void CPrviView::OnEditUndo()
{   // TODO: Add your command handler code here
    GetDocument()->IzvrsiUndo();
    GetDocument()->undoenabled=FALSE;
    Invalidate(FALSE);
}

void CPrviView::OnUpdateEditUndo(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    if(GetDocument()->undoenabled) pCmdUI->Enable(TRUE);
    else pCmdUI->Enable(FALSE);
}
```

Zašto se kod za obradu ove meni opcije nalazi baš u View-u? Rekli smo da View predstavlja interfejs dokumenta prema korisniku, znači View je zadužen za "hvatanje" poruka od korisnika, kada on nešto menja po dokumentu.

Command hendler poziva funkciju IzvrsiUndo iz dokumenta, koja vraća stari sadržaj. Nakon toga se onemogućava ponovno biranje ove opcije (do nove promene dokumenta), jer nema smisla. Na kraju se poziva funkcija Invalidate koja proglašava sadržaj prozora View-a nevažećim i izaziva slanje poruke WM_PAINT tom prozoru, čime se on ponovo iscrtava, sa novom vrednošću dokumenta. Update hendler čita sadržaj flega undoenabled i na osnovu njega dozvoljava ili onemogućava opciju Edit>Undo.

Ovde je još interesantno primetiti način na koji se iz View-a pristupa dokumentu - koristi se funkcija CView::GetDocument() koja vraća pointer na dokument koji je vezan sa View-om iz koga se poziva.

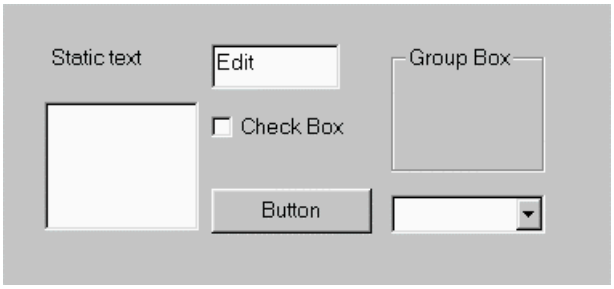
3.3. Modalni dijalog

Dijalog (klasa CDialog) je specijalna vrsta prozora (klasa CWnd) koja omogućava jednostavan unos i prikazivanje podataka. Po načinu funkcionisanja postoje dve vrste dijaloga: modalni i nemodalni. Razlika između njih je u sledećem: kada korisnik aktivira modalni dijalog, nije u mogućnosti da koristi ostatak aplikacije sve dok ga ne zatvori. Sa druge strane, kada se otvori nemodalni dijalog, moguće ga

je držati otvorenog i u isto vreme koristiti druge elemente aplikacije. Čak je moguće da istovremeno egzistira više instanci jednog istog nemodalnog dijaloga.

Praktično svu funkcionalnost dijalogu daju tzv. *kontrole*. Kontrole su gotovi gradivni elementi korisničkog interfejsa koji obavljaju tačno predviđenu funkciju. Strogo uzevši, sve kontrole predstavljaju male prozore (izvedene su iz klase CWnd). Kontrole realizuju operacije unosa i prikaza podataka. Najčešće korišćene kontrole su prikazane na slici 6:

- Statički tekst je najprostoja kontrola, koja nema nikakvu drugu ulogu nego da stalno prikazuje jedan isti tekst. Koristi se za obeležavanje značenja drugih kontrola.
- Edit boks, ili polje za unos i prikazivanje, je verovatno najvažnija kontrola. Koristi se za unos alfanumeričkih podataka. Pored osnovnog izgleda i funkcije, a to je unos podataka, moguće je podesiti izgled ove kontrole tako da se "uklapa" u pozadinu kao i zadati read-only režim čime se dobija jednostavan objekat za prikazivanje podataka.
- Grupa nema nikakvu specijalnu funkciju, i po tome je slična statičkom tekstu. Koristi se samo za vizuelno grupisanje kontrola. Npr. ako dijalog podržava nekoliko raznorodnih operacija poželjno je razdvojiti sve kontrole u odvojene grupe, po logičkim celinama, radi lakšeg korišćenja.
- List boks je kontrola za biranje jedne ili više stavki od nekoliko ponuđenih. Sve moguće opcije su prikazane u listi, a korisnik selektuje one koje mu odgovaraju.
- Ček boks se koristi za realizaciju flegova. Može da prikazuje dva stanja: prazno i precrtano.
- Dugme je osnovna kontrola za izvršenje operacija. Dugme nema sebi pridruženo stanje kao druge kontrole, ali zato pritisak na dugme izaziva akciju, tj. pozivanje pridruženog hendlera.
- Kombo boks služi za selektovanje tačno jedne od nekoliko ponuđenih opcija. Kada se pritisne "strelica nadole" prikazuje se spisak mogućih opcija. Ova kontrola se zbog jednostavne upotrebe koristi mnogo češće nego list boks.



Sl. 6 - Najčešće korišćene kontrole dijaloga: statički tekst, edit boks, grupa, list boks, ček boks, dugme, kombo boks.

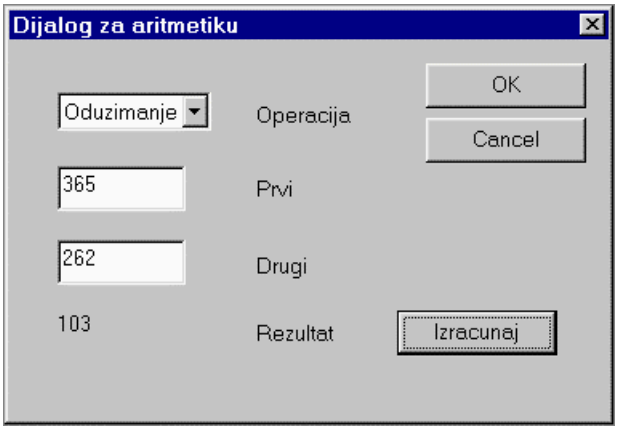
Pored ovih postoje još mnoge druge kontrole čija je namena specifična pa ih koristimo samo u posebnim slučajevima. Postoji mogućnost da programer definiše sopstvene kontrole, tzv. ActiveX kontrole, koje se u principu koriste isto kao u ove ugrađene. Kreiranje ActiveX kontrola spada u naprednije tehnike Windows programiranja, pa o tome ovde neće biti reči. Primetimo da se i na ovaj način, kroz ugrađene kontrole, realizuje zamisao o što sličnijem korisničkom interfejsu svih Windows aplikacija. Tako korisnik koji zna da koristi neku kontrolu, npr. kombo boks, može da očekuje da će i u nekoj drugoj aplikaciji da se ta kontrola ponaša isto.

Kontrole se dodaju dijalogu u resurs editoru, a zatim im se u Class Wizardu pridružuju member varijable (članice klase konkretnog dijaloga) koje služe za čitanje i menjanje stanja kontrole, kao i hendleri koji obrađuju akcije nad kontrolama. U najjednostavnijem slučaju, koji ćemo mi koristiti, kontrole će biti "pasivne", odnosno njihove pridružene member varijable će biti tipa Value (odnosno nekog od C++ tipova int, double, CString, ...) a ne tipa Control. Jedina "aktivna" kontrola će biti dugme, i za nju ćemo definisati hendler funkciju koja se poziva posle pritiska na dugme.

U dijalogu postoje dva default dugmeta, OK i Cancel. Hendleri za ova dva dugmeta se nalaze realizovani u klasi CDialog. Po defaultu, oni ne rade ništa, osim što imaju različite povratne vrednosti na osnovu čega sa mesta gde je pozvan dijalog može da se utvrdi da li je korisnik pritisnuo jedno ili drugo dugme. Ukoliko želimo da imamo trensfer podataka između dijaloga i ostatka aplikacije moramo da definišemo hendlere za njih u konkretnoj klasi dijaloga. Predviđeni su da se koriste na sledeći način: kada npr. korisnik završi sa unosom podataka, i želi da novi unešeni podaci budu validni, on pritiska OK. Hendler OnOK, u klasi konkretnog dijaloga, treba da obavi upis tih novounešenih vrednosti na neko mesto gde će se dalje čuvati (npr. u objekat aplikacije, ili u dokument). U slučaju da korisnik želi

da izađe iz dijaloga bez izmena, pritisnuće Cancel, posle čega će se pozvati hendler OnCancel iz klase konkretnog dijaloga. On ne treba da upisuje te nove vrednosti, već samo da obavi potrebno čišćenje dinamički kreiranih objekata pre zatvaranja dijaloga.

Hoćemo da napravimo modalni dijalog koji će obavljati funkciju jednog mini-kalkulatora (vidi sliku 7). Osnovne funkcije dijaloga će biti sledeće: unosiće se dva broja, operandi; biraće se koja računska operacija treba da se obavi; zahtevanjem proračuna ispisivaće se rezultat. Znači, koristićemo sledeće kontrole: za unos operanda trebaće nam dva edit boksa; za ispis rezultata jedan read-only edit boks; za biranje operacije jedan kombo boks; za startovanje proračuna jedno dugme; naravno, imaćemo pored svake kontrole statički tekst koji će objašnjavati ulogu kontrole. Dijalog treba da izgleda kao na slici 7.



Sl. 7 - Modalni dijalog za aritmetiku.

Da bi se napravio ovaj dijalog treba uraditi sledeće:

1. Kreira se dijalog u resursima (ResourceView>Dialog>Insert Dialog). U property-jima za taj dijalog postavi se resursni identifikator za dijalog: IDD_DLGARITM. Da se naslov dijalogu (caption), npr. "Dijalog za aritmetiku".
2. Doda se kombo boks. Sa tulbara koji sadrži kontrole dovuče se kontrola kombo boksa i postavi na željeno mesto. Kada se otvore property-ji za ovu kontrolu (klik desnim tasterom miša na kontrolu, properties), postave se njeni parametri: Styles>Type = Drop List, Styles>Sort = isključeno, General>ID = IDC_OPERACIJA, Data = Sabiranje, Oduzimanje, Mnozenje, Deljenje. Kod popunjavanja Data polja, u novi red se prelazi sa CTRL+ENTER.
3. Dodaju se edit boksovi za unos operanada. Dovuku se dva edit boksa sa tulbara sa kontrolama, postave na željeno mesto, i popune njihovi property-ji. Samo im treba postaviti resursne identifikatore na IDC_PRVI i IDC_DRUGI.
4. Doda se edit boks za prikaz rezultata. Dovuče se sa tulbara sa kontrolama, postavi na željeno mesto, i popune mu se propery-ji: resursni identifikator = IDC_REZ, Styles>ReadOnly = uključeno, Styles>Border = isključeno.
5. Doda se dugme za izvršenje operacije. Dovuče se sa tulbara sa kontrolama i postavi na željeno mesto. U njegovim propery-jima (klik desnim tasterom miša na dugme, properties) postavi mu se resursni identifikator na IDC_IZRACUNAJ.
6. Dodaju se statički tekstovi pored svake kontrole koji opisuju njihovo značenje: "operacija", "prvi", "drugi", "rezultat".

Ovim je završeno kreiranje dijaloga u resursima. Kontrole se mogu poravnati jedne sa drugim na više načina. Najpre, moguće je uključiti mrežu (grid) koja omogućava da se kontrole postavljaju samo u čvorove mreže pa se tako lakše ravnaju (Layout>Guide Settings). Drugi način je da se više kontrola selektuje istovremeno (CTRL+klik levim tasterom miša), i zatim klikne desnim tasterom miša, posle čega se iz kontekstnog menija izabere stavka "Align Left Edges" ili "Align Top Edges". Komanda se odnosi na sve selektovane kontrole, a one se ravnaju prema poslednjoj izabranoj (njena oznaka da je selektovana je drugačije boje od ostalih).

Nakon ovoga prelazimo na kreiranje koda koji koristi ove resurse.

7. Sa CTRL+W pređemo u Class Wizard. Na pitanje da li želimo kreiranje nove klase za ovaj dijalog odgovorimo potvrdno, i pustimo da Wizard kreira novu klasu imena CDlgAritmetika.
8. U Class Wizard-u kreiranoj klasi CDlgAritmetika dodamo promenljive članice klase (sekcija *Member variables*):

IDC_OPERACIJA	m_operacija	Value	int
IDC_PRVI	m_prvi	Value	double
IDC_DRUGI	m_drugi	Value	double
IDC_REZ	m_rez	Value	double

9. U sekciji *Message Maps* Class Wizard-a dodamo hendlere koji obrađuju poruke u ovom dijalogu - funkcije članice ove klase:

IDC_IZRACUNAJ	BN_CLICKED	OnIzracunaj()
CDlgAritmetika	WM_INITDIALOG	OnInitDialog()

Kada izađemo iz Class Wizard-a sa OK, on će kreirati u kodu potrebne izmene - dodaće member varijable kao i deklaracije, prazna tela funkcija i stvake u mapi poruka za potrebne hendlere.

10. Sledeći korak je da "popunimo" prazna tela funkcija koje je generisao Class Wizard. Prilikom svakog otvaranja dijaloga njemu se šalje poruka WM_INITDIALOG. Hendler funkcija koja obrađuje ovu poruku, OnInitDialog(), treba da sedrži kod za inicijalizaciju kontrola dijaloga.

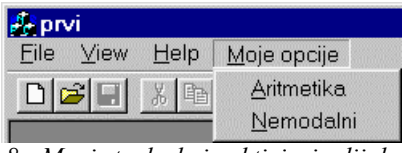
```
BOOL CDlgAritmetika::OnInitDialog()
{   ...
    m_operacija=0;
    m_prvi=0;
    m_drugi=0;
    m_rez=0;
    m_gdejekod=0;
    UpdateData(FALSE);
    ...
}
```

Transfer podataka između kontrola i njima pridruženih member varijabli klase obavlja se MFC funkcijom UpdateData(SmerTransfera) iz klase prozora CWnd. Ako je argument SmerTransfera = FALSE, upisuje se iz varijabli u kontrole i osvežava se izgled dijaloga. Ako je argument SmerTransfera = TRUE, čitaju se tekuće vrednosti kontrola i upisuju u member varijable.
Pored hendlera OnInitDialog() koji obrađuje poruku WM_INITDIALOG, dodali smo i hendler OnIzracunaj() koji obrađuje pritisak na dugme Izracunaj (IDC_IZRACUNAJ). Taj hendler sadrži akciju koja treba da se obavi na pritisak dugmeta - u ovom slučaju proračun:

```
void CDlgAritmetika::OnIzracunaj()
{   ...
    UpdateData(TRUE);
    switch(m_operacija)
    {   case 0:
            m_rez=m_prvi+m_drugi;
            break;
        case 1:
            m_rez=m_prvi-m_drugi;
            break;
        case 2:
            m_rez=m_prvi*m_drugi;
            break;
        case 3:
            if(m_drugi==0)
            {   AfxMessageBox("Deljenje sa nulom!");
                m_rez=0;
            }
            else m_rez=m_prvi/m_drugi;
            break;
    }
    UpdateData(FALSE);
    ...
}
```

Pre izračunavanja vršimo upis unetih podataka iz kontrola u member varijable pozivom funkcije UpdateData(TRUE), a nakon obavljenog proračuna (rezultat se smešta u m_rez) radimo obrnuto - upisujemo u kontrole vrednosti member varijabli pozivom funkcije UpdateData(FALSE).

11. Nakon ovoga imamo potpuno spreman dijalog. Jedino još treba da se poveže sa ostatkom aplikacije. Hoćemo da se ovaj dijalog otvara kada ga korisnik izabere iz glavnog menija. Zato u resursima, u meniju IDR_MAINFRAME (to je meni koji je aktivan kada nije otvoren nijedan dokument) kreiramo novu grupu meni stavki, Moje opcije, i u njoj stavku Aritmetika, sa resursnim identifikatorom ID_MOJE_ARITM (slika 8):



Sl. 8 - Meni stavke koje aktiviraju dijaloge.

12. Sledeće šta treba uraditi je da se u Class Wizard-u generiše COMMAND hendler za ovu stavku, OnMojeAritm(). Hendler treba da se nalazi u klasi CMainFrame, jer ta klasa predstavlja, između ostalog, korisnički interfejs u slučaju kad nije otvoren nijedan dokument (napomena: kada je otvoren makar jedan dokument, njemu pridružena klasa View ima ulogu korisničkog interfejsa). Kada se napravi ovaj hendler, Class Wizard će u fajlu MainFrm.cpp generisati prazno telo funkcije koje treba popuniti:

```
void CMainFrame::OnMojeAritm()
{   ...
    CDlgAritmetika dlg;
    dlg.DoModal();
    ...
}
```

COMMAND hendler meni stavke Moje Opcije>Aritmetika treba da izvrši aktiviranje dijaloga. Pošto je reč o modalnom dijalogu, kontrola toka programa se "zaustavlja" na tom mestu dok se ne zatvori dijalog, pa se on kreira prosto kreiranjem njegovog objekta, a aktivira pozivom funkcije DoModal(). Kada korisnik zatvori dijalog, program nastavlja da se izvršava od prve naredbe posle DoModal(). Da bi klasa CMainFrame "videla" klasu CDlgAritmetika, u fajl MainFrm.cpp treba dodati i direktivu za uključenje zaglavlja klase dijaloga CDlgAritmetika:

```
#include "DlgAritmetika.h"
```

13. Ukoliko želimo da korisnik ima brz pristup do ove opcije, poželjno je definisati *akcelerator*. Akcelerator je kombinacija tastera koja izaziva pozivanje odgovarajuće hendler funkcije, baš kao da je izabrana neka stavka menija. Standardne Windows operacije koje Application Wizard generiše automatski prilikom kreiranja aplikacije (New, Open, Save, Copy, Paste, ...) imaju svoje pridružene default akceleratore (CTRL+N, CTRL+O, CTRL+S, CTRL+C, CTRL+V, ...). Već je napomenuto da kada definišemo svoj akcelerator treba da vodimo računa da ne preklopimo neki od već iskorišćenih, a da takođe nije ni poželjno menjati značenja default akceleratora. Da bi definisali novi akcelerator za stavku menija Moje opcije>Aritmetika uđemo u resurse i izaberemo sekciju Accelerator>IDR_MAINFRAME. Dobićemo akceleratorsku tabelu, odnosno spisak već definisanih kombinacija tastera. Pritiskom na desni taster miša dobijamo kontekstni meni koji iz koga izaberemo stavku "New Accelerator". Nakon toga se otvara dijalog za definisanje akceleratora. U polje ID unećemo resursni identifikator stavke menija koju dupliramo akceleratorom: ID_MOJE_ARITM. U polje Key (vodeći računa da je od modifier-a na desnom delu dijaloga uključen samo Ctrl, i da tip akceleratora bude VirtKey) upišemo R. Alternativno, možemo da pritisnemo dugme "Next Key Typed" nakon čega će dijalog očekivati da pritisnemo kombinaciju tastera koju želimo da uvedemo (CTRL+R). Kada izađemo iz resursnog editora, biće kreiran novi akcelerator CTRL+R koji poziva hendler OnMojeAritm(), tj. otvara modalni dijalog Aritmetika isto kao da je izabrana meni stavka Moje opcije>Aritmetika.

3.4. Nemodalni dijalog

Kreiranje nemodalnog dijaloga počinje u resursima:

1. Slično kao i kod modalnog, napravimo novi dijalog (ResourceView>Dialog>Insert Dialog), damo mu neki resursni ID, npr. IDD_DLGNEMOD, i naslov (*caption*), npr. "Nemodalni dijalog". Od

osobina (properties), postavimo Styles>Style = "Overlapped", i More Styles>Visible = uključeno. Nemodalnom dijalogu ćemo dodati samo jedno dugme, "Akcija", sa resursnim identifikatorom IDC_AKCIJA. Kada se ovo uradi, dijalog treba da izgleda kao na slici 9.

2. Kada su napravljeni resursi za ovaj dijalog, prelazimo u Class Wizard (CTRL+W) da formiramo klasu za ovaj dijalog, pod imenom CDlgNemodalni. Klasa CDlgNemodalni neće imati ni jednu member varijablu (pošto nismo definisali kontrole koje to zahtevaju), već samo sledeće hendlere:

IDC_AKCIJA	BN_CLICKED	OnAkcija()
CDlgNemodalni	PostNcDestroy	PostNcDestroy()
IDOK	BN_CLICKED	OnOK()
IDCANCEL	BN_CLICKED	OnCancel()

Hendler OnAkcija se poziva kada korisnik klikne mišem na dugme Akcija. Pošto je reč samo o primeru, neka ta akcija bude samo ispisivanje poruke:

```
void CDlgNemodalni::OnAkcija()
{   // TODO: Add your control notification handler code here
    AfxMessageBox("Neka akcija");
}
```

Način kreiranja i uništavanja nemodalnog dijaloga se znatno razlikuje od načina kreiranja i uništavanja modalnog dijaloga. Nemodalni dijalog se kreira kao dinamički objekat C++ operatorom `new`, nakon čega se kreira i kao Windows objekat pozivom funkcije Create. Nakon njegovog kreiranja program nastavlja sa izvršavanjem (kod modalnog dijaloga program se ne nastavlja dok se ne zatvori dijalog). Zbog toga glavni prozor aplikacije više sa njim nema veze, pa se uništavanje ovog objekta mora obaviti u samoj klasi nemodalnog dijaloga (tzv. auto-cleanup). To se radi u hendleru PostNcDestroy, pošto je to poslednji u nizu hendlera koji se poziva prilikom gašenja prozora:

```
void CDlgNemodalni::PostNcDestroy()
{   //TODO:Add your specialized code here and/or call the base class
    delete this;
    //CDialog::PostNcDestroy();
}
```

Prolikom gašenja prozora (razmatramo opšti slučaj bilo kakvog prozora, bez obzira što mi konkretno imamo prozor nemodalnog dijaloga, jer je procedura opšteg karaktera), generiše se cela sekvenca prozorskih poruka, koja izgleda otprilike ovako:

1. Kada korisnik klikne na x u gornjem desnom uglu prozora, prozoru se šalje poruka WM_CLOSE.
2. Ova poruka, koja označava zahtev za gašenjem prozora, obrađuje se po defaultu na sledeći način. Poziva se funkcija DestroyWindow, koja uništava prozor i njemu pridružene objekte, i pri tome šalje prozoru koji se uništava dve poruke: prvo poruku WM_DESTROY, a zatim i WM_NCDESTROY.
3. Prozor prima poruku WM_DESTROY čime se poziva hendler OnDestroy koji označava da je klijentska oblast prozora uništena.
4. Prozor prima poruku WM_NCDESTROY čime se poziva hendler OnNcDestroy koji označava da je i neklijentska oblast prozora uništena. OnNcDestroy zatim poziva hendler PostNcDestroy.
5. PostNcDestroy je poslednji u nizu hendlera koji se pozivaju, i u trentku kada se on izvršava prozor više ne postoji kao Windows objekat, već samo kao običan C++ objekat. U ovom hendleru se taj C++ objekat uništava pozivom C++ operatora `delete this`. Po preporuci iz Visual C++ Help-a, sva čišćenja dinamičkih objekata kod prozora koji se gase na ovaj način treba obaviti baš u hendleru PostNcDestroy. Na ovaj način se prozoru dodaje osobina tzv. auto-cleanup-a, odnosno on sam postaje odgovoran za sopstveno čišćenje.

Spomenimo i da u slučaju gašenja aplikacije, WM_DESTROY poziva funkciju PostQuitMessage koja šalje poruku WM_QUIT. Kada aplikacija primi poruku WM_QUIT, izlazi iz petlje poruka i završava sa radom.

Pošto naš nemodalni dijalog poseduje i dva tastera Ok i Cancel, koji po defaultu pozivaju EndDialog (što je pogrešno kod nemodalnih dijaloga) treba dodati njihove hendlere i prepraviti njihovo ponašanje tako da pozivaju DestroyWindow čime se aktivira gore prikazani mehanizam uništavanja sa auto-cleanup-om:

```
void CDlgNemodalni::OnCancel()
{   // TODO: Add extra cleanup here
    DestroyWindow();
    //CDialog::OnCancel();
}

void CDlgNemodalni::OnOK()
{   // TODO: Add extra validation here
    DestroyWindow();
   //CDialog::OnOK();
}
```

Kako je napomenuto, kreiranje nemodalnog dijaloga treba da obezbedi da se on kreira i kao C++ objekat, i kao Windows objekat (tj. da ga sistem "registruje", odnosno da mu dodeli neke resurse). To se radi tako što u konstruktor nemodalnog dijaloga dodamo poziv funkcije Create:

```
CDlgNemodalni::CDlgNemodalni(CWnd* pParent /*=NULL*/)
    : CDialog(CDlgNemodalni::IDD, pParent)
{   //{{AFX_DATA_INIT(CDlgNemodalni)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
    Create(IDD_DLGNEMOD);
}
```

Na ovaj način, kada negde u programu kreiramo ovakav dijalog pomoću new operatora obaviće se cela procedura korektno.

Važna napomena - nemodalni dijalog u programu:

1. nikako ne treba eksplicitno brisati operatorom `delete`, jer se tako on ne uništava kao Windows objekat (sistem i dalje čuva njegove podatke - to se zove "curenje resursa") već samo kao C++ objekat (poziva se njegov destruktor).
2. uvek treba alocirati dinamički operatorom `new`, a nikako statički kao npr. `CDlgNemodalni dlg;` jer kad tok programa izađe iz oblasti gde je taj objekat definisan on se briše pozivom destruktora, što u krajnjoj instanci izaziva iste probleme opisane pod (1).

Drugim rečima, mora da se obezbedi kompletna regularna procedura gašenja sa svim već opisanim porukama koje prethode uklanjanju tog Windows objekta.

Sl. 9 - Nemodalni dijalog.

Ovim je nemodalni dijalog postao potpuno funkcionalan, samo ga još treba povezati sa ostatkom aplikacije, odnosno negde u kodu aplikaciju izvršiti njegovo kreiranje. Uvešćemo, na isti način kao i kod modalnog dijaloga, novu stavku glavnog menija čijim pozivom će se kreirati ovaj dijalog: uđemo u resurse, u meniju IDR_MAINFRAME, pod Moje opcije dodamo stavku "Nemodalni" sa identifikatorom ID_MOJE_NEMOD. Zatim u Class Wizardu dodamo hendler (u CMainFrame klasu) koji obrađuje ovaj događaj, OnMojeNemod:

```
void CMainFrame::OnMojeNemod()
{   // TODO: Add your command handler code here
    CDlgNemodalni *pdlg;
```

```
    pdlg=new CDlgNemodalni();
}
```

Ovim je aplikacija spremna za kompajliranje i startovanje.

Domaći zadaci.

1. Realizovati Undo operaciju sa proizvoljnim brojem vraćanja unazad. Kada se dokument snimi, briše se "istorija" izmene dokumenta. Obratiti naročitu pažnju na pravilnu dinamičku alokaciju i dealokaciju undo bafera.
2. Obezbediti brojač nemodalnih dijaloga. Brojač treba da na jedinstven način označi svaki otvoreni nemodalni dijalog, počev od 1 pa na dalje. Broj nemodalnog dijaloga treba da se ispisuje u title baru pored naziva dijaloga, npr. "Nemodalni dijalog 7". Uputstvo: brojač otvorenih dijaloga je globalna karakteristika aplikacije, pa taj podatak treba čuvati ili u objektu aplikacije, ili u glavnom prozoru aplikacije, ili ga deklarisati kao statičku member varijablu (`static`) klase nemodalnog dijaloga. Treba izmeniti konstruktor nemodalnog dijaloga tako da može da pročita tu vrednost i da je inkrementira.

4. Statusna linija. Tulbar

Posle menija i dijaloga, obradićemo još dva elementa GUI (grafičkog korisničkog interfejsa) - statusnu liniju (*status bar*) i tulbar (*tool bar* - paletu sa alatkama). Statusna linija služi za prikaz tekućeg stanja aplikacije - statusa, i to samo najbitnijih stvari koje treba da budu stalno vidljive, dok se tulbar koristi za brzo pristupanje najčešće upotrebljavanim komandama.

MFC aplikacija po defaultu, kada se kreira Application Wizard-om, poseduje jednu statusnu liniju i jedan tulbar sa dupliranim standardnim akcijama File Open, File New, File Save, i još nekim. Oni postoje u resursima, a njihovi objekti su member varijable klase CMainFrame (znači "vlasništvo" glavnog prozora aplikacije). Kreiraju se prilikom inicijalizacije aplikacije, odnosno prilikom kreiranja glavnog prozora aplikacije, u funkciji CMainFrame::OnCreate.

4.1. Statusna linija

Statusna linija se sastoji od više polja. Prvo polje se uvek koristi za prikaz teksta, najčešće objašnjenja tekuće komande (po defaultu se prikazuje tekst unet u polje "prompt" meni stavke prilikom njenog definisanja u resursima), mada to može da se redefiniše po potrebi, funkcijom SetText. Ostala polja su rezervisana za prikaz flegova stanja aplikacije. Po defaultu postoje tri takva polja, koja prikazuju da li su uključeni tasteri Caps Lock, Num Lock i Scroll Lock. Ta polja mogu da prikazuju string koji je definisan u resursima, ili da budu prazna. I ovo ponašanje može da se redefiniše, pa je moguće, uz malo dodatnog truda, prikazati proizvoljan tekst ili čak crtez.

Mi ćemo statusnoj liniji aplikacije dodati jedno novo polje, koje će da prikazuje da li naša aplikacija Prvi radi u REFRESH režimu, odnosno da li se View automatski osvežava. Ovaj fleg se nalazi u klasi CPrviView (member varijabla refresh) kada je otvoren makar jedan dokument, odnosno u klasi CPrviApp (member varijabla m_app_refresh) kada nije otvoren ni jedan dokument.

1. U resursima ćemo dodati string koji će se ispisivati u polju. U resursima izabrati String Table i dodati string "REFRESH" sa resursnim identifikatorom ID_INDICATOR_REF. String će se naći u grupi stringova za ispis na statusnoj liniji, zajedno sa ostalim ID_INDICATOR_NESTO.
2. U klasi CMainFrame, u fajlu MainFrm.cpp, dodati u listu indikatora i naš indikator za refresh:

```
static UINT indicators[] =
{   ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
    ID_INDICATOR_REF,
};
```

3. Kontrolu nad poljem statusne linije ima njen hendler. Hendleri se za polja statusne linije ne mogu dodavati preko Class Wizard-a, već moraju ručno. U mapi poruka (na kraju MESSAGE MAP sekcije oivičene makroima BEGIN_MESSAGE_MAP i END_MESSAGE_MAP) u CPrviView klasi dodaćemo vezu između resursnog identifikatora polja statusne linije i njene hendler funkcije koju tek treba da napišemo:

```
BEGIN_MESSAGE_MAP(CPrviView, CView)
    ...
    //rucno
    ON_UPDATE_COMMAND_UI(ID_INDICATOR_REF,OnUpdateStatusRefresh)
END_MESSAGE_MAP()
```

Treba biti naročito pažljiv pri ručnom dodavanju hendlera. Delovi koda obojeni sivo su delovi koda koje je zabranjeno menjati, jer njih generiše Class Wizard - ubacivanjem koda na ta mesta vrlo je verovatno da aplikaciju nećemo moći više da iskompajliramo.

U isti fajl (CPrviView.cpp) dodaćemo i telo hendlera OnUpdateStatusRefresh:

```
void CPrviView::OnUpdateStatusRefresh(CCmdUI *pCmdUI)
{   if(refresh) pCmdUI->Enable(TRUE);
    else pCmdUI->Enable(FALSE);
}
```

4. Hendler ćemo deklarisati u CPrviView.h fajlu, zajedno sa ostalim hendlerima ali van "zasivljenog" dela koji je zabranjen za editovanje:

```
// Generated message map functions
    protected:
    //{{AFX_MSG(CPrviView)
        ...
    //}}AFX_MSG
    //rucno
        afx_msg void OnUpdateStatusRefresh(CCmdUI *pCmdUI);
    DECLARE_MESSAGE_MAP()
```

Ovim smo definisali hendler koji obrađuje polje refresh statusne linije kada je otvoren dokument. Svaka izmena flega refresh će se, preko update hendlera, odslikati na prikaz tog polja.

5. Da bi imali kompletnu funkcionalnost treba dodati isti takav hendler i u klasu CMainFrame (koraci 3, 4). Taj hendler će se aktivirati kada nije otvoren ni jedan dokument. U fajlu MainFrm.cpp izmenićemo, na isti način kao i u view-u, mapu poruka:

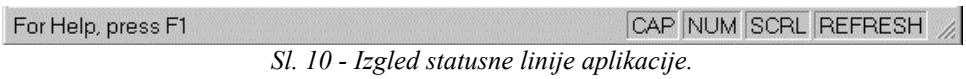
```
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    ...
    //rucno
    ON_UPDATE_COMMAND_UI(ID_INDICATOR_REF,OnUpdateStatusRefresh)
END_MESSAGE_MAP()
```

Dodaćemo telo hendlera:

```
void CMainFrame::OnUpdateStatusRefresh(CCmdUI *pCmdUI)
{   CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    if(pApp->m_app_refresh) pCmdUI->Enable(TRUE);
    else pCmdUI->Enable(FALSE);
}
```

I u fajlu MainFrm.h deklaraciju hendlera:

```
// Generated message map functions
    protected:
    //{{AFX_MSG(CMainFrame)
        ...
    //}}AFX_MSG
    //rucno
        afx_msg void OnUpdateStatusRefresh(CCmdUI *pCmdUI);
    DECLARE_MESSAGE_MAP()
```



Sl. 10 - Izgled statusne linije aplikacije.

Na ovaj način smo završili sa dodavanjem polja statusnoj liniji. Kada se program prevede, ona treba da izgleda kao na slici 10. Paljenjem i gašenjem opcije refresh (pritiskom na desni taster miša, u dokumentu) menja se sadržaj flega refresh, što se odražava na izgled statusne linije. Kada se dokument zatvori, vrednost refresh se prepiše u m_app_refresh, i dalje se prikazuje ta vrednost sve dok se ne otvori novi dokument.

4.2. Tulbar

Tulbar (*toolbar* - paleta sa alatkama) je element grafičkog korisničkog interfejsa koji služi za brzo pristupanje često korišćenim opcijama. U MFC aplikaciji po defaultu postoji već jedan, sa dupliranim meni opcijama za otvaranje, snimanje dokumenta, i još nekim (slika 11).

Sl. 11 - Izgled glavnog tulbara aplikacije.

Od kontrola, na tulbar mogu da se stave samo tasteri (*button*). Oni, međutim, mogu da se iskoriste na dva načina: (1) taster može da pokrene neku akciju pozivanjem svog hendlera, (2) taster može da se eksplicitno definiše (u kodu) kao fleg i tako i da se koristi.

Napravićemo jedan novi tulbar, sa dva tastera - od svake vrste po jedan. Jedan taster će služiti da otvara modalni dijalog za aritmetiku, a drugi da kontroliše fleg refresh.

1. Prvi korak u kreiranju tulbara je da se on definiše u resursima. Neka mu je resursni identifikator IDR_MYTOOLS. Dodaćemo mu dva tastera, ID_MYTOOL_ACTION (obeležen sa !), i ID_MYTOOL_REFRESH (obleležen sa R). Videti sliku 12. U property-jima za tastere se pored resursnog ID-a, može postaviti i tekst koji se ispisuje u statusnoj liniji kada korisnik pređe mišem preko tastera (polje prompt). Možemo da stavimo npr. "Action\nAction" i "Refresh\nRefresh".

Sl. 12 - Izgled novog tulbara.

2. U MainFrm.cpp treba dodati kod koji preslikava resursne identifikatore tastera u niz tastera:

```
static UINT BASED_CODE buttons[] =
{   ID_MYTOOL_REFRESH,
    ID_MYTOOL_ACTION
};
```

a u MainFrm.h, u deklaraciju klase CMainFrame, protected member varijablu tipa tulbara:

```
protected:  // control bar embedded members
    ...
    CToolBar m_wndMyTools;
```

3. Kao i kod većine MFC objekata, tulbar nije dovoljno deklarisati samo kao C++ objekat. Potrebno je da se on kreira propisno, pozivom API/MFC funkcija za kreiranje objekata i njihovu inicijalizaciju. Kreiranje tulbara ćemo obaviti u OnCreate funkciji CMainFrame klase (fajl MainFrm.cpp). Ta funkcija se poziva prilikom inicijalizacije glavnog prozora aplikacije. Prvo kreiramo objekat funkcijom Create, zatim učitavamo resurse pomoću LoadToolBar, i na kraju vršimo inicijalizaciju i podešavanja izgleda tulbara funkcijama EnableDocking i SetButtonStyle:

```
//MyTools
m_wndMyTools.Create(this,CBRS_RIGHT | CBRS_TOOLTIPS | CBRS_FLYBY |
    WS_VISIBLE);
m_wndMyTools.LoadToolBar(IDR_MYTOOLS);
m_wndMyTools.EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndMyTools);
m_wndMyTools.SetButtonStyle(
    m_wndMyTools.CommandToIndex(ID_MYTOOL_REFRESH),
    TBBS_CHECKBOX);
```

Argumenti ovih funkcija imaju sledeće značenje: prvi argument funkcije Create određuje za koji prozor će biti vezan tulbar - u ovom slučaju za glavni prozor aplikacije. Drugi argument je skup flegova koji određuju osnovne karakteristike tulbara: da bude inicijalno zakačen za desnu ivicu, da može da prikazuje tekst zapisan u prompt polju tastera kada se mišem pređe preko njega, i da je inicijalno vidljiv. Fleg prosleđen funkciji EnableDocking određuje gde će tulbar moći sve da se zakači - u ovom

slučaju za bilo koju ivicu prozora. U SetButtonStyle smo promenili default ponašanje tastera Refresh u "checkbox" - kada želimo da se taster ponaša kao fleg moramo da direktno kontrolišemo kada je taster u pritisnutom, odnosno, otpuštenom položaju, što nam "checkbox" stil omogućava. Stil tastera Action nismo menjali jer je on po defaultu TBBS_BUTTON, odnosno običan taster koji poziva komandu kada se pritisne i kod koga ne kontrolišemo stanje u kom se nalazi (pritisnut/otpušten).

4. Sada dolazi na red pisanje hendlera koji obrađuju događaje vezane za tulbar. Hendlere dodajemo na standardan način, prvo ih definišemo u Class Wizard-u, a zatim popunjavamo prazne funkcije koje je generisao Class Wizard. Napravićemo prvo hendlere u CMainFrame klasi. Oni će biti pozivani u slučaju da (1) nije otvoren ni jedan dokument, (2) da ne postoji odgovarajući hendler u CPrviView klasi kada je neki dokument otvoren. Tasteri koji su predviđeni da rade u stilu TBBS_BUTTON zahtevaju samo command hendler, a TBBS_CHECKBOX tasteri i command i update hendler:

```
void CMainFrame::OnMytoolAction()
{   // TODO: Add your command handler code here
    CDlgAritmetika dlg;
    dlg.DoModal();
}

void CMainFrame::OnMytoolRefresh()
{   // TODO: Add your command handler code here
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    pApp->m_app_refresh=!(pApp->m_app_refresh);
}

void CMainFrame::OnUpdateMytoolRefresh(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    if(pApp->m_app_refresh) pCmdUI->SetCheck(TRUE);
    else pCmdUI->SetCheck(FALSE);
}
```

U command hendleru za taster Action samo se otvara modalni dijalog Aritmetika. Command hendler tastera Refresh invertuje fleg m_app_refresh u CPrviApp klasi. Update hendler tastera Refresh određuje kako će biti prikazivan taj taster, pritisnuto ili otpušteno, u zavisnusti od flega m_app_refresh.

5. Takođe moramo da se pobrinemo za slučaj kada je bar jedan dokument otvoren. Hendler za taster Action bi izgledao isto, pa ga nećemo ponovo pisati u klasi CPrviView - pozivaće se automatski ovaj iz CMainFrame-a. Međutim hendlere za taster Refresh treba ponovo napisati i u klasi CPrviView, jer oni sada treba da koriste fleg refresh iz CPrviView klase, a ne m_app_refresh:

```
void CPrviView::OnMytoolRefresh()
{   // TODO: Add your command handler code here
    refresh = !refresh;
}

void CPrviView::OnUpdateMytoolRefresh(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    if(refresh) pCmdUI->SetCheck(TRUE);
    else pCmdUI->SetCheck(FALSE);
}
```

Kao što se vidi, logika njihovog rada je ista kao i u klasi CMainFrame.

6. Tulbar možemo da prikazujemo i sklanjamo po potrebi. Ovo je naročito zgodno kada u aplikaciji imamo više njih, jer se na taj način povećava preglednost. Hoćemo da se vidljivost tulbara postavlja iz glavnog menija, u View sekciji, što je karakteristično za većinu Windows aplikacija (View sekcija u meniju je predviđena za upravljanje izgledom aplikacije). Dodaćemo, u oba menija (IDR_PRVITYPE i IDR_MAINFRAME), stavku View>MyTools sa resursnim ID-om ID_VIEW_MYTOOLS. Ako stavimo da resursni ID bude isti za oba menija biće dovoljno da

napišemo hendlere samo u klasi CMainFrame. Na taj način ćemo izbeći pisanje istog koda dva puta. Command hendler treba da odradi postavljanje flegova tulbara pri prikazivanju i sakrivanju:

```
void CMainFrame::OnViewMytools()
{   // TODO: Add your command handler code here
    BOOL bv=m_wndMyTools.GetStyle() & WS_VISIBLE;
    int ns=bv?SW_HIDE:SW_SHOWNORMAL;
    m_wndMyTools.ShowWindow(ns);
    RecalcLayout();
}
```

Update hendler kontroliše izgled meni stavke. Hoćemo da stavka bude štiklirana u slučaju da je tulbar vidljiv, odnosno da pored nje ne stoji ništa u slučaju da nije vidljiv:

```
void CMainFrame::OnUpdateViewMytools(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    BOOL bv=m_wndMyTools.GetStyle() & WS_VISIBLE;
    if(bv) pCmdUI->SetCheck(TRUE);
    else pCmdUI->SetCheck(FALSE);
}
```

Treba uočiti način na koji se postavlja vidljivost. Pošto je klasa tulbara izvedena iz klase CWnd, odnosno predstavlja "jednu vrstu prozora", koristimo funkcije iz te klase za postavljanje i čitanje karakteristika prozora.

Na ovaj način smo završili sa kreiranjem tulbara i možemo da prevedemo program. Tulbar koji smo napravili je moguće pomerati, kačiti za okvir glavnog prozora, koristiti i u "lebdećem" stanju kada nije prilepljen ni uz jednu ivicu, zatim može se postavljati u horizontalni i vertikalni položaj dok nije zakačen. Korišćenjen opcija iz View sekcije menija može se postavljati ili sakrivati. U zavisnosti od primene, moguće je ovu funkcionalnost ograničiti, kao u slučaju default tulbara aplikacije koji je uvek zakačen za gornju ivicu glavnog prozora i ne može da se pomera.

5. Tabovani dijalog. Klipbord

U aplikacijama koje poseduju veliki broj parametara za podešavanje zgodno je te parametre grupisati po logičkim celinama, ali u isto vreme im pristupati jedinstvenom komandom. Umesto da pravimo za svaku grupu parametara poseban dijalog, ili da sve parametre držimo u jednom dijalogu (što je ponekad nemoguće), u takovim slučajevima se koristi tabovani dijalog (*tabbed dialog box* - dijalog sa karticama). On omogućava da se u jednom dijalogu nađe više njih, a da se izbor trenutno vidljivog vrši preko kartice (*tab*), kao na slici 13. U drugom delu će biti reči o upotrebi klipborda (*clipboard*). Koncept klipborda postoji od najranijih verzija Windowsa, i ima naročiti značaj u aplikcijama sa Doc/View arhitekturom. Klipbord omogućava jednostavno prenošenje podataka između delova istog dokumenta, između različitih dokumenata u MDI aplikaciji, i između različitih aplikacija koje obrađuju isti tip dokumenata.

5.1. Tabovani dijalog

Tabovani dijalozi su, kao što je već rečeno, prvenstveno namenjeni za ažuriranje parametara aplikacije (*options, properties*). Kreiraćemo jedan takav dijalog, sa dva taba (kartice), a radi jednostavnosti svaki tab će da kontroliše promenu samo jedne promenljive tipa int. Tabovani dijalog može da se napravi, kao i običan dijalog, u dve varijante - kao modalni ili nemodalni. Mi ćemo napraviti nemodalni tabovani dijalog zato što želimo da omogućimo da bude stalno otvoren, čak i u toku korišćenja drugih delova aplikacije. Kod modalnog tabovanog dijaloga ažuriranje parametara se obavlja pri izlasku iz dijaloga pritiskom na taster OK, a kod nemodalnog ćemo morati da dodamo taster koji će služiti samo za to. Procedura kreiranja nemodalnog tabovanog dijaloga izgleda ovako:

1. U klasi aplikacije deklarišemo promenljive za svaki parametar i obezbedimo njihovu inicijalizaciju u za to predviđenoj funkciji:

```
// fajl prvi.h
class CPrviApp : public CWinApp
{   public:
    ...
        int m_par1;     // parametar 1
        int m_par2;     // parametar 2
        BOOL m_prop_en; // fleg - da li je otvoren tab. dijalog
    ...
}

// fajl prvi.cpp
BOOL CPrviApp::InitInstance()
{   ...
    m_par1=1;
    m_par2=2;
    m_prop_en=FALSE;
    ...
}
```

Tabovani dijalog je u MFC-u implementiran kao klasa CPropertySheet, a tab kao klasa CPropertyPage. Potrebno je napraviti svaki property page ponaosob, na način sličan kreiranju običnog dijaloga, a zatim i property sheet koji treba da ih objedini.

2. Kreiranje taba (*property page*) počinje u resursima. U Resource View-u izaberemo stavku Dialog, pritiskom desnog tastera na mišu otvorimo kontekstni meni. Iz tog menija izaberemo opciju Insert, i od ponuđenih stavki izaberemo Dialog>IDD_PROPPAGE_SMALL. Kreiranom dijalogu promenimo resursni ID u IDD_PP_1.
3. Dodajemo sledeće kontrole: jedan edit box (IDC_PAR1), taster za ažuriranje Update (IDC_UPDATE), i statički tekst pored edit box-a koji objašnjava njegovu svrhu: "Parametar 1".
4. Kada smo kreirali resurse, prelazimo u Class Wizard (CTRL+W) da bi kreirali klasu za ovaj dijalog. Class Wizard će nas pitati da li želimo za ovaj resurs da koristimo novu klasu ili već neku

postojeću. Kada izaberemo opciju "nova klasa", pojaviće se dijalog sa karakteristikama te nove klase koji treba da popunimo. Bitno je da se za osnovnu klasu (*base class*) izabere CPropertyPage. Ime nove klase neka bude npr. CPPPrva.

5. Klasi CPPprva ćemo u Class Wizard-u dodati member varijablu za edit box, m_par1 tipa int. Takođe ćemo mapi poruka dodati hendlere za inicijalizaciju i obradu pritiska na taster Update.

Member varijable:

IDC_PAR1	m_par1	Value	int

Hendleri:

IDC_UPDATE	BN_CLICKED	OnUpdate()
CPPPrva	WM_INITDIALOG	OnInitDialog()

Ponekad, kada za tim ima potrebe, dodaje se i hendler OnSetActive() koji se startuje kada se tab aktivira. On obično služi za osvežavanje sadržaja kontrola.

6. Napisaćemo hendlere:

```
BOOL CPPPrva::OnInitDialog()
{   ...
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    m_par1 = pApp->m_par1;
    UpdateData(FALSE);
    ...
}

void CPPPrva::OnUpdate()
{   ...
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    UpdateData(TRUE);
    pApp->m_par1 = m_par1;
}
```

U OnInitDialog se vrednost parametra 1 prepisuje iz objekta aplikacije u member varijablu edit kontrole, a zatim osvežava sadržaj kontrole. U OnUpdate se radi obrnuto - čita se sadržaj kontrole, i pročitana vrednost upisuje u objekat aplikacije. Ovakav način ažuriranja zahteva da korisnik posle svake promene u edit polju pritisne taster Update - ako to ne uradi vrednost se neće ažurirati.

7. Na ovaj način smo napravili potpuno funkcionalan tab. Hoćemo da imamo još jedan tab, pa ponavljamo korake 2-6. Neka mu je resursni ID IDD_PP_2, klasa CPPDruga, neka i on ima iste kontrole, member varijable i hendlere kao prvi tab (samo sa oznakom 2 umesto 1), i neka on ažurira promenljivu objekta aplikacije m_par2.
8. Prelazimo na kreiranje tabovanog dijaloga - property sheet-a, koji samo treba da objedini već postojeće tabove u jednu celinu. U resursima kreiramo još jedan property page, damo mu identifikator IDD_PS_PROPERTIES, i pomoću Class Wizard-a kreiramo njegovu klasu CPSProperties, izvedenu iz klase CPropertySheet.
9. U heder fajl PSProperties.h dodamo direktive za uključivanje zaglavlja za property page-ove:

```
// fajl PSProperties.h
#include "PPPrva.h"
#include "PPDruga.h"
```

10. U deklaraciji klase CPSProperties uvedemo dve nove public memeber varijable - pokazivače na objekte klasa property page-ova:

```
// Attributes
public:
    CPPPrva *m_pp1;
    CPPDruga *m_pp2;
```

11. Klasa CPSProperties nudi dva konstruktora - mi ćemo da koristimo drugi, koji kao prvi argument očekuje string. U taj konstruktor ubacimo kod za kreiranje property page-ova:

```
//u konstruktor
m_pp1=new CPPPrva;
m_pp2=new CPPDruga;
AddPage(m_pp1);
AddPage(m_pp2);
Create(AfxGetApp()->m_pMainWnd,
WS_SYSMENU|WS_POPUP|WS_CAPTION|WS_VISIBLE|DS_MODALFRAME);
```

12. Pošto je reč o nemodalnom dijalogu, treba obezbediti auto-cleanup, a to znači da treba u Class Wizard-u, u mapu poruka dodati i hendler PostNcDestroy u kome se vrši čišćenje pri zatvaranju tabovanog dijaloga:

```
void CPSProperties::PostNcDestroy()
{   //TODO:Add your specialized code here and/or call the base class
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    pApp->m_prop_en=FALSE;
    delete m_pp1;
    delete m_pp2;
    delete this;
    //CPropertySheet::PostNcDestroy();
}
```

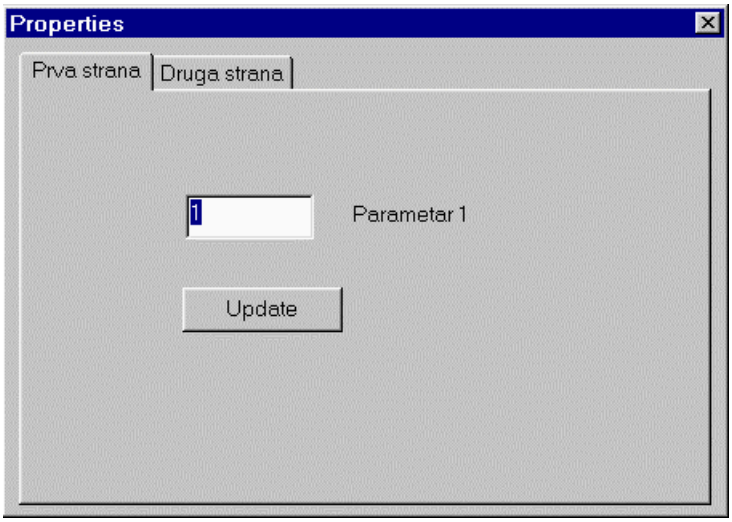
Sada smo obezbedili potpuno funkcionalan tabovani dijalog. Treba ga još povezati sa ostatkom aplikacije. Ubacićemo meni opciju koja će da otvara i zatvara tabovani dijalog Properties.

13. U resursima, u IDR_MAINFRAME meniju, pod File dodamo još jednu stavku Properties. Neka je njen resursni identifikator ID_FILE_PROPERTIES. U Class Wizardu dodamo command i update hendler za ovu opciju (u klasu CMainFrame) - OnFileProperties i OnUpdateFileProperties:

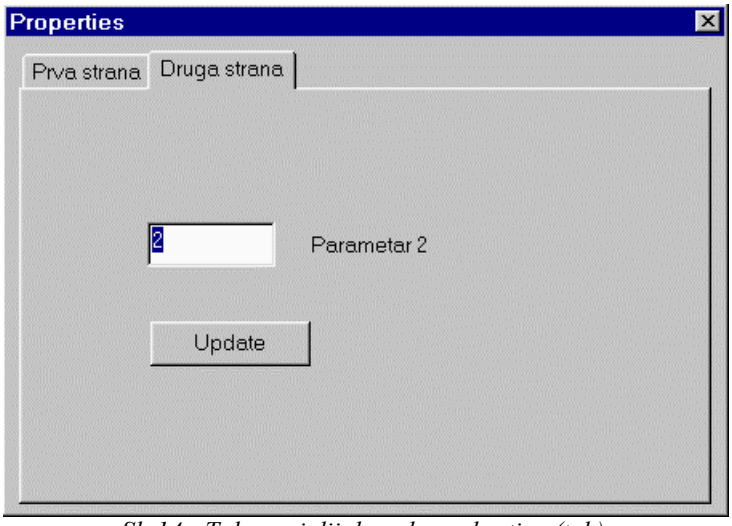
```
void CMainFrame::OnFileProperties()
{   CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    if(!pApp->m_prop_en)
    {   m_ps=new CPSProperties("Properties");
        pApp->m_prop_en=TRUE;
    }
    else
    {   m_ps->DestroyWindow();
        pApp->m_prop_en=FALSE;
    }
}

void CMainFrame::OnUpdateFileProperties(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    pCmdUI->SetCheck(pApp->m_prop_en);
}
```

14. U MainFrm.h dodamo public member varijablu pokazivač na klasu CPSProperties:
```
   public:
       CPSProperties *m_ps;
```
kao i direktivu kojom se uključuje zaglavlje PSProperties.h:
```
   #include "PSProperties.h"
```

Sada je gotovo i povezivanje tabovanog dijaloga sa ostatkom aplikacije. Moguće ga je aktivirati iz MainFrame menija (koji je aktivan kada nije otvoren ni jedan dokument). Da bi mogao da se aktivira i u onom drugom meniju, potrebno jeponoviti istu proceduru 13-14 i za taj meni, s tim što bi se hendleri nalazili u klasi CPrviView. Command hendler vrši kreiranje i uništavanje tabovanog dijaloga, dok Update hendler ažurira znak "štiklirano" pored naziva opcije u meniju, da bi se videlo da li je dijalog otvoren ili zatvoren.



Sl. 13 - Tabovani dijalog, prva kartica (tab).



Sl. 14 - Tabovani dijalog, druga kartica (tab).

5.2. Klipbord

Koncept klipborda postoji još od najranijih verzija Windows-a. Klipbord služi za prenošenje podataka između različitih aplikacija, između različitih dokumenata u okviru iste aplikacije, kao i između delova istog dokumenta. Standardne operacije sa klipbordom su: kopiranje (*copy*), odsecanje (*cut*), i ubacivanje (*paste*). To su toliko često korišćene operacije da su njihove komande dostupne na svim nivoima korisničkog interfejsa: preko menija (*Edit*), preko tulbara i preko akceleratora na tastaturi (CTRL+C, CTRL+X, CTRL+V).

Prilikom prenošenja podataka preko klipborda mora se voditi računa o tipu podataka - aplikacija u koju ubacujemo sadržaj može npr. da očekuje sliku, a mi da imamo na klipbordu npr. tekst. Zato Windows definiše određeni broj formata, kao karakteristike sadržaja klipborda, i načine na koji su podaci predstavljeni u određenom formatu. Za tekst su definisani formati: CF_TEXT (običan ANSI tekst), CF_OEMTEXT (tekst u specifičnoj kodnoj strani), CF_UNICODETEXT (unicode tekst); za rastersku grafiku: CF_BITMAP (bitmapa zavisna od uređaja), CF_DIB (bitmapa nezavisna od uređaja), CF_TIFF (TIFF format); vektorsku grafiku: CF_METAFILEPICT, CF_ENHMETAFILE (WMF, EMF metafajl); audio formati: CF_WAVE, CF_RIFF; spisak fajlova: CF_HDROP; itd.

Aplikacija pisana u MFC-u, i kreirana preko Application Wizard-a omogućava da joj se dodaju operacije sa klipbordom jednostavnim dodavanjem hendlera, pošto su te opcije već predviđene i u meniju, i na tulbaru, i u akceleratorskoj tabeli. Potrebno je, u Class Wizard-u, definisati command i update hendlere za copy, cut i paste operacije. Oni treba da se nalaze u klasi CPrviView, pošto operišu nad dokumentom, i nemaju smisla ako nije otvoren ni jedan dokument.

Klipbordu se pristupa sa nekoliko jednostavnih API funkcija, koje MFC takođe sadrži enkapsulirane u okviru klase CWnd. Princip korišćenja je sledeći: aplikacija koja smešta podatke na klipbord dužna je da očisti stari sadržaj klipborda, da alocira memoriju za podatke, kao i da ih upiše u odgovarajućem standardnom formatu. Aplikacija koja čita podatke sa klipborda može da pročita u kom su formatu, i da ih iskopira u svoj dokument.

Rad sa klipbordom podrazumeva korišćenje jednog zastarelog načina za alokaciju dinamičke memorije, koji potiče još od Windows-a 3.11. Radi kompatibilnosti i lakšeg portovanja softvera, te funkcije su deo i 32-bitnog Windows API-ja, ali se njihova upotreba ne preporučuje osim tamo gde se to eksplicitno zahteva. 16-bitna Windows aplikacija je mogla da alocira memoriju ili sa tzv. globalnog ili lokalnog heap-a. Prvi način je koristio funkcije GlobalAlloc, GlobalRealloc, GlobalLock, GlobalUnlock, i još neke, dok je drugi način pozivao iste funkcije sa prefiksom Local. Prvi korak u alokaciji memorije je pozivanje funkcije GlobalAlloc, kojoj se prosleđuje skup flegova koji određuje karakteristike memorijskog bloka i veličina memorijskog bloka u bajtovima, a koja vraća sistemski identifikator (hendl - *handle*) alociranog memorijskog bloka. Pod Windows-om 3.11 programer je morao da posveti veliku pažnju *memory management*-u - skup flegova je određivao kako će se ponašati taj memorijski blok - da li je fiksiran u memoriji (GMEM_FIXED), pokretan (GMEM_MOVEABLE), odloživ (GMEM_DISCARDABLE), itd. (sve te komplikacije su izbegnute u 32-bitnom API-ju). Mi ćemo koristiti skup flegova GHND, što znači da je blok pokretan i da se vrši inicijalizacija prostora sa nulama. Posle poziva GlobalAlloc mi imamo hendl memorijskog bloka, ali da bi ga koristili treba nam pointer na taj blok. Pointer na blok alocirane memorije dobijamo pozivom funkcije GlobalLock kojoj prosleđujemo hendl memorijskog bloka. Nakon završetka korišćenja tog memorijskog bloka pointer se oslobađa pozivom funkcije GlobalUnlock. Tom memorijskom bloku se i nadalje može pristupati, samo treba kod svakog novog pristupa ponovo uzeti pointer sa GlobalLock, i osloboditi ga sa GlobalUnlock. U periodu kada memorijski blok nije "lock"-ovan, sistem ima ovlašćenja da radi sa njim šta hoće (može da ga pomera, prebacuje na disk, ...) a pointer ima nedefinisanu vrednost. U Windows-u 3.11 se preporučivalo da se memorijski blok drži otključan sve vreme osim tada kada se koristi, da bi aplikacija olakšala rad operativnom sistemu. Memorijski blok se dealocira pozivom funkcije GlobalFree, posle koje i hendl ima nevažeću vrednost. (u 32-bitnim Windows-ima programer je oslobođen razmišljanja oko svega ovoga - prosto pozove alokaciju sa malloc, dealokaciju sa free, a operativni sistem sam brine gde je taj blok memorije).

Mi ćemo definisati samo hendlere za copy i paste. Command hendler za naredbu copy izgleda ovako:

```
void CPrviView::OnEditCopy()
{   HANDLE hCopy;
    LPSTR lpCopy;
    if (OpenClipboard())
    {   BeginWaitCursor();
        EmptyClipboard();
        if ((hCopy = (HANDLE) ::GlobalAlloc (GHND, 2*sizeof(char))
           ) != NULL)
        {    lpCopy = (LPSTR) ::GlobalLock((HGLOBAL) hCopy);
             lpCopy[0]=(char)(GetDocument()->sadrzaj);
             lpCopy[1]='\0';
             ::GlobalUnlock((HGLOBAL) hCopy);
        }
        else
        {   AfxMessageBox("Greska u alokaciji memorije");
        }
        SetClipboardData (CF_TEXT, hCopy);
        CloseClipboard();
        EndWaitCursor();
    }
}
```

Sa OpenClipboard() se otvara klipbord, dodeljuje prozoru koji je pozvao ovu funkciju, čime on dobija ekskluzivno pravo na njegovo korišćenje (druge aplikacije ne mogu da ga koriste sve dok ga ovaj prozor ne oslobodi). Par funkcija BeginWaitCursor() i EndWaitCursor() ne tiču se klipborda, već služe za postavljanje peščanog sata na mesto kurzora što je standardna indikacija korisniku da operacija

može da potraje (naravno, ovo može da se upotrebi bilo gde u kodu gde korisnik za trenutak gubi pravo da upravlja aplikacijom, tj. gde mora da se sačeka završetak neke druge operacije). Prvo se briše stari sadržaj klipborda sa EmptyCliboard(). Zatim se, na gore opisani način, preko funkcije GlobalAlloc, alocira memorijski blok koji će čuvati podatke. U našem slučaju imamo string od samo jednog znaka, koji predstavlja sadržaj dokumenta. Pošto string mora da se završava znakom '\0' moramo da alociramo prostor i za ovaj znak. Pristup memoriskom bloku je uokviren parom funkcija GlobalLock i GlobalUnlock. Kada su podaci upisani u memorijski blok, hendl tog bloka se predaje klipbordu funkcijom SetClipboradData. Ovom funkcijom se i postavlja format podataka. Na kraju, poziva se funkcija CloseClipboard(), koja oslobađa klipbord, čime on postaje dostupan drugim aplikacijama.

Update hendler OnUpdateEditCopy koji se poziva kada se izabere Edit stavka menija treba da omogući da operacija copy bude uvek dostupna:

```
void CPrviView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{   pCmdUI->Enable(TRUE);
}
```

Command hendler za operaciju paste će biti pozvan samo u slučaju da se na klipbordu nalazi odgovarajući tip podataka, o čemu brine update hendler. Update hendler će onemogućiti pozivanje operacije paste ukoliko funkcija IsClipboardFormatAvailable vrati rezultat FALSE, što znači da se na klipbordu ne nalazi ono što nama treba, tj. konkretno tekst (CF_TEXT).

U command hendleru prvo otvaramo klipbord za ekskluzivni pristup pozivom OpenClipboard(). Zatim se sa GetClipboardData preuzme hendl memorijskog bloka koji čuva podatke. Da bi pristupili sadržaju memorijskog bloka, uzimamo pointer API funkcijom GlobalLock. Prepisujemo sadržaj memorijskog bloka u naš dokument, i "otključavamo" hendl memorijskog bloka. Nakon toga se klipbord oslobađa čime se dozvoljava drugim aplikacijama da mu pristupe. Što se tiče samog upisa u dokument, treba obezbediti prepisivanje starog sadržaja u undo bafer (što se radi uvek kada se menja sadržaj), postavljanje flega "dokument promenjen", i forsirano osvežavanje View-a.

```
void CPrviView::OnEditPaste()
{   HANDLE hCopy;
    LPSTR lpCopy;
    if (OpenClipboard())
    {   BeginWaitCursor();
        hCopy = (HANDLE) ::GetClipboardData(CF_TEXT);
        if (hCopy != NULL)
        {   lpCopy=(LPSTR) ::GlobalLock((HGLOBAL)hCopy);
            GetDocument()->NapuniUndoBafer();
            GetDocument()->sadrzaj=lpCopy[0];
            ::GlobalUnlock((HGLOBAL)hCopy);
            GetDocument()->SetModifiedFlag(TRUE);
            GetDocument()->UpdateAllViews(NULL);
        }
        CloseClipboard();
        EndWaitCursor();
    }
}

void CPrviView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{   pCmdUI->Enable(::IsClipboardFormatAvailable(CF_TEXT));
}
```

Ovim se dobija potpuno funkcionalna aplikacija koja je spremna za prevođenje i izvršavanje.

Operacija cut bi se realizovala slično operaciji copy, s tim dodatkom što bi trebalo obezbediti brisanje onog dela dokumenta koji je prebačen na klipbord. Pošto takva operacija nema smisla za naš jednostavan dokument od jednog karaktera, nismo je ni realizovali.

Domaći zadaci.

1. Isprobati funkcionalnost aplikacije. Probati prenos preko klipborda između dva dokumenta, kao i između dve aplikacije.

6. GDI

GDI - *Graphics Device Interface*, je sistemska biblioteka za grafičku podršku Windows okruženju. Zadužena je za crtanje prozora, dijaloga, ikona, odnosno svih elemenata korisničkog interfejsa, kao i grafike koju definiše korisnik. Praktično sav grafički izlaz, tj. ono što je korisniku vidljivo, crta se uz pomoć ove biblioteke.

GDI omogućava tri vrste grafičkog izlaza: vektorsku i rastersku grafiku, i tekst. Vektorska grafika je podržana nizom funkcija za crtanje grafičkih primitiva - tačaka, linija, elipse, lukova, kao i tzv. metafajlovima - mogućnošću da se zapamti sekvenca GDI komandi koja generiše neki crtež. Ispis teksta je na prvi pogled neobično složen, ali to je zbog toga što se programeru omogućava da kontroliše sve njegove parametre - veličinu, font, stil, itd. Rasterska grafika je podržana preko tzv. bitmapa - pravougaonih oblasti koje pamte cele slike, tačku po tačku. I metafajlovi i bitmape su sistemski formati - mogu se prenositi preko klipborda, i čuvati u fajlovima na disku u standardnom formatu.

Od GDI biblioteke se zahteva da funkcioniše na raznolikom hardveru, i zbog toga je njena najvažnija karakteristika nezavisnost od uređaja (*device independence*). To se postiže tako što GDI funkcije ne pristupaju direktno grafičkom hardveru, već preko Windows drajvera za taj uređaj. GDI oslobađa programera brige o tome kakve su stvarne karakteristike fizičkog uređaja. To je veoma važno da bi Windows aplikacija mogla da radi na bilo kakvom sistemu, bez obzira na instaliranu grafičku karticu i grafički režim u kojem ona radi. Takođe zahvaljujući GDI-ju Windows aplikaciju ne zanima koji je štampač instaliran - drajver za štampač pretvara GDI pozive u komande konkretnom uređaju. Još jedna važna karakteristika je mogućnost softverske emulacije operacija koje ne mogu da se realizuju na datom hardveru. Na primer, prilikom štampanja slike u boji na crno-belom štampaču boje će se konvertovati u nijanse sive. Takođe, tzv. *true color* slika (sa 2^{24} boja) može da se prikaže i na grafičkoj kartici koja radi u režimu 256 boja, preslikavanjem stvarnih boja u najsličnije boje koje podržava uređaj.

GDI podržava dve vrste fizičkih uređaja, ekrane i štampače, kao i dve vrste "virtuelnih" uređaja, bitmape i metafajlove. Svaki GDI uređaj poseduje jednu strukturu podataka, tzv. kontekst uređaja (*device context*), koja sadrži osnovne parametre tog uređaja (kao što su broj boja, horizontalna i vertikalna rezolucija, visina i širina, i druge). Na slici 3, gde je prikazana hijerarhija MFC klasa, možemo da uočimo klasu konteksta uređaja CDC, kao i njene izvedene klase CClientDC, CMetaFileDC, CPaintDC, CWindowDC. Klasa CDC predviđena je da bude kontekst štampača i bitmape, CMetaFileDC kontekst metafajla, a CClientDC, CPaintDC i CWindowDC konteksti ekrana (od ovih je za nas najznačajniji CPaintDC). MFC definiše i GDI objekte za crtanje (klase koje su izvedene iz CGDIObject: CBitmap, CBrush, CFont, CPalette, CPen, CRgn).

6.1. Vektorska grafika i tekst

Da bi videli kako se pomoću GDI-ja crta vektorska grafika i tekst, uradićemo jedan ilustrativan primer. Ceo crtež će se nalaziti u nemodalnom dijalogu CDlgNemodalni. Hoćemo da nacrtamo grafik funkcije $y = e^{-0.2x} \sin 5x$, sa envelopom i propratnim tekstom (u različitim fontovima). Takođe, snimićemo deo crteža u metafajl, i rekonstruisati ga pozivom funkcije za crtanje metafajla. Prvo ćemo uraditi pripremne radnje:

1. U DlgNemodalni.cpp dodati direktivu `#include<math.h>`, da bi mogli da koristimo matematičke funkcije exp i sin.
2. U resursnom editoru povećati dimenzije dijaloga IDD_DLGNEMOD, bar na dimenzije 280x200, i pomeriti tastere u gornji desni ugao.
3. Pošto ćemo koristiti metafajl, treba da deklarišemo njegov hendl u DlgNemodalni.h:

```
class CDlgNemodalni : public CDialog
{   ...
    public:
        HMETAFILE hmf;
    ...
```

```
};
```

4. U hendleru OnInitDialog, koji se poziva jedamput prilikom otvaranja dijaloga kao odgovor na poruku WM_INITDIALOG, nacrtaćemo grafik sa envelopom i smestiti ga u metafajl:

```
BOOL CDlgNemodalni::OnInitDialog()
{   CDialog::OnInitDialog();
    // TODO: Add extra initialization here
    CMetaFileDC dc;
    dc.Create();
    int ksx=50,ksy=200; //centar koordinatnog sistema
    int yr=100,xr=50; //rezolucija - broj piksela po jedinici
    double ymax=1,xmax=5; //maks. vrednosti na grafiku
    int ksymax=(int)(ymax*yr),
    ksxmax=(int)(xmax*xr); //visina i sirina koordinatnog sistema
    double f[250],g[250];
    //
    // Generisanje funkcije i envelope
    //
    int i;
    double x,dx;
    for(i=0,x=0,dx=xmax/(ksxmax-1);i<ksxmax;i++,x+=dx)
    {   g[i]=exp(-0.2*x);
        f[i]=sin(5*x)*g[i];
    }
    //
    // Kreiranje "olovaka" - pen
    CPen penOld; //default pen
    CPen penKS; //pen za koordinatni sistem
    CPen penG; // pen za grafik
    CPen penE; //pen za envelopu
    penKS.CreatePen(PS_DASH,1,RGB(255,255,0));
    penG.CreatePen(PS_SOLID,2,RGB(0,0,255));
    penE.CreatePen(PS_SOLID,1,RGB(255,0,0));
    // Pamtimo default pen da bi mogli kasnije da ga vratimo
    CPen* ppenOld=dc.SelectObject(&penOld);
    //
    // Crtanje koordinatnog sistema
    //
    dc.SelectObject(&penKS);
    dc.MoveTo(ksx,ksy-ksymax);
    dc.LineTo(ksx,ksy+ksymax);
    dc.MoveTo(ksx,ksy);
    dc.LineTo(ksx+ksxmax,ksy);
    //
    // Crtanje funkcije
    //
    dc.SelectObject(&penG);
    dc.MoveTo(ksx,ksy-(int)(f[0]*yr));
    for(i=1,x=dx;i<ksxmax;i++,x+=dx)
        dc.LineTo(ksx+i,ksy-(int)(f[i]*yr));
    //
    // Crtanje envelope
    //
    dc.SelectObject(&penE);
    dc.MoveTo(ksx,ksy-(int)(g[0]*yr));
    for(i=1,x=dx;i<ksxmax;i++,x+=dx)
        dc.LineTo(ksx+i,ksy-(int)(g[i]*yr));
    dc.MoveTo(ksx,ksy+(int)(g[0]*yr));
    for(i=1,x=dx;i<ksxmax;i++,x+=dx)
        dc.LineTo(ksx+i,ksy+(int)(g[i]*yr));
```

```
    // Vracamo stari, default pen
    dc.SelectObject(ppenOld);
    hmf=dc.Close();
    return TRUE; //return TRUE unless you set the focus to a control
                 //EXCEPTION: OCX Property Pages should return FALSE
}
```

Prvo kreiramo kontekst metafajla kao C++ objekat, a zatim pozivamo njegovu funkciju Create() kojom se on inicijalizuje. Nadalje, sve što crtamo po ovom kontekstu, neće se prikazivati već će se čuvati u metafajlu. Kada završimo sa crtanjem, pozivamo funkciju Close(), koja zatvara kontekst metafajla i vraća hendl. Taj hendl ćemo koristiti kasnije, u drugim funkcijama, da bi pristupili metafajlu.

U prvom delu funkcije kreiramo nizove f[] i g[] koji će sadržavati vrednosti koje iscrtavamo. Taj deo je klasičan C kod. Da bi nešto nacrtali, moramo da koristimo GDI objekat "olovku" (*pen*). Default olovka crta crnom bojom, punu liniju. Da bi pokazali kako mogu da se dobiju različiti stilovi crtanja, za svaki ćemo kreirati posebnu olovku - žutu, isprekidanu za koordinatni sistem (penKS); crvenu, punu, za envelopu funkcije (penE); i plavu, punu, debelu (širine 2 piksela) za grafik funkcije (penG). Takođe ćemo sačuvati vrednost default olovke (penOld) da bi mogli da restauiramo tu vrednost na kraju crtanja. Olovka se kreira pozivom CPen funkcije CreatePen koja ima kao argumente stil, širinu i boju olovke. Tekuća olovka u kontekstu uređaja se postavlja pozivom CDC funkcije SelectObject, koja kao argument ima pokazivač na novu olovku, a vraća pokazivač na staru olovku. CDC funkcijom MoveTo postavlja se tekuća pozicija za crtanje. CDC funkcijom LineTo crta se prava linija, datom olovkom, od tekuće pozicije do pozicije navedene u pozivu, pri čemu se ažurira vrednost tekuće pozicije. Napomena: koordinatni sistem je po defaultu takav da je centar u gornjem levom uglu, a jedinica mere piksel (to se može izmeniti).

Od GDI funkcija za vektorsku grafiku ovde smo koristili samo dve, MoveTo i LineTo, ali spomenimo i još neke: Arc, ArcTo crtaju luk - deo elipse; Polyline, PolyPolyline, PolylineTo crtaju izlomljenu (poligonalnu) liniju; PolyBezier, PolyBezierTo crtaju jednu ili više Bezijeovih krivih. Funkcije koje imaju sufiks To pri crtanju koriste tekuću poziciju i pri tome je ažuriraju, i zbog toga su zgodne za nadovezivanje elemenata jedne na druge. Ostale funkcije crtaju nezavisno od tekuće pozicije, tj. koriste samo ulazne argumente funkcije.

5. U hendleru OnPaint, koji je odgovor na poruku WM_PAINT, treba da se nalazi kompletan kod za crtanje. Prvo ćemo u Class Wizard-u dodati taj hendler u mapu poruka, a zatim ćemo ga i napisati. U njemu ćemo pozvati iscrtavanje zapamćenog metafajla, kao i ispisivanje dodatnog teksta:

```
void CDlgNemodalni::OnPaint()
{   CPaintDC dc(this); // device context for painting
    // TODO: Add your message handler code here
    int ksx=50,ksy=200; //centar koordinatnog sistema
    int yr=100,xr=50; //rezolucija - broj piksela po jedinici
    double ymax=1,xmax=5; //maks. vrednosti na grafiku
    int ksymax=(int)(ymax*yr),
    ksxmax=(int)(xmax*xr); //visina i sirina koordinatnog sistema
    int oldBkMode;
    // Promena rezima pozadine u prozirno
    oldBkMode=dc.SetBkMode(TRANSPARENT); //necemo OPAQUE
    //
    // Iscrtavanje snimljenog metafajla
    //
    dc.PlayMetaFile(hmf);
    //
    // Ispis teksta
    //
    CString s1="Grafik funkcije";
    CString s2="y=exp(-0.2*x)*sin(5*x)";
    CFont fontNew1;
    CFont *pfontOld;
    // Biranje predefinisanog fonta
    fontNew1.CreateStockObject(ANSI_FIXED_FONT);
```

```
    //
    // Tekst iznad grafika
    //
    // Biramo sistemski font jednake sirine, i pamtimo default font
    pfontOld=dc.SelectObject(&fontNew1);
    // Velicina polja za tekst se proracunava tek nakon
    // definisanja fonta
    CSize ss1=dc.GetTextExtent(s1);
    dc.SetTextColor(RGB(0,255,255)); // Boja teksta - cijan
    dc.TextOut(ksx+ksxmax/2-ss1.cx/2,ksy-ksymax-ss1.cy,s1);//Ispis
    //
    // Tekst ispod grafika
    //
    dc.SetTextColor(RGB(255,0,255)); // Boja - ljubicasta
    dc.SelectObject(pfontOld); // Vracamo default font
    ss1=dc.GetTextExtent(s1); // Proracun dimenzija polja za tekst
    dc.TextOut(ksx+ksxmax/2-ss1.cx/2,ksy+ksymax+ss1.cy,s1); // Ispis
    // Kreiranje novog fonta
    CFont *pfontNew2;
    LOGFONT lf; // Struktura logickog fonta koju treba popuniti
    memset(&lf,0,sizeof(LOGFONT)); // Inicijalizacija
    lstrcpy(lf.lfFaceName,"Times New Roman"); // Ime fonta
    // Proracun visine fonta
    lf.lfHeight=(-1)*MulDiv(10,dc.GetDeviceCaps(LOGPIXELSY),72);
    lf.lfItalic=TRUE; // Osobina fonta - "italic"
    pfontNew2=new CFont(); // Kreiranje objekta fonta
    // Inicijalizacija LOGFONT strukturom
    pfontNew2->CreateFontIndirect(&lf);
    dc.SelectObject(pfontNew2); // Izabiranje kreiranog fonta
    // Proracun dimenzija polja za tekst
    CSize ss2=dc.GetTextExtent(s2);
    // Ispis
    dc.TextOut(ksx+ksxmax/2-ss2.cx/2,ksy+ksymax+ss1.cy+ss2.cy,s2);
    dc.SelectObject(pfontOld); // Vracanje starog, default fonta
    delete pfontNew2; // Brisanje objekta fonta
    dc.SetBkMode(oldBkMode); // Vracanje na stari rezim pozadine
    // Do not call CDialog::OnPaint() for painting messages
}
```

U hendleru OnPaint() crtamo koristeći kontekst uređaja CPaintDC. Prvo kreiramo taj kontekst, i vežemo ga sa tekućim dijalogom (odnosno prozorom tog dijaloga) prosleđujući pokazivač `this`. Zatim vršimo inicijalizaciju raznih parametara grafika, na isti način kao kad smo crtali po metafajlu. Ovi parametri će se koristiti za pozicioniranje teksta oko grafika. Sledeće, menjamo režim pozadine u providan (*transparent*), iz default režima neprovidan (*opaque*). U CPaintDC kontekstu reprodukujemo zapamćeni metafajl, čime dobijemo rekonstruisani crtež. Sada treba još da dodamo tekst.

Po defaultu, tekst se ispisuje u fontu koji je definisan u property-jima za dijalog. Visual C++ postavlja da taj predefinisani font bude MS Sans Serif veličine 8 tačaka, bold. Ovo je tzv. font promenljive širine jer karakteri imaju različite širine (npr. "i" je uže od "m"). Takođe postoji predefinisani fiksni font (jednake širine karaktera). Po default-u to je Courier. Tekst iznad grafika, "Grafik funkcije", hoćemo da prikažemo u svetloplavoj boji, default veličini, i u predefinisanom fiksnom fontu. Kreiramo novi GDI objekat fonta fontNew1, i postavljamo na vrednost ANSI_FIXED_FONT (sa CreateStockObject). Moguće vrednosti predefinisanih fontova su:

- ANSI_FIXED_FONT - fiksni font.
- ANSI_VAR_FONT - font promenljive širine.
- DEVICE_DEFAULT_FONT - predefisani font uređaja.
- OEM_FIXED_FONT - fiksni font u nekoj kodnoj strani.
- SYSTEM_FONT - predefinisani sistemski font.
- SYSTEM_FIXED_FONT - predefinisani fiksni sistemski font.

Kao i u slučaju olovke, CDC funkcijom SelectObject biramo taj novi font, a funkcija nam vraća pokazivač na stari font, koji čuvamo u pfontOld da bi mogli da ga rekonstruišemo kasnije.

Boju teksta u kontekstu uređaja postavljamo funkcijom SetTextColor. Njoj se prosleđuje vrednost boje "zapakovana" u DWORD na način 0x00BBGGRR, koju daje makro RGB. Funkcija GetTextExtent vraća veličinu pravougaonika koji zauzima tekst (*text box*), preko objekta CSize. Tu informaciju koristimo prilikom centriranja teksta - funkcija za ispis, TextOut, zahteva kao argument koordinate gornjeg levog ugla teksta. Njih proračunavamo na način prikazan u pozivu TextOut. *x* koordinata se postavlja tako da tekst bude centriran u odnosu na širinu grafika: $x_0+x_{max}/2-t_x/2$. y koordinata se postavlja za jedan red iznad gornje granice grafika: $y_0-y_{max}-t_y$ ((x_0, y_0) je koordinatni početak, (x_{max}, $2y_{max}$) su dimenzije grafika, a (t_x, t_y) dimenzije polja za tekst).

Napisaćemo isti ovaj tekst ispod grafika, u default fontu za ovaj dijalog, ali u drugoj boji. Sa SetTextColor promenimo boju teksta u kontekstu uređaja u ljubičastu, i vratimo zapamćeni default font sa SelectObject. Dimenzije polja za tekst moramo da proračunamo ponovo sa GetTextExtent, jer one nisu iste za dva različita fonta. Tekst ispisujemo sa TextOut, na *x* poziciji koja se isto računa kao i za tekst iznad grafika (centrirano), ali na *y* poziciji koja je za jedan red ispod donje granice grafika: $y_0+y_{max}+t_y$.

Videli smo kako se u principu koriste funkcije za ispis teksta, i kako se menja font, iz skupa predefinisanih fontova. Ostaje nam da vidimo kako se bira proizvoljan font. Recimo da hoćemo da ispišemo koja je to funkcija na grafiku, u iskošenom (italic) Times New Roman fontu, kako se i pišu jednačine. Biranje proizvoljnog fonta je složenije od uzimanja predefinisanog fonta. Moramo da popunimo strukturu LOGFONT, i da font inicijalizujemo tom strukturom pozivom funkcije CreateFontIndirect. Prvo inicijalizujemo LOGFONT strukturu sa nulama (poziv memset). Zatim popunjavamo ona polja koja su nama interesantna:

- Polje `lfFaceName` je string koji sadrži ime fonta, kod nas je to "Times New Roman".
- Polje `lfHeight` treba da sadrži visinu fonta, u pikselima. CDC funkcija GetDeviceCaps vraća parametre uređaja, a za prosleđeni argument LOGPIXELY vraća broj piksela po inču, po vertikali, za konkretan uređaj.
- Polje `lfItalic` određuje da li će font biti iskošen (italic).

Nakon inicijalizacije fonta strukturom LOGFONT, on je spreman za korišćenje, i bira se CDC funkcijom SelectObject (pokazivač na stari font već imamo od ranije). Dalje se tekst ispisuje na standardni, već opisani način, jedan red ispod postojećeg teksta, i centrirano. Posle iscrtavanja se vraća stari, default font. Pošto smo objekat fonta kreirali dinamički operatorom new, moramo i da ga obrišemo operatorom delete. Na kraju se restauriše i režim pozadine.

6. Promenićemo ponašanje dijaloga na pritisak tastera Akcija. Neka, umesto prikazivanja poruke, snima metafajl u fajl crtez.wmf na disku:

```
void CDlgNemodalni::OnAkcija()
{   // TODO: Add your control notification handler code here
    // AfxMessageBox("Neka akcija");
    ::CopyMetaFile(hmf,"crtez.wmf");
}
```

7. Na kraju, kada se zatvara dijalog moramo da obrišemo metafajl, a to se radi u PostNcDestroy hendleru:

```
void CDlgNemodalni::PostNcDestroy()
{   // TODO:Add your specialized code here and/or call the base class
    ::DeleteMetaFile(hmf);
    delete this;
    //CDialog::PostNcDestroy();
}
```

Nakon svih ovih izmena program može da se prevede i startuje. Treba da se u nemodalnom dijalogu dobije crtež kao na slici 15.

Metafajlovi sa kojima smo radili u ovom primeru postoje još od prvih verzija Windows-a. Sa pojavom novih, 32-bitnih verzija Windows-a, pojavila se potreba za redefinisanjem ovog formata, kao i dodavanjem novih mogućnosti. Radi kompatibilnosti i mogućnosti portovanje 16-bitnog softvera na 32-bitne platforme, WMF metafajlovi nisu menjani, a zato je uveden novi, usavršeni format - EMF (*enhanced metafile*). MFC i API funkcije koje rade sa EMF imaju slična imena kao i funkcije za WMF, uz dodatak Enh/Enhanced. Tako npr. hendl ima tip HENHMETAFILE, funkcije članice klase CDC su CreateEnhanced, CloseEnhanced, PlayMetaFile (sa argumentom više u odnosu na klasičan metafajl), dok su API funkcije DeleteEnhMetaFile, CopyEnhMetaFile.

Kao što je napomenuto na početku, GDI nudi mogućnost podešavanja jedinice mere. Koje jedinice mere se koriste, koja je orijentacija i položaj koordinatnih osa određuje tzv. režim preslikavanja koordinata (*mapping mode*). Po defaultu (a to je ono što smo mi koristili) GDI radi sa 1:1 (MM_TEXT) preslikavanjem, što znači da su logičke i fizičke koordinate iste, odnosno, da nam je jedinica mere piksel. U donjoj tabeli su dati svi režimi preslikavanja koje podržava GDI.

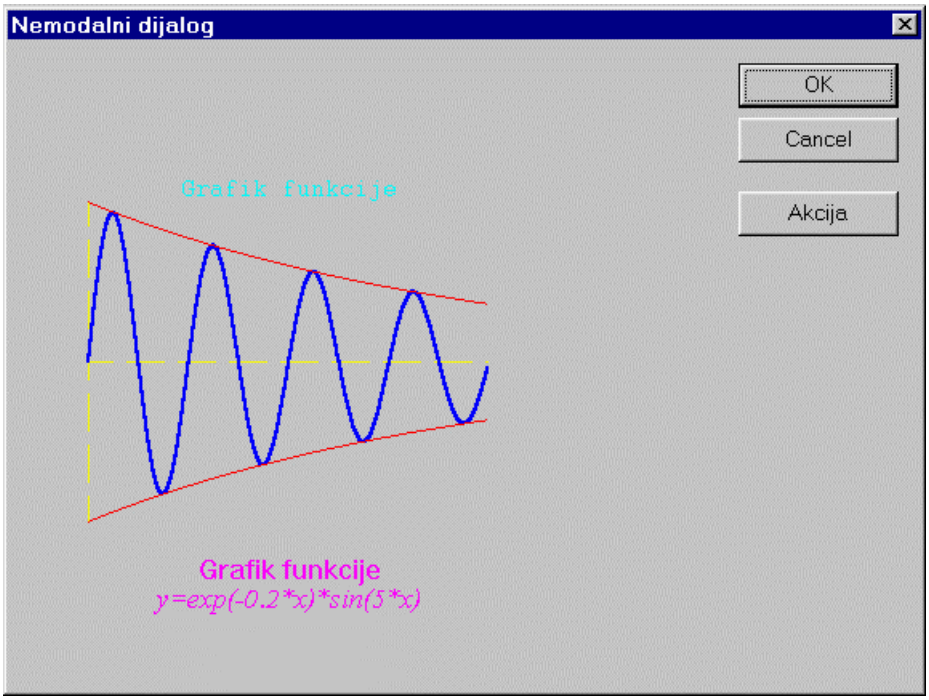
režim preslikavanja	fizička jedinica	smer x ose	smer y ose
MM_TEXT	piksel	na desno	na dole
MM_LOMETRIC	0.1 mm	na desno	na gore
MM_HIMETRIC	0.01 mm	na desno	na gore
MM_LOENGLISH	0.01 inča	na desno	na gore
MM_HIENGLISH	0.001 inča	na desno	na gore
MM_TWIPS	1/1440 inča	na desno	na gore
MM_ISOTROPIC	proizv. (isto po x/y)	proizvoljno	proizvoljno
MM_ANISOTROPIC	proizvoljno	proizvoljno	proizvoljno

Režim preslikavanja se postavlja CDC funkcijom SetMapMode. Funkcijom GetMapMode se može pročitati koji režim je trenutno aktivan.

Režimi preslikavanja MM_ISOTROPIC i MM_ANISOTROPIC zahtevaju dodatna podešavanja. Pre svega, između logičkih koordinata (onih koje mi koristimo u pozivima funkcija) i fizičkih koordinata (na uređaju koji prikazuje crtež) postoji linearna veza:

$$X = (x - x_0)\frac{E_X^V}{E_X^W} + X_0, \quad Y = (y - y_0)\frac{E_Y^V}{E_Y^W} + Y_0$$

(X, Y) su fizičke koordinate, (x, y) logičke koordinate; (X_0, Y_0) je fizički koordinatni početak (*viewport origin*), (x_0, y_0) je logički koordinatni početak (*window origin*); (E^V_X, E^V_Y) su fizički koeficijenti za skaliranje (*viewport extent*), (E^W_X, E^W_Y) su logički koeficijenti za skaliranje (*window extent*). Po defaultu parametri imaju vrednosti (X_0, Y_0)=(0,0), (x_0, y_0)=(0,0), (E^W_X, E^W_Y)=(1,1), (E^V_X, E^V_Y)=(1,1). Početak koordinatnog sistema (bilo logičkog ili fizičkog) može da se promeni CDC funkcijama SetWindowOrg i SetViewportOrg; koeficijenti skaliranja mogu da se promene pozivom CDC funkcija SetWindowExt i SetViewportExt.



Sl. 15 - Crtanje u nemodalnom dijalogu.

6.2. Rasterska grafika

Druga vrsta grafike koju podržava GDI je rasterska grafika. Kod nje je slika predstavljena matricom tačaka - piksela (*pixel* - *picture element*), a svaki piksel je predstavljen intenzitetom (za sivu sliku), bojom u nekom koordinatnim sistemu boja - RGB, HSI, YUV (za *true color* sliku), ili indeksom u tabeli preslikavanja boja (za *color mapped* slike). Rasterska grafika se najčešće koristi za prikaz digitalizovanih slika iz realnog sveta dobijenih nekim 2D senzorom (skenerom, kamerom, fotoaparatom, medicinskim uređajima za snimanje).

Windows podržava rad sa rasterskom grafikom preko koncepta bitmapa. Bitmapa je, slično metafajlu, sistemski medijum za memorisanje grafike. Bitmapa se može snimiti i učitati iz fajla, preneti preko klipborda, prikazati. Pošto je rad sa njima složeniji nego sa vektorskom grafikom, a primene specifičnije, nećemo pisati kod za njih. Čitalac se upućuje na literaturu [1] gde je to detaljno opisano. Preporučuje se pokretanje i analiza primera DIBLook koji se isporučuje uz Visual C++ 5. To je MFC doc/view aplikacija koja može da učita bitmapu iz fajla i da je prikaže u view-u. U okviru tog primera se može videti praktično sve što je potrebno za rad sa bitmapama.

Windows nudi podršku za dve vrste bitmapa: zavisne od uređaja (DDB - *device dependent bitmap*) i nezavisne od uređaja (DIB - *device independent bitmap*). Svaka bitmapa se sastoji u principu iz dva dela: zaglavlja (*header*), i podataka. Zaglavlje je struktura koja sadrži informacije o bitmapi kao što su visina, širina, broj bita po pikselu, itd. Sama slika, koja sledi iza zaglavlja, je najčešće složena liniju po liniju, a svaka linija piksel po piksel. Zbog toga bitmape često zauzimaju dosta memorije (odnosno prostora na disku). Windows API definiše C-ovske strukture za razne tipove bitmapa kojima se predstavljaju njihova zaglavlja.

DDB bitmapa kao zaglavlje sadrži strukturu BITMAP koja izgleda ovako:

```
typedef struct _tagBITMAP
{   LONG   bmType ;        // set to 0
    LONG   bmWidth ;       // width in pixels
    LONG   bmHeight ;      // height in pixels
    LONG   bmWidthBytes ;  // width of row in bytes
    WORD   bmPlanes ;      // number of color planes
    WORD   bmBitsPixel ;   // number of bits per pixel
    LPVOID bmBits ;        // pointer to pixel bits
} BITMAP, * PBITMAP ;
```

Iza zaglavlja se nalazi slika. Dužina svake linije slike mora da bude celobrojni umnožak 2 bajta (16 bita). Koordinatni početak je u gornjem levom uglu. Pikseli ove vrste bitmape se čuvaju onako kako zahteva uređaj za prikaz. To je najčešće u obliku kolor mapirane slike, gde svaki piksel predstavlja indeks boje u tabeli preslikavanja boja (kolor mapi). Kolor mapa ne postoji u okviru bitmape - već je definisana uređajem. Zbog toga se one i nazivaju zavisne od uređaja. Ove bitmape mogu da se brzo iscrtavaju korišćenjem bit-blok transfera (BitBlt). Ne postoji standardan fajl format za DDB kao za DIB, jer DDB nema smisla čuvati u fajlu bez kolor mape - na nekom drugom sistemu, sa drugačijim displejom, DDB se neće korektno prikazati.

Bitmape nezavisne od uređaja (DIB) sadrže, osim zaglavlja i piksela kao i DDB, i kolor mapu. Na taj način se preslikavanje između indeksa i stvarnih boja vrši nezavisno od uređaja na kom se prikazuje. Za DIB je predviđen format čuvanja u fajlu sa ekstenzijom BMP, DIB, ili RLE. DIB se sastoji iz sledećih celina:

1. zaglavlje fajla (samo kad se čuva u fajlu)
2. zaglavlje bitmape
3. kolor mapa (osim za *true color*)
4. pikseli bitmape

Zaglavlje fajla i bitmape su standardne C strukture, veličina kolor mape zavisi od bpp (*bytes per pixel*) za sliku, dok veličina niza piksela zavisi od dimenzija slike. Kada se bitmapa nalazi u memoriji, ona ne sadrži deo 1 - zaglavlje fajla. Taj deo se dodaje tek prilikom upisa u fajl.

Postoje dva DIB formata: (1) stari, OS/2 format, i (2) novi, Windows format. Stari OS/2 format kao zaglavlje fajla koristi strukturu BITMAPFILEHEADER, kao zaglavlje bitmape strukturu BITMAPCOREHEADER, a kolor mapa je niz struktura RGBTRIPLE. One izgledaju ovako:

```
typedef struct tagBITMAPFILEHEADER  // bmfh
{   WORD  bfType ;        // signature word "BM" or 0x4D42
    DWORD bfSize ;        // entire size of file
    WORD  bfReserved1 ;   // must be zero
    WORD  bfReserved2 ;   // must be zero
    DWORD bfOffsetBits ;  // offset in file of DIB pixel bits
} BITMAPFILEHEADER, * PBITMAPFILEHEADER ;

typedef struct tagBITMAPCOREHEADER  // bmch
{   DWORD bcSize ;      // size of the structure = 12
    WORD  bcWidth ;     // width of image in pixels
    WORD  bcHeight ;    // height of image in pixels
    WORD  bcPlanes ;    // = 1
    WORD  bcBitCount ;  // bits per pixel (1, 4, 8, or 24)
} BITMAPCOREHEADER, * PBITMAPCOREHEADER ;

typedef struct tagRGBTRIPLE  // rgbt
{   BYTE rgbtBlue ;   // blue level
    BYTE rgbtGreen ;  // green level
    BYTE rgbtRed ;    // red level
} RGBTRIPLE ;
```

Koordinatni početak se nalazi u donjem levom uglu - znači linije bitmape se slažu od dole na gore. Svaka linija bitmape mora da bude umnožak 4 bajta (32 bita). Ovo je uvedeno, isto kao i granularnost 16 bita za DDB, zbog lakšeg čitanja i pristupa.

Kod novog, Windows DIB formata, kao zaglavlje fajla se koristi struktura BITMAPFILEHEADER, kao zaglavlje bitmape BITMAPINFOHEADER struktura, a kolor mapa je niz RGBQUAD struktura. One izgledaju ovako:

```
typedef struct tagBITMAPINFOHEADER  // bmih
{   DWORD biSize ;           // size of the structure = 40
    LONG  biWidth ;          // width of the image in pixels
```

```
    LONG  biHeight ;         // height of the image in pixels
    WORD  biPlanes ;         // = 1
    WORD  biBitCount ;       // bits per pixel (1,4,8,16,24,or 32)
    DWORD biCompression ;    // compression code
    DWORD biSizeImage ;      // number of bytes in image
    LONG  biXPelsPerMeter ;  // horizontal resolution
    LONG  biYPelsPerMeter ;  // vertical resolution
    DWORD biClrUsed ;        // number of colors used
    DWORD biClrImportant ;   // number of important colors
} BITMAPINFOHEADER, * PBITMAPINFOHEADER ;

typedef struct tagRGBQUAD  // rgbq
{   BYTE rgbBlue;
    BYTE rgbGreen;
    BYTE rgbRed;
    BYTE rgbReserved;
} RGBQUAD;
```

Windows DIB nudi i jedan primitivan vid kompresije, tzv. RLE (*run-length encoded*). To nije optimalna kompresija, ali daje određena poboljšanja naročito ako slika sadrži velike oblasti iste boje. Run length kompresija, ukratko, radi na sledećem principu: nizovi susednih piksela iste boje zamenjuju se dužinom niza, i bojom (odnosno indeksom boje u kolor mapi) regiona. Npr. ako u jednoj liniji imamo niz piksela 15 15 15 27 27 27 27 27 on će se zameniti sa 3 15 5 27. Podrška za RLE se mora ručno dodati u kod, a poseban indikator u zaglavlju bitmape biCompression označava da je slika kompresovana (BI_RGB znači da nije kompresovana, BI_RLE4 i BI_RLE8 su RLE sa 4, odnosno 8 bita po pikselu).

Polje biBitCount označava koliko bita zauzima indeks boje u kolor mapi. Polje biClrUsed označava koliko indeksa ima kolor mapa. Npr. ako imamo bitmapu sa 256 boja, postavićemo biBitCount na 8 (2^8=256), biClrUsed na 256. Ukoliko se biClrUsed postavi na 0 (to je isto validna opcija) to znači da se koristi maksimalan broj boja koji dozvoljava biBitCount, a to je $2^{biBitCount}$. Polje biClrImportant označava koliko, od biClrUsed boja iz kolor mape, se koristi u prikazu. Ukoliko se ovo polje postavi na 0, to znači da se koristi svih biClrUsed boja.

Specijalan slučaj DIB je *true color* bitmapa koja nema kolor mapu, već pikseli predstavljaju konkretne R, G i B vrednosti komponenata boje piksela. Tada se biClrUsed i biClrImportant postavljaju na 0. Kod *true color* DIB je dozvoljeno i da se biSizeImage postavi na 0.

7. Procesi. Memorija. Dinamičke biblioteke

Ovde ćemo se ukratko upoznati sa načinom na koji Windows funkcioniše na niskom nivou. U prve dve celine su obrađeni procesi i način organizacije memorije. U trećoj celini obrađen je pojam dinamičke biblioteke (DLL). Pošto se dinamičke biblioteke često koriste pri pravljenju Windows aplikacija, pokazano je na primerima koji se vezuju sa aplikacijom Prvi kako se one koriste. Rad sa procesima i memorijom zahteva mnogo detaljniji pristup što izlazi iz okvira ovog praktikuma. Takođe, primene tih tehnika su vrlo specifične i nije verovatno da će zatrebati nakome ko prvi put izučava Windows programiranje. Čitalac se upućuje na literaturu [2] gde je sve ove teme, kao i mnoge druge sistemske teme koje ovde nisu ni spomenute, opisane do detalja.

7.1. Procesi

Kao što je napomenuto ranije, pri konstruisanju operativnog sistema Windows jedan od osnovnih zahteva je bio da on bude multiprogramski. To praktično znači da se na sistemskom nivou obezbedi multitasking - simultano izvršavanje više programa (*task*, *process*). Na sistemima koji poseduju više procesora, moguće je stvarno postići da više programa rade u paraleli. Na jednoprocesorskim sistemima, koristi se tzv. *time sharing*, odnosno deljenje procesora između više programa, na takav način da to bude transparentno za korisnika. Programi se nalaze u redu za čekanje i bivaju opsluženi od strane procesora kada dođu na red, jedno kratko vreme, nakon čega se ponovo vraćaju u red za čekanje. Svakom programu je dodeljen parametar - prioritet, koji označava koliko često će biti obrađivan u odnosu na druge programe koji se izvršavaju istovremeno. Upravljanje procesima je dosta složen zadatak za operativni sistem, i od načina na koji je to rešeno umnogome zavise i performanse celokupnog sistema.

Upravljanje procesima, kao i upravljanje memorijom, dosta se razlikuje u 16-bitnim (3.11) i 32-bitnim (NT, 9x) Windows-ima. Nadalje će se cela priča o upravljanju sistemskim resursima odnositi na 32-bitne Windows-e.

Osnovna programska jedinica za izvršavanje je proces. Svaki proces ima svoj adresni prostor, veličine 4 Gb (32 bita adrese) u kome se nalaze, preslikani, izvršni kod aplikacije, biblioteke, kao i memorijski prostor za podatke koje oni koriste. Operativni sistem vodi evidenciju o trenutno aktivnim procesima, čuvajući njihove parametre, upravljajući njihovim kreiranjem i uništavanjem, kao i upotrebom adresnog prostora i resursa sistema. Prilikom kreiranja novog procesa, operativni sistem mu dodeljuje adresni prostor, inicijalizuje parametre, i pokreće primarnu nit procesa. Kada se završi izvršavanje procesa, operativni sistem u potpunosti uništava njegov adresni prostor, oslobađa resurse koje je proces u toku izvršavanja zauzeo, i ažurira svoju evidenciju o stanju aktivnih procesa.

Kod 16-bitnih Windows-a organizacija procesa je bila bitno drugačija: svi procesi su delili isti adresni prostor, više instanci istog procesa je delilo isti memorijski prostor gde je smešten kod, a takođe su i biblioteke učitavane samo jedamput, i bile deljene od strane više programa. Ovakva štedljiva organizacija je bila nužna zbog hardverskih ograničenja. 16-bitni Windows programi su radili dobro dok je sve u redu; međutim, u slučaju pada aplikacije bila je ugrožena stabilnost sistema, jer su i aplikacija i OS delili isti prostor. Takođe je bilo izraženo tzv. "curenje resursa" (*resource leaks*) - polako gomilanje "đubreta" u memoriji koje potiče od procesa koji su završili sa radom a nisu propisno obavili čišćenje, koje posle dužeg rada dovodi do pada celog sistema.

32-bitni Windows-i su u tom pogledu znatno pouzdaniji i stabilniji – posle završetka rada procesa garantovano je čišćenje svega što je iza njega ostalo, čak i ako je programer to zaboravio. Kod Windows-a NT stabilnost je dalje unapređena "ograđivanjem" procesa jednih od drugih.

Kada se kreira proces, njemu se dodeljuje jedinstveni sistemski identifikator – hendl, koji se koristi u daljem radu kada se obraća tom procesu. Procesu se dodeljuje i niz drugih parametara, kao što su npr. promenljive okruženja (*environment*), tekući direktorijum, itd. Kada se pokrene aplikacija, poziva se njena WinMain/AfxWinMain funkcija što je za programera ulazna tačka programa. Aplikacija dobija od sistema, kao jedan od argumenata, i hendl tog procesa. Proces može da startuje drugi proces,

korišćenjem API funkcije CreateProcess (ili WinExec), kojoj prosleđuje parametre novog procesa preko strukutra SECURITY_ATTRIBUTES, STARTUPINFO i PROCESS_INFORMATION.

Proces može da se ugasi na više načina. Najpre, kada se završi njegova primarna nit. Na ovaj način se proces normalno završava, pod uslovom da su i ostale niti završile sa radom. Zatim, pozivom funkcije ExitProcess. Ovo je "kulturan" način gašenja procesa, pri čemu se regularno oslobađaju zauzeti resursi, za razliku od trećeg načina, preko funkcije TerminateProcess, koja proces gasi neopozivo, ne dajući mu vremena ni da počisti resurse iza sebe (ali to na sreću radi operativni sistem).

Osnovne klase prioriteta procesa su:

- IDLE_PRIORITY_CLASS - ova klasa prioriteta se koristi za procese koji rade u pozadini, npr. za screen saver-e.
- NORMAL_PRORITY_CLASS - pod ovu klasu proriteta potpada većina standardnih Windows aplikacija.
- HIGH_PRORITY_CLASS - ovu klasu prioriteta koristi npr. NT Task Manager (on mora da ima veći prioritet od "običnog" procesa, da bi mogao da ga prekine).
- REALTIME_PRIORITY_CLASS - ova klasa prioriteta se jako retko koristi, iz prostog razloga što tada proces ima veći prioritet od korisničkog interfejsa Windows-a, pa u slučaju zaglavljivanja aplikacije korisnik ne može više da komunicira sa sistemom.

Kod Windows-a 2000 su uvedene još dve klase prioriteta, ABOVE_NORMAL_PRIORITY_CLASS i BELOW_NORMAL_PRIORITY_CLASS.

Kao što postoji multutasking na sistemskoim nivou, postoji i na nivou procesa. Svaki proces može da sadrži jedan ili više podprocesa - tzv. programskih niti (*thread*). Niti se u okviru procesa izvršavaju konkurentno, dele isti adresni prostor procesa u kome "žive", jedino imaju nezavisne stekove. Kada se kreira nov proces, on po defaultu ima jednu, tzv. primarnu nit, koja počinje izvršavanje. Ta nit može da kreira druge niti. Slično kao kod procesa, nit se kreira pozivom API funkcije CreateThread, a gasi pozivima ExitThread i TerminateThread. Za razliku od gašenja procesa, gašenje niti može da ostavi "đubre" u memoriji koje tu ostaje sve dok se ne završi proces, pa na to treba obratiti naročitu pažnju. Nit može da se suspenduje API funkcijom SuspendThread, kao i da se ponovo aktivira API funkcijom ResumeThread. Podrška za *multithreading* programiranje je ugrađena i u MFC, kroz klasu CWinThread. Sinhronizacija između niti obavlja se preko sistemskih objekata događaja, semafora, kritičnih sekcija i muteksa. Događaji (*event*) omogućavaju da nit čeka na određeni događaj pre nego što se nastavi izvršavanje. Semafori (*semaphore*) se koriste za pristup deljenim resursima - semafor može da zabrani korišćenje resursa ukoliko ga koristi neka druga nit. Kritične sekcije (*critical section*) u niti služe da se onemoguće ostale niti da preuzmu procesor u trenutku kada ta nit radi neki posao koji mora da se završi. Muteks (*mutex*) se koristi kada mora da se obezbedi ekskluzivno pravo pristupa nekom resursu jednoj niti. Svi ovi objekti su raspoloživi za korišćenje ili dirtektno iz API-ja (funkcije CreateEvent, CreateSemaphore, EnterCriticalSection, LeaveCriticalSection, CreateMutex, ...) ili preko odgovarajućih MFC klasa (CEvent, CSemaphore, CCriticalSection, CMutex). U helpu za Visual C++, kao i u knjizi [2] možete naći veoma iscrpna objašnjenja kako se ovi objekti upotrebljavaju kao i primere *multithreading* aplikacija.

Nemaju sve niti unutar jednog procesa isti prioritet. Klasa prioriteta procesa se kombinuje sa tzv. relativnim prioritetom niti, da bi se dobio ukupni - bazni prioritet niti. Relativni prioriteti mogu da budu:

Relativni prioritet niti	*Modifikacija klase prioriteta procesa*
THREAD_PRIORITY_IDLE	minimalna vrednost
THREAD_PRIORITY_LOWEST	-2
THREAD_PRIORITY_BELOW_NORMAL	-1
THREAD_PRIORITY_NORMAL	0
THREAD_PRIORITY_ABOVE_NORMAL	+1
THREAD_PRIORITY_HIGHEST	+2
THREAD_PRIORITY_TIME_CRITICAL	maksimalna vrednost

Tako, npr. za nit sa normalnim relativnim prioritetom, bazni prioriteti niti u zavisnosti od klase prioriteta procesa mogu da budu:

Klasa prioriteta procesa	*Bazni prioritet niti*		
	za IDLE	*za NORMAL*	*za TIME CRITICAL*
IDLE_PRIORITY_CLASS	1	4	15
NORMAL_PRIORITY_CLASS (background)	1	7	15
NORMAL_PRIORITY_CLASS (foreground)	1	8	15
HIGH_PRIORITY_CLASS	1	13	15
REALTIME_PRIORITY_CLASS	16	24	31

Dato je i poređenje sa IDLE i TIME CRITICAL relativnim prioritetima jer se tu može videti sledeća stvar: sve klase prioriteta osim REALTIME se nalaze u opsegu 1-15, dok je opseg 16-31 rezervisan samo za REALTIME. Zato je on toliko opasan - čak ni NT Task Manager, koji radi u klasi HIGH, ne može da "ubije" zaglavljenu REALTIME aplikaciju jer uvek ima niži prioritet od nje. Pod Windows-om NT se dozvoljava pokretanje procesa u REALTIME klasi jedino korisnicima sa administratorskim privilegijama.

7.2. Memorija

Upravljanje memorijom kod 32-bitnih Windows-a je deo OS koji je pretrpeo najviše izmena i unapređenja pri prelasku sa 16-bitnih na 32-bitne OS. 16-bitni Windows je imao tzv. segmentnu organizaciju memorije, koja je direktna posledica organizacije Intelovih mikroprocesora. U 16-bitnom zaštićenom režimu procesora (*protected mode*) u kome je radio Windows 3.11, adresni prostor je bio izdeljen na segmente veličine 64 Kb. Adresa (odnosno pointer) imala je oblik (*segment:offset)*, gde je *segment* označavao početnu adresu segmenta, a *offset* pomeraj u odnosu na početak segmenta. I *segment* i *offset* su 16-bitne veličine. Pokazivači su bili definisani ili kao NEAR, tzv. "bliski" 16-bitni pokazivači koji sadrže samo *offset* deo adrese gde se *segment* podrazumeva, ili kao FAR - "daleki" pokazivači koji sadrže oba dela adrese. Alokacija memorije je bila posebna priča: da li se memorija alocira sa lokalnog ili globalnog *heap-a*; koji su atributi memorijskog bloka (moveable, discardable, ...); da li je pointer na memorijski blok "zaključan" pre korišćenja; itd. O svemu ovome je programer morao da brine. Kada smo pričali o upotrebi klipborda, pokazano je kako se te funkcije koriste. Dalje, svi procesi su delili isti adresni prostor, zajedno sa operativnim sistemom. Zato je Windows 3.11 bio izuzetno nestabilan - program koji krahira mogao je da poremeti kako druge programe, tako i sam OS. Druga stvar koja je dalje komplikovala upotrebu je česta upotreba deljenih resursa: DLL koji koristi više aplikacija se učitavao samo jedamput; više instanci iste aplikacije je imalo samo jednu kopiju u memoriji (otuda argument hPrevInst - *handle of previous instance*, u pozivu WinMain), čak su i neki sistemski objekti bili deljeni. Kako je već spomenuto ranije, nepravilna dealokacija je ostavljala "đubre" u memoriji (prostor koji je zauzet a više ne može da se koristi), koja je posle dužeg rada dovodila do pada sistema. Virtuelna memorija je bila podržana preko tzv. *swap file-a*, gde su se odlagale stranice koje trenutno nisu potrebne.

32-bitna organizacija memorije je umnogome uprostila upotrebu memorije. Uveden je ravan (*flat*) memorijski model, koji koristi samo jedan, 32-bitni tip pokazivača. Svaki proces ima svoj privatni virtuelni adresni prostor je veličine 4 Gb, koji je organizovan kao na slici 16. Postoje određene razlike u organizaciji memorije između Windows-a 95 i NT, zbog zahteva za kompatiblinošću 95-ice sa 3.11-icom.

Gornja 2 Gb i kod 95-ice i kod NT-a pripadaju operativnom sistemu. Tu se nalaze programi koji pokreću operativni sistem. Kod Windows-a 95, u gornjih 1Gb se nalaze VxD drajveri, programi za upravljanje memorijom i fajl sistemom, itd. 1 Gb ispod tog prostora predstavlja deljenu memoriju od strane svih procesa. U deljenoj oblasti memorije drže se sistemski deljeni DLL-ovi kao što su KERNEL32.dll, USER32.dll, GDI32.dll i ADVAPI32.dll koje koriste sve Windows aplikacije, 16-bitne aplikacije, kao i memorijski preslikane datoteke (*memory mapped files*) koje se u 32-bitnim Windows-ima koriste za međuprocesnu komunikaciju. Kod Windows-a 95 aplikaciji je dozvoljeno da piše po ovim oblastima, pa Windows 95 nije u potpunosti zaštićen – aplikacija može da izazove pad sistema. Kod Windows-a NT, gornja 2 Gb su potpuno nepristupačna za aplikaciju – na svaki pokušaj pristupa dobija se “Access Violation” nakon čega sistem gasi aplikaciju. Zato Windows NT učitava sistemske DLL-ove, kao i memorijski preslikane datoteke u donji deo adresnog prostora, zajedno sa izvršnim kodom i podacima procesa.

Oblast niža 2 Gb je oblast privatnog prostora procesa. Tu se nalaze preslikana EXE datoteka, svi potrebni DLL-ovi, stek, kao i blokovi memorije dinamički alocirani od strane aplikacije. Windows NT

rezerviše granične regione ove oblasti za oblasti nevažećih i nula pokazivača (po 64 Kb). Windows 95 takođe ima oblast nula-pokazivača od 4 Kb na početku, a zatim i prostor u kome se izvršavaju MS-DOS aplikacije. Ovo praktično znači da "koristan prostor" 32-bitne aplikacije kod 95-ice počinje od adrese 0x00400000, dok kod NT-a počinje od adrese 0x00010000.

<table>
<tr><th colspan="2">Windows 95</th><th colspan="2">Windows NT</th></tr>
<tr><td>0x FFFF FFFF
(1 Gb)
0x C000 0000</td><td>VxD,
memory manager,
file system</td><td rowspan="2">0x FFFF FFFF
(2 Gb)
0x 8000 0000</td><td rowspan="2">OS</td></tr>
<tr><td>0x BFFF FFFF
(1 Gb)
0x 8000 0000</td><td>mem-map files,
shared DLLs,
16-bit apps</td></tr>
<tr><td rowspan="2">0x 7FFF FFFF
(2 Gb - 4 Mb)
0x 0040 0000</td><td rowspan="2">Win32 process
private</td><td>0x 7FFF FFFF
(64 Kb)
0x 7FFF 0000</td><td>Bad pointer
assignment</td></tr>
<tr><td>0x 7FFE FFFF
(2 Gb - 128 Kb)
0x 0001 0000</td><td>Win32 process
private</td></tr>
<tr><td>0x 003F FFFF
(4 Mb - 4 Kb)
0x 0000 1000</td><td>MS-DOS,
16-bit Windows
process</td><td rowspan="2">0x 0000 FFFF
(64 Kb)
0x 0000 0000</td><td rowspan="2">NULL pointer
assignment</td></tr>
<tr><td>0x 0000 0FFF
(4 Kb)
0x 0000 0000</td><td>NULL pointer
assignment</td></tr>
</table>

Sl. 16 - Organizacija memorije kod Windows-a 95/NT

Za jedan konkretan proces, virtuelni adresni prostor na Windows-u 95 može da izgleda ovako (u prilogu D je dat kompletan sadržaj):

```
Adresa   Status     Veličina  Atributi        Fajl
BFFFF000 Free          4096
BFF70000 Private     585728 5  -rw- C:\WINDOWS\SYSTEM\KERNEL32.DLL
BFF61000 Free         61440
BFF50000 Private      69632 3  -Rw- C:\WINDOWS\SYSTEM\USER32.DLL
BFF46000 Free         40960
BFF20000 Private     155648 5  -Rw- C:\WINDOWS\SYSTEM\GDI32.DLL
BFF1F000 Free          4096
BFEB0000 Private     454656 3  -Rw-
BFEA0000 Private      65536 3  -Rw- C:\WINDOWS\SYSTEM\ADVAPI32.DLL
BFE96000 Free         40960
BFE90000 Private      24576 3  -Rw-
BFE86000 Free         40960
BFE80000 Private      24576 3  -Rw-
BFE20000 Free        393216
BFE10000 Private      65536 3  -Rw-
BFB73000 Free       2740224
BFB50000 Private     143360 7  -Rw-
835A0000 Free    1012596736
834A0000 Private    1048576 3  -rw-

                             . . .

80001000 Private       4096 1  -RW-
80000000 Private       4096 1  ----
7D670000 Free      43581440
7D660000 Private      65536 4  -rw- C:\WINDOWS\SYSTEM\INDICDLL.DLL
78040000 Free      90308608
78000000 Private     262144 3  -Rw- C:\WINDOWS\SYSTEM\MSVCRT.DLL
00870000 Free    2004418560
```

```
00770000 Private    1048576 2  -RW-
00660000 Private    1114112 4  -rw-
00540000 Private    1179648 6  -rw- Thread Stack
00530000 Private      65536 2  -RW-
00420000 Private    1114112 4  -rw-
00400000 Private     131072 4  -rw- C:\WINDOWS\TEMP\VMMAP.EXE
00000000 Free       4194304
```

U ovom pregledu je malim slovima označena oblast u kojoj postoje stranice i sa jednom i sa drugom osobinom. Velikim slovom je označena oblast gde sve stranice imaju tu osobinu. Crticom (-) su označeni blokovi koji nigde nemaju tu osobinu.

Adresni prostor je izdeljen na stranice (*page*) veličine 4 Kb. Virtuelna 32-bitna adresa se sastoji iz 3 dela: najviših 10 bita je indeks unutar direktorijuma stranica (*page directory*); srednjih 10 bita je indeks u tabeli stranica (*page table*), dok najnižih 12 bita predstavljaju adresu unutar stranice. Stranicama u memoriji su dodeljeni atributi, koji govore šta aplikacija može da radi sa tim memorijskim blokom: samo da čita, da čita i piše, da izvršava, itd. Kod Windows-a NT postoje sledeći atributi:

Atributi memorije	*Flegovi*
PAGE_NOACCESS (*)	`----`
PAGE_READONLY (*)	`-R--`
PAGE_READWRITE (*)	`-RW-`
PAGE_EXECUTE	`---E`
PAGE_EXECUTE_READ	`-R-E`
PAGE_EXECUTE_READWRITE	`-RWE`
PAGE_WRITECOPY	`CRW-`
PAGE_EXECUTE_WRITECOPY	`CRWE`

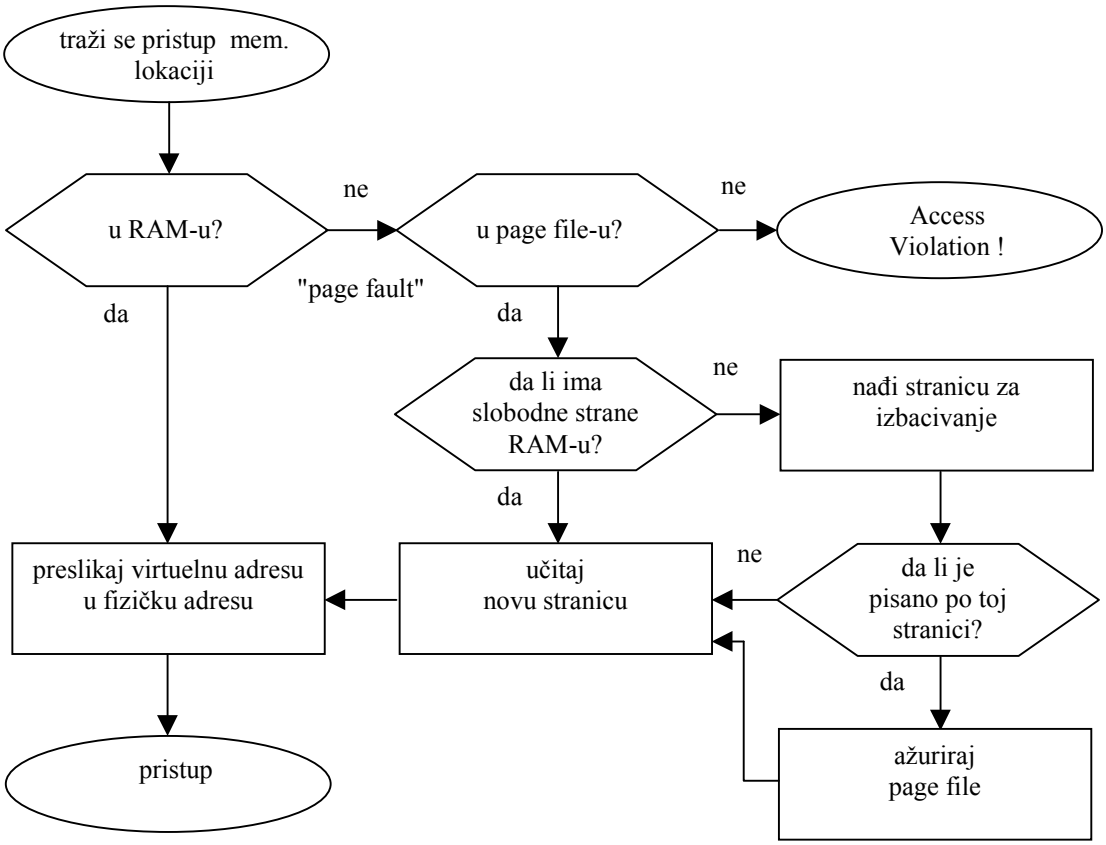
Windows 95 podržava samo prva tri atributa označena zvezdicom (*). Flegovi pored oznake atributa govore o tome šta sa tom stranicom može da se radi (*R = read*, *W = write*, *E = execute*). Ako se pokuša nedozvoljena vrsta pristupa, sistem će javiti "Access Violation". Fleg C (*copy on write*) omogućava da se u slučaju pisanja po stranici, da bi se sačuvao originalni sadržaj, napravi kopija izmenjene stranice u page fajlu. Windows 95 ne podržava E i C flegove, što se može videti i u gornjem prikazu sadržaja adresnog prostora. Pravo izvršavanja (E) se dobija pravom čitanja (R), a copy on write mehanizam je implementiran tako što se svi blokovi sa tim atributom već unapred iskopiraju u page fajl, ne čekajući potencijalni pristup sa upisanjem.

Mehanizam virtuelne memorije je kod 32-bitnih Windows-a implementiran preko *page* fajla i memorijski preslikanih datoteka. Page fajl sadrži stranice virtuelne memorije koje su alocirane, ali se ne koriste trenutno. Kada nam zatreba stranica koja nije u RAM-u, događa se tzv. stranični prekid (*page fault*), posle čega se učitava potrebna stranica, a izbacuje ona za koju OS pretpostavlja da će najmanje trebati ubuduće. Page fajl sadrži uglovnom delove adresnog prostora koje je alocirala aplikacija za čuvanje podataka. Delovi adresnog prostora gde se nalazi izvršni kod (EXE i DLL-ovi, kao i sistemski DLL-ovi) se ne nalaze u page fajlu, osim u specijalnim slučajevima, od kojih je jedan već pomenut - kada se aktivira *copy on write* mehanizam.

Memorijski preslikane datoteke su datoteke na hard disku koje operativni sistem koristi za čuvanje blokova memorije. Sadržaj datoteke je ustvari sadržaj memorije, počev od neke adrese. Kada OS učitava aplikaciju (EXE fajl) i njene DLL-ove sa hard diska u adresni prostor, on ih u stvari ne učitava, već te fajlove na disku (EXE, DLL) otvara kao memorijski preslikane datoteke i njihovom sadržaju dodeljuje deo adresnog prostora procesa. Ovakav mehanizam se upotrebljava iz dva razloga: tako se smanjuje vreme podizanja aplikacije, i ne opterećuje se page fajl. Međutim, u slučaju kada se aplikacija startuje sa flopi diska, ceo sadržaj se kopira u page fajl, jer je pristup flopiju isuviše spor.

Memorijski preslikane datoteke imaju još primenu i u komunikaciji između dva procesa, tako što oba procesa preslikaju deo svog adresnog prostora u isti fajl; takođe se koriste i za rad sa vrlo velikim datotekama u radu sa bazama podataka. Memorijski preslikane datoteke se ne ponašaju kao obične datoteke. Njih sistem tretira kao deo adresnog prostora, pa se s njima radi kao sa običnom memorijom, a VMM (*virtual memory manager*) čita i ažurira stranice memorije u fajlu.

Na slici 17 prikazan je mehanizam pristupa (bilo čitanja ili upisa) memorijskoj lokaciji preko VMM. Kada se zatraži pristup lokaciji, prvo se ispita da li se njena memorijska stranica već nalazi u RAM-u. Ukoliko se nalazi u RAM-u, virtuelna adresa se preslika u fizičku, i obavi se pristup. Ukoliko se ne nalazi u RAM-u, generiše se stranični prekid (*page fault*). Sledeće mesto gde se traži stranica je page fajl. Ukoliko se ne nalazi ni tamo, to znači da je pristup bloku nemoguć, da se javlja "Access Violation". U slučaju da smo našli potrebnu stranicu u page fajlu, treba je učitati u RAM. Ukoliko u RAM-u ima dovoljno mesta, sve je u redu - učita se stranica, preslika virtuelna adresa u fizičku i obavi pristup. Ukoliko više nema mesta u RAM-u, treba izbaciti neku stranicu iz RAM-a da bi se nova učitala. Kada se pronađe ona koju treba izbaciti, ukoliko je pisano po njoj, ažurira se njena "slika" u page fajlu. Kada se sve to obavi, učita se na njeno mesto nova stranica, preslikaju adrese i pristupi lokaciji.



Sl. 17 - Mehanizam preslikavanja virtuelne u fizičku adresu

7.3. Dinamičke biblioteke

Pod pojmom biblioteke podrazumeva se prevedena kolekcija funkcija, koja sama za sebe nije izvršni fajl (EXE), ali se u fazi linkovanja povezuje sa izvršnim kodom aplikacije koja koristi te funkcije. Većina OS i kompajlera podržava tzv. statičke biblioteke. Statičke biblioteke obično imaju ekstenziju LIB. Većina standrdnih C-ovskih biblioteka koje se isporučuju uz svaki kompajler se dobija u vidu statičkih biblioteka. Programer je takođe u mogućnosti da sam kreira svoju statičku biblioteku zadavanjem odgovarajućih opcija C kompajleru. Statičke biblioteke se povezuju sa aplikacijom u fazi linkovanja, u tzv. vreme prevođenja aplikacije (*compile-time*). Ovde *compile* ima značenje "kompletnog generisanja EXE" fajla koje uključuje: fazu prevođenja gde se dobijaju objektni OBJ fajlovi i fazu linkovanja gde se oni povezuju u EXE fajl. Svaka aplikacija koja koristi neku statičku biblioteku ima u okviru svog izvršnog fajla celu kopiju biblioteke.

Operativni sistem Windows, još od najranijih verzija, je ravnopravno uveo dinamičke biblioteke. Dinamičke biblioteke, koje imaju ekstenziju DLL (*dynamic link library*), u vreme izvršavanja (*run-time*) se ne nalaze u okviru izvršnog fajla aplikacije, već egzistiraju kao posebni DLL fajlovi. Oni se vezuju sa aplikacijom dinamički, u *run-time*-u, prilikom startovanja aplikacije. Ovakav pristup donosi uštede jer sve aplikacije koje koriste neki DLL koriste istu kopiju na disku, a ne sadrže je u svom izvršnom fajlu. Kod ranijih verzija Windows-a nisu bile toliko bitne uštede u prostoru na disku, ali je

ovo napravljeno upravo da obezbedi uštede u memoriji, jer se pod 16-bitnim Windows-ima DLL učitava samo jednom, za sve procese koji ga koriste. DLL-ovi su toliko bitan deo operativnog sistema Windows, da se i neki njegovi delovi (KERNEL, USER, GDI) koriste kao DLL-ovi.

Kod 32-bitnih Windows-a promenjena je organizacija procesa i memorije, ali je koncept DLL-a ostao. Kao što je moglo da se vidi u prethodnom delu, svaki proces učitava svoje kopije DLL-ova u svoj privatni adresni prostor, sa jednim malim izuzetkom kod Windows-a 95, koji ima područje i za deljene, sistemske DLL-ove.

DLL se sastoji iz zaglavlja i tela funkcija (koda). Zaglavlje se sastoji iz import i eksport sekcije. Import sekcija sadrži listu importovanih simbola, odnosno reference na funkcije koje ne postoje u tom DLL-u a koriste se u njemu, zajedno sa imenom DLL-a gde mogu da se nađu. Eksport sekcija sadrži listu eksportovanih simbola, a to su upravo funkcije koje se nalaze u tom DLL-u, koje on stavlja na raspolaganje drugim aplikacijama. Pomoću uslužnog programa DUMPBIN koji se isporučuje uz Visual C++ mogu se pročitati import i eksport sekcije DLL-ova i EXE fajlova. U prilogu E prikazana je kompletna import i ekport sekcija DLL-a koji ćemo kreirati kasnije, MojDLL.dll, i import sekcija naše aplikacije Prvi, gde se može videti da npr.:

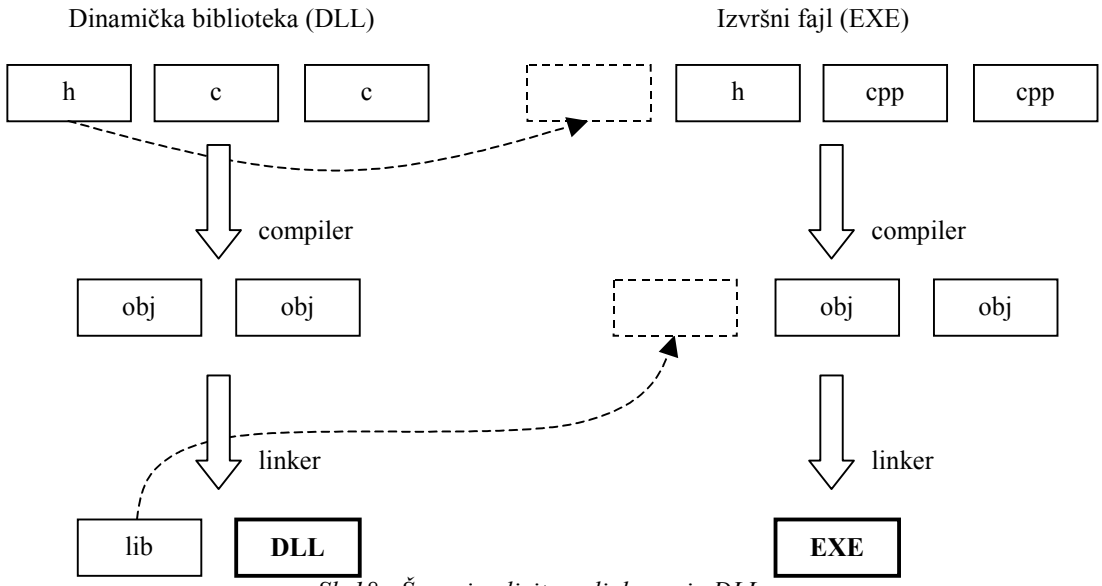
- Dinamička biblioteka MojDLL zahteva da bude učitan zajedno s njom i KERNEL32.dll (to je uvek slučaj pošto je to sistemski DLL). Dat je spisak funkcija iz KERNEL32.dll koje se koriste u MojDLL.dll.
- Dinamička biblioteka MojDLL eksportuje funkciju "Aritmetika", koju mogu da koriste druge aplikacije.
- Aplikacija Prvi zahteva sledeće DLL-ove: MojDLL.dll, MFC42D.DLL (DLL koji sadrži MFC, D=debug verzija), MSVCRTD.dll (C-run-time biblioteka, D=debug verzija), KERNEL32.dll, USER32.dll, GDI32.dll, MFCO42D.DLL.

Prilikom startovanja aplikacije, prvo se u adresni prostor učitava EXE fajl (odnosno njegova datoteka na disku se preslikava u adresni prostor). Zatim se ustanovljava koje DLL-ove zahteva taj izvršni fajl. DLL-ovi se traže prvo u direktorijumu EXE fajla aplikacije; ako se ne nađu tamo, u tekućem direktorijumu procesa; ako nisu ni tamo, u Windows sistemskom direktorijumu (\Windows\System ili \WinNT\System32); ako nisu ni tamo, u Windows direktorijumu (\Windows ili \WinNT); ako nisu ni tamo, u svim direktorijumima koji se pominju u promenljivoj okruženja (*environment variable*) PATH. Ukoliko neki od DLL-ova ne može da bude nađen ni na jednoj od ovih lokacija, sistem javlja grešku i gasi aplikaciju. Da bi se proces startovao, potrebno je da se svi potrebni DLL-ovi preslikaju u njegov adresni prostor. Jednom kada se DLL učita, sve njegove eksportovane funkcije postaju dostupne aplikaciji kao i njene sopstvene. Kada se ugasi proces, DLL-ovi se oslobađaju. Ovakva procedura dinamičkog linkovanja usporava startovanje aplikacije, ali su u Microsoft-u procenili da to nije veliki nedostatak u odnosu na sve prednosti DLL-ova.

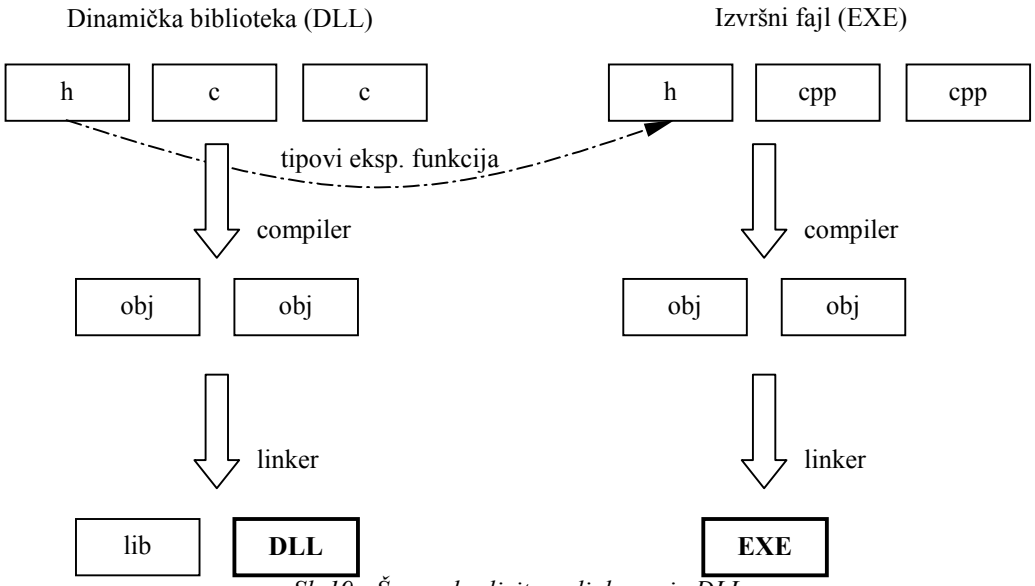
Funkcije u DLL-u se mogu eksportovati na dva načina: preko svog imena, i preko rednog broja (*ordinal value*). Prvi način (preko imena funkcije) se najčešće koristi, i treba ga upotrebljavati. Drugi način je ostao iz prethodnih verzija Windows-a pa je zadržan zbog kompatibilnosti, ali se ne preporučuje – označavanje funkcija preko rednog broja može da izazove zabunu. Postoji više načina da se generiše DLL. Jedan od njih funkcioniše na sledeći način. Funkcije koje se eksportuju treba da počinju specifikatorom __declspec(dllexport). Taj specifikator označava da ime te funkcije treba da se nalazi u eksport sekciji DLL-a. U glavnoj aplikaciji treba uključiti zaglavlje (.h fajl) koji sadrži prototip funkcije, sa extern “C” ispred deklaracije. Tako se deklariše ime funkcije u programu, a vezivanje tog imena za konkretan kod se odlaže za fazu linkovanja. Preporučuje se da se, kada je to moguće, DLL pravi u čistom C-u da bi eksportovane funkcije bile C funkcije. Kada se DLL pravi u C++, imena funkcija se kombinuju sa brojem i tipovima argumenata funkcije prilikom formiranja imena u eksport sekciji, što može da izazove probleme pri povezivanju. Ovo je naročito problematično ako se EXE pravi jednim, a DLL drugim kompajlerom, što je čest slučaj u praksi kada se pri pisanju aplikacije koriste gotove biblioteke raznih proizvođača softvera.

Gore opisani metod linkovanja DLL-a i EXE fajla je tzv. implicitno (podrazumevano) linkovanje, koje se obavlja prilikom podizanja aplikacije. Postupak kreiranja DLL-a i EXE fajla po implicitnoj šemi dat je na slici 18. Iz projekta u kome pravimo DLL, u projekat aplikacije prebacujemo tri fajla. Zaglavlje (.h) koje sadrži deklaracije eksportovanih funkcija DLL-a, koje se koristi u fazi prevođenja aplikacije iz izvornog u objektni kod. LIB fajl, koji u ovom slučaju ne sadrži kod funkcija (kao što bi sadržala statička LIB biblioteka) već služi samo za razrešavanje imena funkcija u fazi linkovanja. Na kraju,

DLL fajl, koji predstavlja gotovu biblioteku. Za startovanje aplikacije dovoljno je imati kreirani EXE fajl i pridruženi DLL.



Sl. 18 - Šema implicitnog linkovanja DLL-a



Sl. 19 - Šema eksplicitnog linkovanja DLL-a

Postoji još jedan metod linkovanja DLL-a i EXE aplikacije, tzv. eksplicitno linkovanje (na zahtev), prikazano šematski na na slici 19. Kod ovog metoda povezivanja DLL se na učitava pri podizanju aplikacije, već eksplicitno, API funkcijom LoadLibrary. Eksplicitno linkovan DLL se izbacuje iz memorije (oslobađa) API funkcijom FreeLibrary. Ukoliko se ne oslobodi sa FreeLibrary, DLL će svakako biti počišćen prilikom gašenja aplikacije. Kod eksplicitno linkovanog DLL-a pozivanje funkcija je nešto složenije nego kod implicitno linkovanog. Odmah je jasno da ne možemo funkcije pozivati na standardan način, jer to zahteva razrešavanje imena u toku prevođenja. Ako znamo tip funkcije koju pozivamo, npr. int f (int), deklarišemo pokazivač na funkciju tog tipa:

```
typedef int(*pf_type)(int);
pf_type p;
```

Zatim se pokazivač na željenu funkciju u DLL-u postavi pozivom API funkcije GetProcAddress:

```
p=(pf_type)GetProcAddress(handleDLL,"imefunkcije");
```

"imefunkcije" je naziv funkcije u eksport sekciji. Umesto imena funkcije može da se prosledi redni broj, pomoću C makroa MAKEINTRESOURCE, npr.

```
    p=(pf_type)GetProcAddress(handleDLL,MAKEINTRESOURCE(1));
```

Kada smo dobili pointer na funkciju, ona se poziva prosto:

```
    a=p(5);
```

Napravićemo jedan DLL, koji će eksportovati samo jednu funkciju, da bi na primeru videli celu proceduru. Otvoraćemo u Visual C++-u novi projekat, nazvati ga MojDLL (File>New>Win32 DLL). Kreiraćemo heder fajl MojDLL.h i izvorni fajl MojDLL.c. Hoćemo da naš DLL eksportuje funkciju koja će primati dva double argumenta i tip aritmetičke operacije, a koji će vraćati rezultat aritmetičke operacije. Tu funkciju, "Aritmetika" ćemo pozivati iz glavnog programa, iz dijaloga za aritmetiku.

1. U heder fajlu MojDLL.h se nalazi deklaracija eksportovane funkcije:

```
/* MojDLL.h */
extern "C" double Aritmetika(double operand1, double operand2
    ,long operacija);
```

Ovaj heder fajl se uključuje u projekat aplikacije.

2. Izvorni fajl MojDLL.c sadrži telo funkcije, i treba da izgleda ovako:

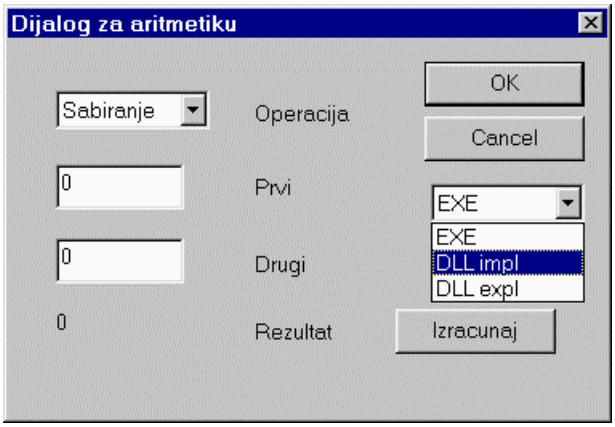
```
/* MojDLL.c */
__declspec(dllexport) double Aritmetika(double operand1,
    double operand2, long operacija)
{   double rezultat;
    switch(operacija)
    {   case 0:
            rezultat = operand1 + operand2;
            break;
        case 1:
            rezultat = operand1 - operand2;
            break;
        case 2:
            rezultat = operand1 * operand2;
            break;
        case 3:
            if(operand2==0) rezultat = 0;
            else rezultat = operand1 / operand2;
            break;
        default: rezultat = 0;
    }
    return rezultat;
}
```

Sam kod funkcije je prilično jasan i nema potrebe za objašnjenjima. Kao što je rečeno ranije, extern "C" deklaracija uvodi ime Aritmetika u glavni program, ali ostavlja razrešavanje tog imena za fazu linkovanja. Specifikator __declspec(dllexport) funkciju Aritmetika stavlja u eksport sekciju, i tako je čini vidljivom van DLL-a.

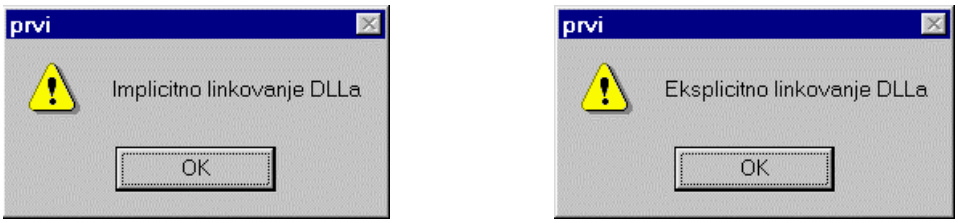
3. Postavimo konfiguraciju kompajlera na Release (Bulid>Set Active Configuration>Release), i prevedemo DLL. Fajlove MojDLL.h, MojDLL.lib i MojDLL.dll prebacimo u direktorijum gde je izvorni kod projekta aplikacije Prvi. Pošto ćemo demonstrirati oba načina linkovanja DLL-a, a da nebi pravili drugi DLL, iskopiramo MojDLL.dll u projekat Prvi još jedamput, sada pod imenom MojDLL1.dll. MojDLL.dll ćemo linkovati implicitno (zato nam trebaju i .h i .lib) a MojDLL1.dll eksplicitno.

4. Zatvorimo projekat DLL-a i otvorimo projekat aplikacije. U dijalogu IDD_DLGARITM, u resursima, dodamo jedna kombo boks sa resursnim identifikatorom IDC_GDEJEKOD (u property-jima za njega isključimo "sort", postavimo tip na "Drop List", i unesemo tri stavke u listu: "EXE", "DLL impl" i "DLL expl" (baš tim redosledom). Taj kombo boks će nam poslužiti za izbor metoda linkovanja DLL-a. Ideja je da se u sva tri slučaja uradi ista operacija (proračun), ali da se za "EXE" pozove već postojeći proračun u EXE fajlu, za "DLL impl" funkcija Aritmetika iz MojDLL.dll koji je linkovan

eksplicitno, a za "DLL expl" funkcija Aritmetika iz MojDLL1.dll koji je linkovan eksplicitno, i da se pri tome izbaci poruka koji tip linkovanja je korišćen. Na slici 20 je prikazan izgled dijaloga Aritmetika sa dodatim kombo boksom, a na slici 21 poruke (message box) koje će prikazivati aplikacija neposredno pre poziva proračuna u DLL-u.



Sl. 20 - Dijalog za aritmetiku u kome testiramo linkovanje DLL-a



Sl. 21 - Poruke koje izbacuje aplikacija kada se izabere vrsta linkovanja

5. U Class Wizard-u, u klasi CDlgAritmetika, za napravljeni kombo boks dodamo public member varijablu `int m_gdejekod`. U OnInitDialog za CDlgAritmetika (DlgAritmetika.cpp) dodamo inicijlizaciju ove member varijable: `m_gdejekod=0;` Ova member varijabla će nam omogućiti da pročitamo koju vrstu linkovanja je zatražio korisnik aplikacije.

6. Da bi pri implicitnom linkovanju imali pristup imenu funkcije, u fajlu gde se ono koristi (DlgAritmetika.cpp) dodamo direktivu za uključivanje heder fajla DLL-a:
   `#include "MojDLL.h"`

7. Za implicitno linkovanje se zahteva da se u parametre projekta, u delu gde se navodi koje LIB-ove zahteva EXE aplikacija, navede i MojDLL.lib. To se kod Visual C++-a nalazi u Project>Settings>Link, u polju "Object/Library modules". Ukoliko ne navedemo ime LIB-a u ovom polju, kompajler će javiti grešku u fazi linkovanja (ime funkcije Aritmetika će biti "unresolved external symbol").

8. Kod koji poziva funkciju Aritmetika dodaćemo u hendler OnIzracunaj iz klase CDlgAritmetika (fajl DlgAritmetika.cpp). Stari kod iz te funkcije, koji proračunava rezultat direktno, smestićemo u jednu `case` granu (u kojoj se nalazi komentar "stari kod").

```
void CDlgAritmetika::OnIzracunaj()
{   // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    switch(m_gdejekod)
    {   case 0:
            /* STARI KOD */
            switch(m_operacija)
            {   case 0:
            ...
            }
            break;
        case 1:
```

```
            m_rez=Aritmetika(m_prvi,m_drugi,(long)m_operacija);
            AfxMessageBox("Implicitno linkovanje DLLa");
            break;
        case 2:
            AfxMessageBox("Eksplicitno linkovanje DLLa");
            // za sada nista - dodacemo kasnije
            break;
        }
        UpdateData(FALSE);
}
```

Kada se izabere vrsta linkovanja DLL-a, tip aritmetičke operacije i unesu operandi, klikom na taster Izracunaj poziva se ovaj hendler. On prvo pročita kontrole dijaloga funkcijom UpdateData, zatim, u zavisnosti od m_gdejekod poziva ili stari kod za proračun, ili funkciju Aritmetika iz MojDll.dll, ili istu tu funkciju iz MojDll1.dll (ovo ćemo dodati kasnije, za sad samo ispisuje poruku). Kada se završi proračun, kontrole dijaloga se osveže funkcijom UpdateData. Na ovaj način (koraci 6-8) se u aplikaciju dodaje poziv funkcije iz implicitno linkovanog DLL-a.

9. Dodaćemo kod za eksplicitno linkovanje DLL-a MojDLL1.dll. U deklaraciju klase CPrviApp (fajl Prvi.h) dodamo public member varijablu `HINSTANCE hDLL` - hendl DLL-a. U funkciju InitInstance() dodamo poziv API funkcije LoadLibrary:

```
//Eksplicitno ucitavanje DLLa
if((hDLL=LoadLibrary("MojDLL1.dll"))==NULL)
    AfxMessageBox("MojDLL1.DLL nije ucitan");
```

U funkciju ExitInstance(), u istom fajlu, dodamo poziv FreeLibrary koji vrši oslobađanje DLL-a:

```
FreeLibrary(hDLL);
```

10. Sa mesta odakle će se pozvati funkcija iz DLL-a, a to je fajl CDlgAritmetika.h deklarišemo tip funkcije:

```
typedef double (*pfnAritmetika)(double,double,long);
```

Na kraju, u CDlgAritmetika.cpp, u funkciju OnIzracunaj(), u `switch(m_gdejekod)` strukturi, u `case 2` granu, gde za sada stoji samo ispis poruke o eksplicitnom linkovanju, dodamo kod koji ovo radi:

```
        case 2:
            pfnAritmetika fn;
            HINSTANCE hDLL;
            if((hDLL=GetModuleHandle("MojDLL1.dll"))==NULL)
                AfxMessageBox("DLL nije nadjen");
            fn=(pfnAritmetika)GetProcAddress(hDLL,"Aritmetika");
            m_rez=fn(m_prvi,m_drugi,(long)m_operacija);
            AfxMessageBox("Eksplicitno linkovanje DLLa");
            break;
```

GetModuleHandle vraća hendl DLL-a. Ukoliko DLL nije propisno linkovan, ova funkcija će vratiti grešku – NULL. Taj hendl koristimo, zajedno sa imenom funkcije iz eksport sekcije DLL-a, za dobijanje pointera na funkciju (adrese funkcije) fn. Mora da se napravi pointer na konkretan tip funkcije, u našem slučaju double f(double,double,long). Posle dobijanja pointera na funkciju fn, ona se poziva kao da joj je fn ime.

Kada se posle svih ovih izmena program prevede, u dijalogu Aritmetika (slika 20) će biti moguće pozvati proračun na sva tri načina. Poruke, kao na slici 21, govore kroz koji deo programa se prolazi, a korektan rezultat proračuna je pokazatelj da je stvarno pozvana funkcija Aritmetika iz DLL-a. Za naprednije tehnike upotrebe DLL-ova, kao što su upotreba DllMain(), odloženo linkovanje DLL-a, *forward*-ovanje (redirekcija) funkcija, *rebase*-ovanje DLL-a, *bind* DLL-a, čitalac se upućuje na knjigu [2].

Literatura

- **Windows programiranje.**
  [1] C. Petzold - "Programming Windows", 5th edition, Microsoft Press, 1998.
  [2] J. Richter - "Programming Applications for MS Windows", 4th edition, Microsoft Press, 1999. (starija izdanja ove knjige imaju naslov "Advanced Windows")
  [3] P. Yao, R. Leinecker - "Visual C++ 5 Biblija", Mikro knjiga, 1997.

- **C/C++.**
  [4] B. Kernighan, D. Ritchie - "Programski jezik C", Savremena administracija, 1989.
  [5] L. Kraus - "Programski jezik C sa rešenim zadacima", Mikro knjiga
  [6] D. Milićev - "Objektno orijentisano programiranje na jeziku C++", Mikro knjiga, 1995.
  [7] B. Eckel - "Thinking in C++", 2nd edition, Vol. 1, Vol. 2, Prentice-Hall, 2000.

- **Materijal u elektronskom obliku.**
  - On line help za Visual C++ je jedna od stvari koju ćete koristiti svakodnevno. Tu možete da nađete detaljne opise svih API funkcija, MFC klasa, primere kako ih treba koristiti; pogledajte Samples programe - iz njih se može dosta naučiti o konkretnoj upotrebi klasa. MSDN koji se isporučuje uz Visual C++ 6 sadrži ogroman broj tekstova o raznim temama vezanim za Windows programiranje, pa se i tu može naći odgovor na poneko pitanje. Jedan savet: nemojte čitati ovaj materijal u celosti od početka do kraja, jer će vam za to trebati nekoliko godina - već kad vam zatreba konkretno određena tema.
  - Na web adresi *http://www.bruceeckel.com* možete naći knjige [7], primere koda iz ovih knjiga, kao i još neke knjige.
  - Na web adresi *http://www.codeguru.com* možete naći, u sekciji Visual C++, obilje Windows API/MFC izvornog koda, gotovih kontrola, biblioteka, kao i diskusione forume gde možete dobiti savete od iskusnih programera i drugih korisnika koji se bave ovom oblašću.
  - Sajt DevCentral, na web adresi *http://devcentral.iftech.com/*, je dosta sličan prethodnom - nudi proširenja MFC biblioteke, diskusione forume, kao i mailing listu DevJournal NewsLetter *http://journal.iftech.com/*.
  - Internet pretraživač DevSeek, koji je specijalizovan za sajtove koji se bave programiranjem, nalazi se na adresi *http://www.devseek.com/*.
  - Na web adresi *http://mfcfaq.stingray.com/* nalazi se sajt MFC FAQ (*frequently asked questions* - često postavljana pitanja u vezi MFC-a).
  - Na web adresi *http://kiklop.etf.bg.ac.yu/~web1/te5prv/* nalaze se, u elektronskom obliku, skripte i zadaci za kurs "Programiranje u realnom vremenu" kao i izvorni C++ kod za razne primere iz ovog kursa.
  - Na web adresi *http://galeb.etf.bg.ac.yu/~uki/* je home page autora ovog praktikuma. Pitanja, primedbe i sugestije možete poslati na email adresu *uki@galeb.etf.bg.ac.yu*

Prilog A. Primer aplikacije bazirane na API-ju

Ovaj primer je preuzet iz knjige [1]. Predstavlja najjednostavniju moguću Windows aplikaciju pisanu u API-ju, u čistom C-u. Kreira se jedan prozor, na kome se ispisuje poruka. Iako jednostavan, ovaj primer poseduje sve osnovne delove Windows programa - ulaznu funkciju, petlju poruka, prozorsku funkciju sa switch-case obradom poruka.

```
/*------------------------------------------------------------
   HELLOWIN.C -- Displays "Hello, Windows 98!" in client area
                 (c) Charles Petzold, 1998
  ------------------------------------------------------------*/

#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{   static TCHAR szAppName[] = TEXT ("HelloWin") ;
    HWND         hwnd ;
    MSG          msg ;
    WNDCLASS     wndclass ;

    wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc   = WndProc ;
    wndclass.cbClsExtra    = 0 ;
    wndclass.cbWndExtra    = 0 ;
    wndclass.hInstance     = hInstance ;
    wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
    wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName  = NULL ;
    wndclass.lpszClassName = szAppName ;

    if (!RegisterClass (&wndclass))
    {   MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                    szAppName, MB_ICONERROR) ;
        return 0 ;
    }
    hwnd = CreateWindow (szAppName, // window class name
        TEXT ("The Hello Program"), // window caption
        WS_OVERLAPPEDWINDOW,        // window style
        CW_USEDEFAULT,              // initial x position
        CW_USEDEFAULT,              // initial y position
        CW_USEDEFAULT,              // initial x size
        CW_USEDEFAULT,              // initial y size
        NULL,                       // parent window handle
        NULL,                       // window menu handle
        hInstance,                  // program instance handle
        NULL) ;                     // creation parameters

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {   TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
```

```
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam)
{   HDC         hdc ;
    PAINTSTRUCT ps ;
    RECT        rect ;

    switch (message)
    {   case WM_CREATE:
            PlaySound (TEXT ("hellowin.wav"), NULL, SND_FILENAME |
                SND_ASYNC) ;
            return 0 ;
        case WM_PAINT:
            hdc = BeginPaint (hwnd, &ps) ;
            GetClientRect (hwnd, &rect) ;
            DrawText (hdc, TEXT ("Hello, Windows 98!"), -1, &rect,
                DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
            EndPaint (hwnd, &ps) ;
            return 0 ;
        case WM_DESTROY:
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

Prilog B. MFC aplikacija Prvi

U ovom prilogu dat je kompletan kod aplikacije koja je deo po deo kreirana u ovom kursu. Fajlovi su podeljene u četiri grupe:

1. Osnovni 1: Prvi.h, Prvi.cpp; MainFrm.h, MainFrm.cpp; PrviDoc.h, PrviDoc.cpp; PrviView.h, PrviView.cpp
2. Osnovni 2: ChildFrm.h, ChildFrm.cpp; resource.h, Prvi.rc; StdAfx.h, StdAfx.cpp
3. Dijalozi: DlgAritmetika.h, DlgAritmetika.cpp; DlgNemodalni.h, DlgNemodalni.cpp
4. Property Pages: PSProperties.h, PSProperties.cpp; PPPrva.h, PPPrva.cpp; PPDruga.h, PPDruga.cpp

B1. Grupa fajlova Osnovni 1:

- **B1-1: Prvi.h**

```
// prvi.h : main header file for the PRVI application
//
#if !defined
(AFX_PRVI_H__92680FA7_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
#define AFX_PRVI_H__92680FA7_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////
// CPrviApp:
// See prvi.cpp for the implementation of this class
//
class CPrviApp : public CWinApp
{   public:
        int m_app_refresh;
        int m_par1;
        int m_par2;
        BOOL m_prop_en;
        HINSTANCE hDLL;
        CPrviApp();
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPrviApp)
    public:
        virtual BOOL InitInstance();
        virtual int ExitInstance();
    //}}AFX_VIRTUAL
// Implementation
    //{{AFX_MSG(CPrviApp)
        afx_msg void OnAppAbout();
    // NOTE-the ClassWizard will add and remove member funct. here.
    // DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
/////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
// immediately before the previous line.
#endif
//!defined
//(AFX_PRVI_H__92680FA7_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
```

- **B1-1: Prvi.cpp**

```
// prvi.cpp : Defines the class behaviors for the application.
//
#include "stdafx.h"
#include "prvi.h"
#include "MainFrm.h"
#include "ChildFrm.h"
#include "prviDoc.h"
#include "prviView.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////
// CPrviApp
BEGIN_MESSAGE_MAP(CPrviApp, CWinApp)
    //{{AFX_MSG_MAP(CPrviApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    // NOTE-the ClassWizard will add and remove mapping macros here.
    // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////
// CPrviApp construction
CPrviApp::CPrviApp()
{   // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////
// The one and only CPrviApp object
CPrviApp theApp;

/////////////////////////////////////////////////////////////////////////
// CPrviApp initialization
BOOL CPrviApp::InitInstance()
{   AfxEnableControlContainer();
    // Standard initialization
    // If you arenot using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.
#ifdef _AFXDLL
    Enable3dControls(); // Call this when using MFC in a shared DLL
```

```
#else
    Enable3dControlsStatic(); // Linking to MFC statically
#endif
    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    //SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    SetRegistryKey(_T("VTA"));
    LoadStdProfileSettings(); //Load std INI file options(incl. MRU)
    m_app_refresh = GetProfileInt("Init","refresh",TRUE);
    m_par1=1;
    m_par2=2;
    m_prop_en=FALSE;
    // Register the application's document templates.  Doc. templates
    // serve as the connection between docs, frame windows and views.
    CMultiDocTemplate* pDocTemplate;
    pDocTemplate = new CMultiDocTemplate(
        IDR_PRVITYPE,
        RUNTIME_CLASS(CPrviDoc),
        RUNTIME_CLASS(CChildFrame), // custom MDI child frame
        RUNTIME_CLASS(CPrviView));
    AddDocTemplate(pDocTemplate);
    // create main MDI Frame window
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    m_pMainWnd = pMainFrame;
    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo)) return FALSE;
    // The main window has been initialized, so show and update it.
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();
    //Eksplicitno ucitavanje DLLa
    if((hDLL=LoadLibrary("MojDLL1.dll"))==NULL)
        AfxMessageBox("MojDLL1.DLL nije ucitan");
    return TRUE;
}

/////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{   public:
        CAboutDlg();
// Dialog Data
    //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA
// ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);
            // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
    protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{   //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{   CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CPrviApp::OnAppAbout()
{   CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CPrviApp commands
int CPrviApp::ExitInstance()
{   // TODO: Add your special. code here and/or call the base class
    WriteProfileInt("Init","refresh",m_app_refresh);
    FreeLibrary(hDLL);
    return CWinApp::ExitInstance();
}
```

- **B1-2: MainFrm.h**

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////
#if !defined
(AFX_MAINFRM_H__92680FAB_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
#define
AFX_MAINFRM_H__92680FAB_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
#include "PSProperties.h"

class CMainFrame : public CMDIFrameWnd
{   DECLARE_DYNAMIC(CMainFrame)
    public:
        CMainFrame();
// Attributes
    public:
        CPSProperties *m_ps;
// Operations
    public:
// Overrides
    // ClassWizard generated virtual function overrides
```

```
    //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL
// Implementation
    public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
    protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;
        CToolBar    m_wndMyTools;
// Generated message map functions
    protected:
    //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnMojeAritm();
        afx_msg void OnMojeNemod();
        afx_msg void OnMytoolAction();
        afx_msg void OnMytoolRefresh();
        afx_msg void OnUpdateMytoolRefresh(CCmdUI* pCmdUI);
        afx_msg void OnViewMytools();
        afx_msg void OnUpdateViewMytools(CCmdUI* pCmdUI);
        afx_msg void OnFileProperties();
        afx_msg void OnUpdateFileProperties(CCmdUI* pCmdUI);
    //}}AFX_MSG
    //rucno
        afx_msg void OnUpdateStatusRefresh(CCmdUI *pCmdUI);
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
// immediately before the previous line.
#endif
//!defined
//(AFX_MAINFRM_H__92680FAB_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
```

- **B1-2: MainFrm.cpp**

```
// MainFrm.cpp : implementation of the CMainFrame class
//
#include "stdafx.h"
#include "prvi.h"
#include "MainFrm.h"
#include "DlgAritmetika.h"
#include "DlgNemodalni.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////
// CMainFrame
IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
```

```
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_COMMAND(ID_MOJE_ARITM, OnMojeAritm)
    ON_COMMAND(ID_MOJE_NEMOD, OnMojeNemod)
    ON_COMMAND(ID_MYTOOL_ACTION, OnMytoolAction)
    ON_COMMAND(ID_MYTOOL_REFRESH, OnMytoolRefresh)
    ON_UPDATE_COMMAND_UI(ID_MYTOOL_REFRESH, OnUpdateMytoolRefresh)
    ON_COMMAND(ID_VIEW_MYTOOLS, OnViewMytools)
    ON_UPDATE_COMMAND_UI(ID_VIEW_MYTOOLS, OnUpdateViewMytools)
    ON_COMMAND(ID_FILE_PROPERTIES, OnFileProperties)
    ON_UPDATE_COMMAND_UI(ID_FILE_PROPERTIES, OnUpdateFileProperties)
    //}}AFX_MSG_MAP
    // Global help commands
    ON_COMMAND(ID_HELP_FINDER, CMDIFrameWnd::OnHelpFinder)
    ON_COMMAND(ID_HELP, CMDIFrameWnd::OnHelp)
    ON_COMMAND(ID_CONTEXT_HELP, CMDIFrameWnd::OnContextHelp)
    ON_COMMAND(ID_DEFAULT_HELP, CMDIFrameWnd::OnHelpFinder)
    //rucno
    ON_UPDATE_COMMAND_UI(ID_INDICATOR_REF,OnUpdateStatusRefresh)
END_MESSAGE_MAP()

static UINT indicators[] =
{   ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
    ID_INDICATOR_REF,
};

static UINT BASED_CODE buttons[] =
{   ID_MYTOOL_REFRESH,
    ID_MYTOOL_ACTION
};

////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction
CMainFrame::CMainFrame()
{   // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{   if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1) return -1;
    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {   TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }
    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
        sizeof(indicators)/sizeof(UINT)))
    {   TRACE0("Failed to create status bar\n");
        return -1;      // fail to create
    }
    // TODO: Remove this if you don't want tool tips
    //or a resizeable toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);
```

```
    // TODO: Delete these 3 lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);
    //MyTools
    m_wndMyTools.Create(this,CBRS_RIGHT | CBRS_TOOLTIPS | CBRS_FLYBY
        | WS_VISIBLE);
    m_wndMyTools.LoadToolBar(IDR_MYTOOLS);
    m_wndMyTools.EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndMyTools);
    m_wndMyTools.SetButtonStyle(
        m_wndMyTools.CommandToIndex(ID_MYTOOL_REFRESH),
        TBBS_CHECKBOX);
    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{   // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs
    return CMDIFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics
#ifdef _DEBUG
void CMainFrame::AssertValid() const
{   CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{   CMDIFrameWnd::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
void CMainFrame::OnMojeAritm()
{   // TODO: Add your command handler code here
    CDlgAritmetika dlg;
    dlg.DoModal();
}

void CMainFrame::OnMojeNemod()
{   // TODO: Add your command handler code here
    CDlgNemodalni *pdlg;
    pdlg=new CDlgNemodalni();
}

void CMainFrame::OnUpdateStatusRefresh(CCmdUI *pCmdUI)
{   CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    if(pApp->m_app_refresh) pCmdUI->Enable(TRUE);
    else pCmdUI->Enable(FALSE);
}

void CMainFrame::OnMytoolAction()
{   // TODO: Add your command handler code here
    CDlgAritmetika dlg;
    dlg.DoModal();
}
```

```
void CMainFrame::OnMytoolRefresh()
{   // TODO: Add your command handler code here
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    pApp->m_app_refresh=!(pApp->m_app_refresh);
}

void CMainFrame::OnUpdateMytoolRefresh(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    if(pApp->m_app_refresh) pCmdUI->SetCheck(TRUE);
    else pCmdUI->SetCheck(FALSE);
}

void CMainFrame::OnViewMytools()
{   // TODO: Add your command handler code here
    BOOL bv=m_wndMyTools.GetStyle() & WS_VISIBLE;
    int ns=bv?SW_HIDE:SW_SHOWNORMAL;
    m_wndMyTools.ShowWindow(ns);
    RecalcLayout();
}

void CMainFrame::OnUpdateViewMytools(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    BOOL bv=m_wndMyTools.GetStyle() & WS_VISIBLE;
    if(bv) pCmdUI->SetCheck(TRUE);
    else pCmdUI->SetCheck(FALSE);
}

void CMainFrame::OnFileProperties()
{   // TODO: Add your command handler code here
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    if(!pApp->m_prop_en)
    {   m_ps=new CPSProperties("Properties");
        pApp->m_prop_en=TRUE;
    }
    else
    {   m_ps->DestroyWindow();
        pApp->m_prop_en=FALSE;
    }
}

void CMainFrame::OnUpdateFileProperties(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    pCmdUI->SetCheck(pApp->m_prop_en);
}
```

- **B1-3: PrviDoc.h**

```
// prviDoc.h : interface of the CPrviDoc class
//
/////////////////////////////////////////////////////////////////////////
#if !defined
(AFX_PRVIDOC_H__92680FAF_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
#define
AFX_PRVIDOC_H__92680FAF_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
```

```
class CPrviDoc : public CDocument
{   protected: // create from serialization only
        CPrviDoc();
    DECLARE_DYNCREATE(CPrviDoc)
// Attributes
    public:
        int sadrzaj;
        int undobuf;
        BOOL undoenabled;
// Operations
    public:
        void NapuniUndoBafer(void);
        void IzvrsiUndo(void);
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPrviDoc)
    public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL
// Implementation
    public:
        virtual ~CPrviDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
    protected:
// Generated message map functions
    protected:
    //{{AFX_MSG(CPrviDoc)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
// immediately before the previous line.
#endif
//!defined
//(AFX_PRVIDOC_H__92680FAF_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
```

- **B1-3: PrviDoc.cpp**

```
// prviDoc.cpp : implementation of the CPrviDoc class
//
#include "stdafx.h"
#include "prvi.h"
#include "prviDoc.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////
// CPrviDoc
IMPLEMENT_DYNCREATE(CPrviDoc, CDocument)
BEGIN_MESSAGE_MAP(CPrviDoc, CDocument)
```

```
    //{{AFX_MSG_MAP(CPrviDoc)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////
// CPrviDoc construction/destruction
CPrviDoc::CPrviDoc()
{   // TODO: add one-time construction code here
    sadrzaj=15;
    undobuf=0;
    undoenabled=FALSE;
}

CPrviDoc::~CPrviDoc()
{
}

BOOL CPrviDoc::OnNewDocument()
{   if (!CDocument::OnNewDocument()) return FALSE;
    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)
    return TRUE;
}

/////////////////////////////////////////////////////////////////////////
// CPrviDoc serialization
void CPrviDoc::Serialize(CArchive& ar)
{   if (ar.IsStoring())
    {   // TODO: add storing code here
        ar << sadrzaj;
        undoenabled=FALSE;
    }
    else
    {   // TODO: add loading code here
        ar >> sadrzaj;
    }
}

/////////////////////////////////////////////////////////////////////////
// CPrviDoc diagnostics
#ifdef _DEBUG
void CPrviDoc::AssertValid() const
{   CDocument::AssertValid();
}

void CPrviDoc::Dump(CDumpContext& dc) const
{   CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////
// CPrviDoc commands
void CPrviDoc::NapuniUndoBafer(void)
{   undobuf = sadrzaj;
    undoenabled = TRUE;
}

void CPrviDoc::IzvrsiUndo(void)
{   if(undoenabled) sadrzaj = undobuf;
}
```

- **B1-4: PrviView.h**

```
// prviView.h : interface of the CPrviView class
//
/////////////////////////////////////////////////////////////////////////
#if !defined
(AFX_PRVIVIEW_H__92680FB1_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
#define
AFX_PRVIVIEW_H__92680FB1_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CPrviView : public CView
{   protected: // create from serialization only
        CPrviView();
    DECLARE_DYNCREATE(CPrviView)
// Attributes
    public:
        CPrviDoc* GetDocument();
        int mousex;
        int mousey;
        BOOL refresh;
// Operations
    public:
// Overrides
// ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPrviView)
    public:
        virtual void OnDraw(CDC* pDC); //overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    //}}AFX_VIRTUAL
// Implementation
    public:
        virtual ~CPrviView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
    protected:
// Generated message map functions
    protected:
    //{{AFX_MSG(CPrviView)
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
        afx_msg void OnPaint();
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt,UINT nFlags);
        afx_msg void OnEditUndo();
        afx_msg void OnUpdateEditUndo(CCmdUI* pCmdUI);
        afx_msg void OnMytoolRefresh();
        afx_msg void OnUpdateMytoolRefresh(CCmdUI* pCmdUI);
        afx_msg void OnEditCopy();
        afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
        afx_msg void OnEditPaste();
        afx_msg void OnUpdateEditPaste(CCmdUI* pCmdUI);
```

```
    //}}AFX_MSG
    //rucno
        afx_msg void OnUpdateStatusRefresh(CCmdUI *pCmdUI);
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in prviView.cpp
inline CPrviDoc* CPrviView::GetDocument()
   { return (CPrviDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
// immediately before the previous line.

#endif
//!defined
//(AFX_PRVIVIEW_H__92680FB1_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
```

- **B1-4: PrviView.cpp**

```
// prviView.cpp : implementation of the CPrviView class
//
#include "stdafx.h"
#include "prvi.h"
#include "prviDoc.h"
#include "prviView.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////
// CPrviView
IMPLEMENT_DYNCREATE(CPrviView, CView)
BEGIN_MESSAGE_MAP(CPrviView, CView)
    //{{AFX_MSG_MAP(CPrviView)
    ON_WM_MOUSEMOVE()
    ON_WM_PAINT()
    ON_WM_LBUTTONDOWN()
    ON_WM_RBUTTONDOWN()
    ON_WM_KEYDOWN()
    ON_COMMAND(ID_EDIT_UNDO, OnEditUndo)
    ON_UPDATE_COMMAND_UI(ID_EDIT_UNDO, OnUpdateEditUndo)
    ON_COMMAND(ID_MYTOOL_REFRESH, OnMytoolRefresh)
    ON_UPDATE_COMMAND_UI(ID_MYTOOL_REFRESH, OnUpdateMytoolRefresh)
    ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
    ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
    ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
    //rucno
    ON_UPDATE_COMMAND_UI(ID_INDICATOR_REF,OnUpdateStatusRefresh)
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////
// CPrviView construction/destruction
CPrviView::CPrviView()
{   // TODO: add construction code here
    mousex=0;
    mousey=0;
    //refresh=TRUE;
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    refresh = pApp->m_app_refresh;
}

CPrviView::~CPrviView()
{   CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    pApp->m_app_refresh = refresh;
}

BOOL CPrviView::PreCreateWindow(CREATESTRUCT& cs)
{   // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs
    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////
// CPrviView drawing
void CPrviView::OnDraw(CDC* pDC)
{   CPrviDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    CString izlaz;
    izlaz.Format("Koordinate misa=(%d,%d) Sadrzaj=%d         "
        ,mousex,mousey,GetDocument()->sadrzaj);
    pDC->TextOut(20,20,izlaz);
    // TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////
// CPrviView printing
BOOL CPrviView::OnPreparePrinting(CPrintInfo* pInfo)
{   // default preparation
    return DoPreparePrinting(pInfo);
}

void CPrviView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{   // TODO: add extra initialization before printing
}

void CPrviView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{   // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////
// CPrviView diagnostics
#ifdef _DEBUG
void CPrviView::AssertValid() const
{   CView::AssertValid();
}

void CPrviView::Dump(CDumpContext& dc) const
{   CView::Dump(dc);
}
```

```
CPrviDoc* CPrviView::GetDocument() // non-debug version is inline
{   ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CPrviDoc)));
    return (CPrviDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////
// CPrviView message handlers
void CPrviView::OnMouseMove(UINT nFlags, CPoint point)
{   // TODO: Add your message handler code here and/or call default
 //CMDIFrameWnd *pMainFrame=(CMDIFrameWnd *)AfxGetApp()->m_pMainWnd;
 //CPrviDoc *pDoc=GetDocument();
    mousex=point.x;
    mousey=point.y;
    if(refresh) Invalidate(FALSE);
    CView::OnMouseMove(nFlags, point);
}

void CPrviView::OnPaint()
{   CPaintDC dc(this); // device context for painting
    // TODO: Add your message handler code here
    CString izlaz;
    izlaz.Format("Koordinate misa=(%d,%d) Sadrzaj=%d         "
        ,mousex,mousey,GetDocument()->sadrzaj);
    dc.TextOut(20,20,izlaz);
    // Do not call CView::OnPaint() for painting messages
}

void CPrviView::OnLButtonDown(UINT nFlags, CPoint point)
{   // TODO: Add your message handler code here and/or call default
    Invalidate(TRUE);
    CView::OnLButtonDown(nFlags, point);
}

void CPrviView::OnRButtonDown(UINT nFlags, CPoint point)
{   // TODO: Add your message handler code here and/or call default
    refresh=!refresh;
    CView::OnRButtonDown(nFlags, point);
}

void CPrviView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{   // TODO: Add your message handler code here and/or call default
    GetDocument()->NapuniUndoBafer();
    GetDocument()->sadrzaj=nChar;
    GetDocument()->SetModifiedFlag(TRUE);
    if(refresh) Invalidate(FALSE);
    CView::OnKeyDown(nChar, nRepCnt, nFlags);
}

void CPrviView::OnEditUndo()
{   // TODO: Add your command handler code here
    GetDocument()->IzvrsiUndo();
    GetDocument()->undoenabled=FALSE;
    Invalidate(FALSE);
}

void CPrviView::OnUpdateEditUndo(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    if(GetDocument()->undoenabled) pCmdUI->Enable(TRUE);
    else pCmdUI->Enable(FALSE);
}
```

```
void CPrviView::OnUpdateStatusRefresh(CCmdUI *pCmdUI)
{   if(refresh) pCmdUI->Enable(TRUE);
    else pCmdUI->Enable(FALSE);
}

void CPrviView::OnMytoolRefresh()
{   // TODO: Add your command handler code here
    refresh = !refresh;
}

void CPrviView::OnUpdateMytoolRefresh(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    if(refresh) pCmdUI->SetCheck(TRUE);
    else pCmdUI->SetCheck(FALSE);
}

void CPrviView::OnEditCopy()
{   // TODO: Add your command handler code here
    HANDLE hCopy;
    LPSTR lpCopy;
    if (OpenClipboard())
    {   BeginWaitCursor();
        EmptyClipboard();
        if ((hCopy = (HANDLE) ::GlobalAlloc (GHND, 2*sizeof(char)))
            != NULL)
        {   lpCopy = (LPSTR) ::GlobalLock((HGLOBAL) hCopy);
            lpCopy[0]=(char)(GetDocument()->sadrzaj);
            lpCopy[1]='\0';
            ::GlobalUnlock((HGLOBAL) hCopy);
        }
        else
        {   AfxMessageBox("Greska u alokaciji memorije");
        }
        SetClipboardData (CF_TEXT, hCopy);
        CloseClipboard();
        EndWaitCursor();
    }
}

void CPrviView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    pCmdUI->Enable(TRUE);
}

void CPrviView::OnEditPaste()
{   // TODO: Add your command handler code here
    HANDLE hCopy;
    LPSTR lpCopy;
    if (OpenClipboard())
    {   BeginWaitCursor();
        hCopy = (HANDLE) ::GetClipboardData(CF_TEXT);
        if (hCopy != NULL)
        {   lpCopy=(LPSTR) ::GlobalLock((HGLOBAL)hCopy);
            GetDocument()->NapuniUndoBafer();
            GetDocument()->sadrzaj=lpCopy[0];
            ::GlobalUnlock((HGLOBAL)hCopy);
            GetDocument()->SetModifiedFlag(TRUE);
            GetDocument()->UpdateAllViews(NULL);
        }
        CloseClipboard();
```

```
        EndWaitCursor();
    }
}

void CPrviView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{   // TODO: Add your command update UI handler code here
    pCmdUI->Enable(::IsClipboardFormatAvailable(CF_TEXT));
}
```

B2. Grupa fajlova Osnovni 2:

- **B2-1: ChildFrm.h**

```
// ChildFrm.h : interface of the CChildFrame class
//
/////////////////////////////////////////////////////////////////////////////
#if !defined
(AFX_CHILDFRM_H__92680FAD_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
#define
AFX_CHILDFRM_H__92680FAD_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CChildFrame : public CMDIChildWnd
{   DECLARE_DYNCREATE(CChildFrame)
    public:
        CChildFrame();
// Attributes
    public:
// Operations
    public:
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChildFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL
// Implementation
    public:
        virtual ~CChildFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
// Generated message map functions
    protected:
    //{{AFX_MSG(CChildFrame)
    // NOTE - the ClassWizard will add and remove member funcs here.
    // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
// immediately before the previous line.

#endif
// !defined
```

```
// (AFX_CHILDFRM_H__92680FAD_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
```

- **B2-1: ChildFrm.cpp**

```
// ChildFrm.cpp : implementation of the CChildFrame class
//
#include "stdafx.h"
#include "prvi.h"
#include "ChildFrm.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CChildFrame
IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)
BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
    //{{AFX_MSG_MAP(CChildFrame)
    // NOTE-the ClassWizard will add and remove mapping macros here.
    // DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChildFrame construction/destruction
CChildFrame::CChildFrame()
{   // TODO: add member initialization code here
}

CChildFrame::~CChildFrame()
{
}

BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{   // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs
    return CMDIChildWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CChildFrame diagnostics
#ifdef _DEBUG
void CChildFrame::AssertValid() const
{   CMDIChildWnd::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
{   CMDIChildWnd::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CChildFrame message handlers
```

- **B2-2: resource.h**

```
//{{NO_DEPENDENCIES}}
```

```
// Microsoft Developer Studio generated include file.
// Used by prvi.rc
//
#define IDD_ABOUTBOX                    100
#define IDD_PP_1                        102
#define IDR_MAINFRAME                   128
#define IDR_PRVITYPE                    129
#define IDD_DLGARITM                    130
#define IDD_DLGNEMOD                    133
#define IDR_MYTOOLS                     134
#define IDD_PP_2                        134
#define IDD_PS_PROPERTIES               135
#define IDC_OPERACIJA                   1000
#define IDC_PRVI                        1002
#define IDC_DRUGI                       1003
#define IDC_REZ                         1004
#define IDC_IZRACUNAJ                   1005
#define IDC_AKCIJA                      1006
#define IDC_PAR1                        1007
#define IDC_PAR2                        1008
#define IDC_UPDATE                      1009
#define IDC_GDEJEKOD                    1011
#define ID_MOJESTAVKE_KREIRAJPROZOR     32771
#define ID_MOJE_PROBA                   32772
#define ID_MOJEOPCIJE_ARITMETIKA        32773
#define ID_MOJE_ARITM                   32773
#define ID_MOJE_NEMOD                   32774
#define ID_MYTOOL_REFRESH               32775
#define ID_MYTOOL_ACTION                32776
#define ID_VIEW_MYTOOLS                 32777
#define ID_FILE_PROPERTIES              32778
#define ID_INDICATOR_REF                59142

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        137
#define _APS_NEXT_COMMAND_VALUE         32779
#define _APS_NEXT_CONTROL_VALUE         1012
#define _APS_NEXT_SYMED_VALUE           102
#endif
#endif
```

- **B2-3: Prvi.rc**

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS
```

```
/////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources
#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

/////////////////////////////////////////////////////////////////////////////
//
// Icon
//
// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\prvi.ico"
IDR_PRVITYPE            ICON    DISCARDABLE     "res\\prviDoc.ico"

/////////////////////////////////////////////////////////////////////////////
//
// Bitmap
//
IDR_MAINFRAME           BITMAP  MOVEABLE PURE   "res\\Toolbar.bmp"

/////////////////////////////////////////////////////////////////////////////
//
// Toolbar
//
IDR_MAINFRAME TOOLBAR DISCARDABLE  16, 15
BEGIN
    BUTTON      ID_FILE_NEW
    BUTTON      ID_FILE_OPEN
    BUTTON      ID_FILE_SAVE
    SEPARATOR
    BUTTON      ID_EDIT_CUT
    BUTTON      ID_EDIT_COPY
    BUTTON      ID_EDIT_PASTE
    SEPARATOR
    BUTTON      ID_FILE_PRINT
    BUTTON      ID_APP_ABOUT
    BUTTON      ID_CONTEXT_HELP
END

/////////////////////////////////////////////////////////////////////////////
//
// Menu
//
IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",                ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",            ID_FILE_OPEN
        MENUITEM "&Properties",                 ID_FILE_PROPERTIES
        MENUITEM SEPARATOR
        MENUITEM "P&rint Setup...",             ID_FILE_PRINT_SETUP
        MENUITEM SEPARATOR
        MENUITEM "Recent File",             ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
```

```
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
        MENUITEM "&My tools",                   ID_VIEW_MYTOOLS
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&Help Topics",                ID_HELP_FINDER
        MENUITEM SEPARATOR
        MENUITEM "&About prvi...",              ID_APP_ABOUT
    END
    POPUP "&Moje opcije"
    BEGIN
        MENUITEM "&Aritmetika",                 ID_MOJE_ARITM
        MENUITEM "&Nemodalni",                  ID_MOJE_NEMOD
    END
END

IDR_PRVITYPE MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",                ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",            ID_FILE_OPEN
        MENUITEM "&Close",                      ID_FILE_CLOSE
        MENUITEM "&Save\tCtrl+S",               ID_FILE_SAVE
        MENUITEM "Save &As...",                 ID_FILE_SAVE_AS
        MENUITEM SEPARATOR
        MENUITEM "&Print...\tCtrl+P",           ID_FILE_PRINT
        MENUITEM "Print Pre&view",              ID_FILE_PRINT_PREVIEW
        MENUITEM "P&rint Setup...",             ID_FILE_PRINT_SETUP
        MENUITEM SEPARATOR
        MENUITEM "Recent File",             ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo\tCtrl+Z",               ID_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\tCtrl+X",                ID_EDIT_CUT
        MENUITEM "&Copy\tCtrl+C",               ID_EDIT_COPY
        MENUITEM "&Paste\tCtrl+V",              ID_EDIT_PASTE
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
        MENUITEM "&My tools",                   ID_VIEW_MYTOOLS
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&New Window",                 ID_WINDOW_NEW
        MENUITEM "&Cascade",                    ID_WINDOW_CASCADE
        MENUITEM "&Tile",                       ID_WINDOW_TILE_HORZ
        MENUITEM "&Arrange Icons",              ID_WINDOW_ARRANGE
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&Help Topics",                ID_HELP_FINDER
```

```
        MENUITEM SEPARATOR
        MENUITEM "&About prvi...",              ID_APP_ABOUT
    END
END

//////////////////////////////////////////////////////////////////////
//
// Accelerator
//
IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
    "C",            ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
    "N",            ID_FILE_NEW,            VIRTKEY, CONTROL, NOINVERT
    "O",            ID_FILE_OPEN,           VIRTKEY, CONTROL, NOINVERT
    "P",            ID_FILE_PRINT,          VIRTKEY, CONTROL, NOINVERT
    "R",            ID_MOJE_ARITM,          VIRTKEY, CONTROL, NOINVERT
    "S",            ID_FILE_SAVE,           VIRTKEY, CONTROL, NOINVERT
    "V",            ID_EDIT_PASTE,          VIRTKEY, CONTROL, NOINVERT
    VK_BACK,        ID_EDIT_UNDO,           VIRTKEY, ALT, NOINVERT
    VK_DELETE,      ID_EDIT_CUT,            VIRTKEY, SHIFT, NOINVERT
    VK_F1,          ID_HELP,                VIRTKEY, NOINVERT
    VK_F1,          ID_CONTEXT_HELP,        VIRTKEY, SHIFT, NOINVERT
    VK_F6,          ID_NEXT_PANE,           VIRTKEY, NOINVERT
    VK_F6,          ID_PREV_PANE,           VIRTKEY, SHIFT, NOINVERT
    VK_INSERT,      ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
    VK_INSERT,      ID_EDIT_PASTE,          VIRTKEY, SHIFT, NOINVERT
    "X",            ID_EDIT_CUT,            VIRTKEY, CONTROL, NOINVERT
    "Z",            ID_EDIT_UNDO,           VIRTKEY, CONTROL, NOINVERT
END

//////////////////////////////////////////////////////////////////////
//
// Dialog
//
IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 217, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About prvi"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,11,17,20,20
    LTEXT           "prvi Version
1.0",IDC_STATIC,40,10,119,8,SS_NOPREFIX
    LTEXT           "Copyright (C) 1900",IDC_STATIC,40,25,119,8
    DEFPUSHBUTTON   "OK",IDOK,178,7,32,14,WS_GROUP
END

IDD_PP_1 DIALOG DISCARDABLE  0, 0, 195, 127
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Prva strana"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDC_PAR1,50,32,40,14,ES_AUTOHSCROLL
    LTEXT           "Parametar 1",IDC_STATIC,102,34,38,8
    PUSHBUTTON      "Update",IDC_UPDATE,49,61,50,14
END

IDD_PP_2 DIALOG DISCARDABLE  0, 0, 195, 127
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Druga strana"
FONT 8, "MS Sans Serif"
BEGIN
```

```
    EDITTEXT        IDC_PAR2,38,41,40,14,ES_AUTOHSCROLL
    LTEXT           "Parametar 2",IDC_STATIC,89,44,38,8
    PUSHBUTTON      "Update",IDC_UPDATE,39,72,50,14
END

IDD_PS_PROPERTIES DIALOG DISCARDABLE  0, 0, 195, 127
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Properties"
FONT 8, "MS Sans Serif"
BEGIN
END

#ifndef _MAC
/////////////////////////////////////////////////////////////////////////////
//
// Version
//
VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904B0"
        BEGIN
            VALUE "CompanyName", "\0"
            VALUE "FileDescription", "prvi MFC Application\0"
            VALUE "FileVersion", "1, 0, 0, 1\0"
            VALUE "InternalName", "prvi\0"
            VALUE "LegalCopyright", "Copyright (C) 1900\0"
            VALUE "LegalTrademarks", "\0"
            VALUE "OriginalFilename", "prvi.EXE\0"
            VALUE "ProductName", "prvi Application\0"
            VALUE "ProductVersion", "1, 0, 0, 1\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END
#endif    // !_MAC

/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//
#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
```

```
        LEFTMARGIN, 7
        RIGHTMARGIN, 210
        TOPMARGIN, 7
        BOTTOMMARGIN, 48
    END

    IDD_PP_1, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 188
        TOPMARGIN, 7
        BOTTOMMARGIN, 120
    END

    IDD_PP_2, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 188
        TOPMARGIN, 7
        BOTTOMMARGIN, 120
    END

    IDD_PS_PROPERTIES, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 188
        TOPMARGIN, 7
        BOTTOMMARGIN, 120
    END
END
#endif    // APSTUDIO_INVOKED

////////////////////////////////////////////////////////////////////////
//
// String Table
//
STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    IDR_MAINFRAME           "prvi"
    IDR_PRVITYPE     "\nPrvi\nPrvi\n\n\nPrvi.Document\nPrvi Document"
END

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    AFX_IDS_APP_TITLE       "prvi"
    AFX_IDS_IDLEMESSAGE     "For Help, press F1"
    AFX_IDS_HELPMODEMESSAGE "Select an object on which to get Help"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EXT        "EXT"
    ID_INDICATOR_CAPS       "CAP"
    ID_INDICATOR_NUM        "NUM"
    ID_INDICATOR_SCRL       "SCRL"
    ID_INDICATOR_OVR        "OVR"
    ID_INDICATOR_REC        "REC"
    ID_INDICATOR_REF        "REFRESH"
END

STRINGTABLE DISCARDABLE
```

```
BEGIN
    ID_FILE_NEW             "Create a new document\nNew"
    ID_FILE_OPEN            "Open an existing document\nOpen"
    ID_FILE_CLOSE           "Close the active document\nClose"
    ID_FILE_SAVE            "Save the active document\nSave"
    ID_FILE_SAVE_AS         "Save the active document with a new
name\nSave As"
    ID_FILE_PAGE_SETUP      "Change the printing options\nPage Setup"
    ID_FILE_PRINT_SETUP     "Change the printer and printing
options\nPrint Setup"
    ID_FILE_PRINT           "Print the active document\nPrint"
    ID_FILE_PRINT_PREVIEW   "Display full pages\nPrint Preview"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_APP_ABOUT            "Display program information, version
number and copyright\nAbout"
    ID_APP_EXIT             "Quit the application; prompts to save
documents\nExit"
    ID_HELP_INDEX           "Opens Help\nHelp Topics"
    ID_HELP_FINDER          "List Help topics\nHelp Topics"
    ID_HELP_USING           "Display instructions about how to use
help\nHelp"
    ID_CONTEXT_HELP         "Display help for clicked on buttons,
menus and windows\nHelp"
    ID_HELP                 "Display help for current task or
command\nHelp"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_MRU_FILE1       "Open this document"
    ID_FILE_MRU_FILE2       "Open this document"
    ID_FILE_MRU_FILE3       "Open this document"
    ID_FILE_MRU_FILE4       "Open this document"
    ID_FILE_MRU_FILE5       "Open this document"
    ID_FILE_MRU_FILE6       "Open this document"
    ID_FILE_MRU_FILE7       "Open this document"
    ID_FILE_MRU_FILE8       "Open this document"
    ID_FILE_MRU_FILE9       "Open this document"
    ID_FILE_MRU_FILE10      "Open this document"
    ID_FILE_MRU_FILE11      "Open this document"
    ID_FILE_MRU_FILE12      "Open this document"
    ID_FILE_MRU_FILE13      "Open this document"
    ID_FILE_MRU_FILE14      "Open this document"
    ID_FILE_MRU_FILE15      "Open this document"
    ID_FILE_MRU_FILE16      "Open this document"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_NEXT_PANE          "Switch to the next window pane\nNext Pane"
    ID_PREV_PANE            "Switch back to the previous window
pane\nPrevious Pane"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_WINDOW_NEW           "Open another window for the active
document\nNew Window"
```

```
    ID_WINDOW_ARRANGE       "Arrange icons at the bottom of the
window\nArrange Icons"
    ID_WINDOW_CASCADE       "Arrange windows so they overlap\nCascade
Windows"
    ID_WINDOW_TILE_HORZ     "Arrange windows as non-overlapping
tiles\nTile Windows"
    ID_WINDOW_TILE_VERT     "Arrange windows as non-overlapping
tiles\nTile Windows"
    ID_WINDOW_SPLIT         "Split the active window into
panes\nSplit"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_EDIT_CLEAR           "Erase the selection\nErase"
    ID_EDIT_CLEAR_ALL       "Erase everything\nErase All"
    ID_EDIT_COPY            "Copy the selection and put it on the
Clipboard\nCopy"
    ID_EDIT_CUT             "Cut the selection and put it on the
Clipboard\nCut"
    ID_EDIT_FIND            "Find the specified text\nFind"
    ID_EDIT_PASTE           "Insert Clipboard contents\nPaste"
    ID_EDIT_REPEAT          "Repeat the last action\nRepeat"
    ID_EDIT_REPLACE         "Replace specific text with different
text\nReplace"
    ID_EDIT_SELECT_ALL      "Select the entire document\nSelect All"
    ID_EDIT_UNDO            "Undo the last action\nUndo"
    ID_EDIT_REDO            "Redo the previously undone action\nRedo"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_VIEW_TOOLBAR         "Show or hide the toolbar\nToggle
ToolBar"
    ID_VIEW_STATUS_BAR      "Show or hide the status bar\nToggle
StatusBar"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCSIZE          "Change the window size"
    AFX_IDS_SCMOVE          "Change the window position"
    AFX_IDS_SCMINIMIZE      "Reduce the window to an icon"
    AFX_IDS_SCMAXIMIZE      "Enlarge the window to full size"
    AFX_IDS_SCNEXTWINDOW    "Switch to the next document window"
    AFX_IDS_SCPREVWINDOW    "Switch to the previous document window"
    AFX_IDS_SCCLOSE         "Close the active window and prompts to
save the documents"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCRESTORE       "Restore the window to normal size"
    AFX_IDS_SCTASKLIST      "Activate Task List"
    AFX_IDS_MDICHILD        "Activate this window"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_PREVIEW_CLOSE  "Close print preview mode\nCancel Preview"
END
```

```
STRINGTABLE DISCARDABLE
BEGIN
    ID_MOJESTAVKE_KREIRAJPROZOR "Kreira prozor"
    ID_MOJE_PROBA           "Proba"
    ID_MOJE_ARITM           "Aritmeticke operacije"
    ID_MOJE_NEMOD           "Nemodalni dijalog"
    ID_VIEW_MYTOOLS         "Shows or hides my tools\nShows or hides
my tools"
    ID_FILE_PROPERTIES      "Properties"
END
#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////
// English (U.K.) resources
#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENG)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_UK
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//
1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
    "#define _AFX_NO_OLE_RESOURCES\r\n"
    "#define _AFX_NO_TRACKER_RESOURCES\r\n"
    "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
    "\r\n"
    "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
    "#ifdef _WIN32\r\n"
    "LANGUAGE 9, 1\r\n"
    "#pragma code_page(1252)\r\n"
    "#endif\r\n"
    "#include ""res\\prvi.rc2""  // non-Microsoft Visual C++ edited
resources\r\n"
    "#include ""afxres.rc""         // Standard components\r\n"
    "#include ""afxprint.rc""       // printing/print preview
resources\r\n"
    "#endif\0"
END
#endif    // APSTUDIO_INVOKED

/////////////////////////////////////////////////////////////////////////
//
```

```
// Bitmap
//
IDR_MYTOOLS             BITMAP  DISCARDABLE     "res\\mytools.bmp"

/////////////////////////////////////////////////////////////////////////
//
// Toolbar
//
IDR_MYTOOLS TOOLBAR DISCARDABLE  16, 15
BEGIN
    BUTTON      ID_MYTOOL_REFRESH
    BUTTON      ID_MYTOOL_ACTION
END

/////////////////////////////////////////////////////////////////////////
//
// Dialog
//
IDD_DLGARITM DIALOG DISCARDABLE  0, 0, 187, 118
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dijalog za aritmetiku"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,130,7,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,130,24,50,14
    COMBOBOX        IDC_OPERACIJA,15,16,48,72,CBS_DROPDOWNLIST |
                    WS_VSCROLL | WS_TABSTOP
    LTEXT           "Operacija",IDC_STATIC,77,19,31,8
    EDITTEXT        IDC_PRVI,15,39,40,14,ES_AUTOHSCROLL
    EDITTEXT        IDC_DRUGI,15,62,40,14,ES_AUTOHSCROLL
    EDITTEXT        IDC_REZ,15,84,40,14,ES_AUTOHSCROLL | ES_READONLY
                    | NOT WS_BORDER
    LTEXT           "Prvi",IDC_STATIC,77,42,13,8
    LTEXT           "Drugi",IDC_STATIC,77,66,18,8
    LTEXT           "Rezultat",IDC_STATIC,77,87,27,8
    PUSHBUTTON      "Izracunaj",IDC_IZRACUNAJ,121,84,50,14
    COMBOBOX        IDC_GDEJEKOD,132,45,48,40,CBS_DROPDOWNLIST |
                    WS_VSCROLL | WS_TABSTOP
END

IDD_DLGNEMOD DIALOG DISCARDABLE  0, 0, 285, 203
STYLE DS_MODALFRAME | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Nemodalni dijalog"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,228,7,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,228,24,50,14
    PUSHBUTTON      "Akcija",IDC_AKCIJA,228,47,50,14
END

/////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//
#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_DLGARITM, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 180
```

```
        TOPMARGIN, 7
        BOTTOMMARGIN, 111
    END

    IDD_DLGNEMOD, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 278
        TOPMARGIN, 7
        BOTTOMMARGIN, 196
    END
END
#endif    // APSTUDIO_INVOKED

////////////////////////////////////////////////////////////////////////
//
// Dialog Info
//
IDD_DLGARITM DLGINIT
BEGIN
    IDC_OPERACIJA, 0x403, 10, 0
0x6153, 0x6962, 0x6172, 0x6a6e, 0x0065,
    IDC_OPERACIJA, 0x403, 11, 0
0x644f, 0x7a75, 0x6d69, 0x6e61, 0x656a, "\000"
    IDC_OPERACIJA, 0x403, 9, 0
0x6e4d, 0x7a6f, 0x6e65, 0x656a, "\000"
    IDC_OPERACIJA, 0x403, 9, 0
0x6544, 0x6a6c, 0x6e65, 0x656a, "\000"
    IDC_GDEJEKOD, 0x403, 4, 0
0x5845, 0x0045,
    IDC_GDEJEKOD, 0x403, 9, 0
0x4c44, 0x204c, 0x6d69, 0x6c70, "\000"
    IDC_GDEJEKOD, 0x403, 9, 0
0x4c44, 0x204c, 0x7865, 0x6c70, "\000"
    0
END

////////////////////////////////////////////////////////////////////////
//
// String Table
//
STRINGTABLE DISCARDABLE
BEGIN
    ID_MYTOOL_REFRESH       "Refresh\nRefresh"
    ID_MYTOOL_ACTION        "Action\nAction"
END

#endif    // English (U.K.) resources
////////////////////////////////////////////////////////////////////////

#ifndef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
```

```
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif
#include "res\prvi.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#include "afxprint.rc"       // printing/print preview resources
#endif
/////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

- **B2-4: StdAfx.h**

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//
#if !defined
(AFX_STDAFX_H__92680FA9_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
#define AFX_STDAFX_H__92680FA9_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
#define VC_EXTRALEAN //Exclude rarely-used stuff from Windows headers
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC OLE automation classes
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>     // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
// immediately before the previous line.
#endif
// !defined
// (AFX_STDAFX_H__92680FA9_9AAD_11D4_B72F_8C97951B8C97__INCLUDED_)
```

- **B2-4: StdAfx.cpp**

```
// stdafx.cpp : source file that includes just the standard includes
//    prvi.pch will be the pre-compiled header
//    stdafx.obj will contain the pre-compiled type information
#include "stdafx.h"
```

B3. Grupa fajlova Dijalozi:

- **B3-1: DlgAritmetika.h**

```
#if !defined
(AFX_DLGARITMETIKA_H__36F0D883_A018_11D4_B72F_8C97951B8C97__INCLUDED_
)
#define
AFX_DLGARITMETIKA_H__36F0D883_A018_11D4_B72F_8C97951B8C97__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
```

```
// DlgAritmetika.h : header file
//
typedef double (*pfnAritmetika)(double,double,long);

/////////////////////////////////////////////////////////////////////
// CDlgAritmetika dialog
class CDlgAritmetika : public CDialog
{   // Construction
    public:
        CDlgAritmetika(CWnd* pParent = NULL);  //standard constructor
// Dialog Data
    //{{AFX_DATA(CDlgAritmetika)
        enum { IDD = IDD_DLGARITM };
        int       m_operacija;
        double    m_prvi;
        double    m_drugi;
        double    m_rez;
        int       m_gdejekod;
    //}}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDlgAritmetika)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
    protected:
    // Generated message map functions
    //{{AFX_MSG(CDlgAritmetika)
        afx_msg void OnIzracunaj();
        virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
// immediately before the previous line.
#endif // !defined
(AFX_DLGARITMETIKA_H__36F0D883_A018_11D4_B72F_8C97951B8C97__INCLUDED_
)
```

- **B3-1: DlgAritmetika.cpp**

```
// DlgAritmetika.cpp : implementation file
//
#include "stdafx.h"
#include "prvi.h"
#include "DlgAritmetika.h"
#include "MojDLL.H"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////
// CDlgAritmetika dialog
CDlgAritmetika::CDlgAritmetika(CWnd* pParent /*=NULL*/)
```

```
      : CDialog(CDlgAritmetika::IDD, pParent)
{   //{{AFX_DATA_INIT(CDlgAritmetika)
    m_operacija = -1;
    m_prvi = 0.0;
    m_drugi = 0.0;
    m_rez = 0.0;
    m_gdejekod = -1;
    //}}AFX_DATA_INIT
}

void CDlgAritmetika::DoDataExchange(CDataExchange* pDX)
{   CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDlgAritmetika)
    DDX_CBIndex(pDX, IDC_OPERACIJA, m_operacija);
    DDX_Text(pDX, IDC_PRVI, m_prvi);
    DDX_Text(pDX, IDC_DRUGI, m_drugi);
    DDX_Text(pDX, IDC_REZ, m_rez);
    DDX_CBIndex(pDX, IDC_GDEJEKOD, m_gdejekod);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDlgAritmetika, CDialog)
    //{{AFX_MSG_MAP(CDlgAritmetika)
    ON_BN_CLICKED(IDC_IZRACUNAJ, OnIzracunaj)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDlgAritmetika message handlers
void CDlgAritmetika::OnIzracunaj()
{   // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    switch(m_gdejekod)
    {   case 0:
            switch(m_operacija)
            {   case 0:
                    m_rez=m_prvi+m_drugi;
                    break;
                case 1:
                    m_rez=m_prvi-m_drugi;
                    break;
                case 2:
                    m_rez=m_prvi*m_drugi;
                    break;
                case 3:
                    if(m_drugi==0)
                    {   AfxMessageBox("Deljenje sa nulom!");
                        m_rez=0;
                    }
                    else m_rez=m_prvi/m_drugi;
                    break;
            }
            break;
        case 1:
            m_rez=Aritmetika(m_prvi,m_drugi,(long)m_operacija);
            AfxMessageBox("Implicitno linkovanje DLLa");
            break;
        case 2:
            pfnAritmetika fn;
            HINSTANCE hDLL;
            if((hDLL=GetModuleHandle("MojDLL1.dll"))==NULL)
```

```
                AfxMessageBox("DLL nije nadjen");
            fn=(pfnAritmetika)GetProcAddress(hDLL,"Aritmetika");
            m_rez=fn(m_prvi,m_drugi,(long)m_operacija);
            AfxMessageBox("Eksplicitno linkovanje DLLa");
            break;
    }
    UpdateData(FALSE);
}

BOOL CDlgAritmetika::OnInitDialog()
{   CDialog::OnInitDialog();
    // TODO: Add extra initialization here
    m_operacija=0;
    m_prvi=0;
    m_drugi=0;
    m_rez=0;
    m_gdejekod=0;
    UpdateData(FALSE);
    return TRUE;//return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}
```

- **B3-2: DlgNemodalni.h**

```
#if !defined
(AFX_DLGNEMODALNI_H__36F0D884_A018_11D4_B72F_8C97951B8C97__INCLUDED_)
#define
AFX_DLGNEMODALNI_H__36F0D884_A018_11D4_B72F_8C97951B8C97__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// DlgNemodalni.h : header file
//

/////////////////////////////////////////////////////////////////////////
// CDlgNemodalni dialog
class CDlgNemodalni : public CDialog
{   // Construction
    public:
        HMETAFILE hmf;
        CDlgNemodalni(CWnd* pParent = NULL); //standard constructor
// Dialog Data
    //{{AFX_DATA(CDlgNemodalni)
        enum { IDD = IDD_DLGNEMOD };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDlgNemodalni)
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        // DDX/DDV support
        virtual void PostNcDestroy();
    //}}AFX_VIRTUAL
// Implementation
    protected:
    // Generated message map functions
    //{{AFX_MSG(CDlgNemodalni)
        afx_msg void OnAkcija();
        virtual void OnCancel();
```

```
        virtual void OnOK();
        virtual BOOL OnInitDialog();
        afx_msg void OnPaint();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
// immediately before the previous line.

#endif
// !defined
(AFX_DLGNEMODALNI_H__36F0D884_A018_11D4_B72F_8C97951B8C97__INCLUDED_)
```

- **B3-2: DlgNemodalni.cpp**

```
// DlgNemodalni.cpp : implementation file
//
#include "stdafx.h"
#include "prvi.h"
#include "DlgNemodalni.h"
#include "math.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////
// CDlgNemodalni dialog
CDlgNemodalni::CDlgNemodalni(CWnd* pParent /*=NULL*/)
      : CDialog(CDlgNemodalni::IDD, pParent)
{   //{{AFX_DATA_INIT(CDlgNemodalni)
    // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
    Create(IDD_DLGNEMOD);
}

void CDlgNemodalni::DoDataExchange(CDataExchange* pDX)
{   CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDlgNemodalni)
    // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDlgNemodalni, CDialog)
    //{{AFX_MSG_MAP(CDlgNemodalni)
    ON_BN_CLICKED(IDC_AKCIJA, OnAkcija)
    ON_WM_PAINT()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////
// CDlgNemodalni message handlers
void CDlgNemodalni::OnAkcija()
{   // TODO: Add your control notification handler code here
    // AfxMessageBox("Neka akcija");
    ::CopyMetaFile(hmf,"crtez.wmf");
}
```

```
void CDlgNemodalni::PostNcDestroy()
{   // TODO:Add your specialized code here and/or call the base class
    ::DeleteMetaFile(hmf);
    delete this;
    //CDialog::PostNcDestroy();
}

void CDlgNemodalni::OnCancel()
{   // TODO: Add extra cleanup here
    DestroyWindow();
    //CDialog::OnCancel();
}

void CDlgNemodalni::OnOK()
{   // TODO: Add extra validation here
    DestroyWindow();
    //CDialog::OnOK();
}

BOOL CDlgNemodalni::OnInitDialog()
{   CDialog::OnInitDialog();
    // TODO: Add extra initialization here
    CMetaFileDC dc;
    dc.Create();
    int ksx=50,ksy=200; //centar koordinatnog sistema
    int yr=100,xr=50; //rezolucija - broj piksela po jedinici
    double ymax=1,xmax=5; //maks. vrednosti na grafiku
    int ksymax=(int)(ymax*yr),
    ksxmax=(int)(xmax*xr); //visina i sirina koordinatnog sistema
    double f[250],g[250];
    //
    // Generisanje funkcije i envelope
    //
    int i;
    double x,dx;
    for(i=0,x=0,dx=xmax/(ksxmax-1);i<ksxmax;i++,x+=dx)
    {   g[i]=exp(-0.2*x);
        f[i]=sin(5*x)*g[i];
    }
    //
    // Kreiranje "olovaka" - pen
    CPen penOld; //default pen
    CPen penKS; //pen za koordinatni sistem
    CPen penG; // pen za grafik
    CPen penE; //pen za envelopu
    penKS.CreatePen(PS_DASH,1,RGB(255,255,0));
    penG.CreatePen(PS_SOLID,2,RGB(0,0,255));
    penE.CreatePen(PS_SOLID,1,RGB(255,0,0));
    // Pamtimo default pen da bi mogli kasnije da ga vratimo
    CPen* ppenOld=dc.SelectObject(&penOld);
    //
    // Crtanje koordinatnog sistema
    //
    dc.SelectObject(&penKS);
    dc.MoveTo(ksx,ksy-ksymax);
    dc.LineTo(ksx,ksy+ksymax);
    dc.MoveTo(ksx,ksy);
    dc.LineTo(ksx+ksxmax,ksy);
    //
    // Crtanje funkcije
```

```
    //
    dc.SelectObject(&penG);
    dc.MoveTo(ksx,ksy-(int)(f[0]*yr));
    for(i=1,x=dx;i<ksxmax;i++,x+=dx)
        dc.LineTo(ksx+i,ksy-(int)(f[i]*yr));
    //
    // Crtanje envelope
    //
    dc.SelectObject(&penE);
    dc.MoveTo(ksx,ksy-(int)(g[0]*yr));
    for(i=1,x=dx;i<ksxmax;i++,x+=dx)
        dc.LineTo(ksx+i,ksy-(int)(g[i]*yr));
    dc.MoveTo(ksx,ksy+(int)(g[0]*yr));
    for(i=1,x=dx;i<ksxmax;i++,x+=dx)
        dc.LineTo(ksx+i,ksy+(int)(g[i]*yr));
    // Vracamo stari, dafault pen
    dc.SelectObject(ppenOld);
    hmf=dc.Close();
    return TRUE; //return TRUE unless you set the focus to a control
                 //EXCEPTION: OCX Property Pages should return FALSE
}

void CDlgNemodalni::OnPaint()
{   CPaintDC dc(this); // device context for painting
    // TODO: Add your message handler code here
    int ksx=50,ksy=200; //centar koordinatnog sistema
    int yr=100,xr=50; //rezolucija - broj piksela po jedinici
    double ymax=1,xmax=5; //maks. vrednosti na grafiku
    int ksymax=(int)(ymax*yr),
    ksxmax=(int)(xmax*xr); //visina i sirina koordinatnog sistema
    int oldBkMode;
    // Promena rezima pozadine u prozirno
    oldBkMode=dc.SetBkMode(TRANSPARENT); //necemo OPAQUE
    //
    // Iscrtavanje snimljenog metafajla
    //
    dc.PlayMetaFile(hmf);
    //
    // Ispis teksta
    //
    CString s1="Grafik funkcije";
    CString s2="y=exp(-0.2*x)*sin(5*x)";
    CFont fontNew1;
    CFont *pfontOld;
    // Biranje predefinisanog fonta
    fontNew1.CreateStockObject(ANSI_FIXED_FONT);
    //
    // Tekst iznad grafika
    //
    // Biramo sistemski font jednake sirine, i pamtimo default font
    pfontOld=dc.SelectObject(&fontNew1);
    // Velicina polja za tekst se proracunava tek nakon
    // definisanja fonta
    CSize ss1=dc.GetTextExtent(s1);
    dc.SetTextColor(RGB(0,255,255)); // Boja teksta - cijan
    dc.TextOut(ksx+ksxmax/2-ss1.cx/2,ksy-ksymax-ss1.cy,s1);//Ispis
    //
    // Tekst ispod grafika
    //
    dc.SetTextColor(RGB(255,0,255)); // Boja - ljubicasta
    dc.SelectObject(pfontOld); // Vracamo default font
```

```
    ss1=dc.GetTextExtent(s1); // Proracun dimenzija polja za tekst
    dc.TextOut(ksx+ksxmax/2-ss1.cx/2,ksy+ksymax+ss1.cy,s1); // Ispis
    // Kreiranje novog fonta
    CFont *pfontNew2;
    LOGFONT lf; // Struktura logickog fonta koju treba popuniti
    memset(&lf,0,sizeof(LOGFONT)); // Inicijalizacija
    lstrcpy(lf.lfFaceName,"Times New Roman"); // Ime fonta
    // Proracun visine fonta
    lf.lfHeight=(-1)*MulDiv(10,dc.GetDeviceCaps(LOGPIXELSY),72);
    lf.lfItalic=TRUE; // Osobina fonta - "italic"
    pfontNew2=new CFont(); // Kreiranje objekta fonta
    // Inicijalizacija LOGFONT strukturom
    pfontNew2->CreateFontIndirect(&lf);
    dc.SelectObject(pfontNew2); // Izabiranje kreiranog fonta
    // Proracun dimenzija polja za tekst
    CSize ss2=dc.GetTextExtent(s2);
    // Ispis
    dc.TextOut(ksx+ksxmax/2-ss2.cx/2,ksy+ksymax+ss1.cy+ss2.cy,s2);
    dc.SelectObject(pfontOld); // Vracanje starog, default fonta
    delete pfontNew2; // Brisanje objekta fonta
    dc.SetBkMode(oldBkMode); // Vracanje na stari rezim pozadine
    // Do not call CDialog::OnPaint() for painting messages
}
```

B4. Grupa fajlova Property Pages:

- **B4-1: PSProperties.h**

```
#if !defined
(AFX_PSPROPERTIES_H__47441665_A415_11D4_B72F_8C97951B8C97__INCLUDED_)
#define
AFX_PSPROPERTIES_H__47441665_A415_11D4_B72F_8C97951B8C97__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PSProperties.h : header file
//

#include "PPPrva.h"
#include "PPDruga.h"

/////////////////////////////////////////////////////////////////////
// CPSProperties
class CPSProperties : public CPropertySheet
{   DECLARE_DYNAMIC(CPSProperties)
// Construction
    public:
        CPSProperties(UINT nIDCaption, CWnd* pParentWnd = NULL, UINT
            iSelectPage = 0);
        CPSProperties(LPCTSTR pszCaption, CWnd* pParentWnd = NULL,
            UINT iSelectPage = 0);
// Attributes
    public:
        CPPPrva *m_pp1;
        CPPDruga *m_pp2;
// Operations
    public:
// Overrides
    // ClassWizard generated virtual function overrides
```

```
    //{{AFX_VIRTUAL(CPSProperties)
    protected:
        virtual void PostNcDestroy();
    //}}AFX_VIRTUAL
// Implementation
    public:
        virtual ~CPSProperties();
// Generated message map functions
    protected:
    //{{AFX_MSG(CPSProperties)
    // NOTE-the ClassWizard will add/remove member functions here.
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
// immediately before the previous line.
#endif
// !defined
(AFX_PSPROPERTIES_H__47441665_A415_11D4_B72F_8C97951B8C97__INCLUDED_)
```

- **B4-1: PSProperties.cpp**

```
// PSProperties.cpp : implementation file
//
#include "stdafx.h"
#include "prvi.h"
#include "PSProperties.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////
// CPSProperties
IMPLEMENT_DYNAMIC(CPSProperties, CPropertySheet)
CPSProperties::CPSProperties(UINT nIDCaption, CWnd* pParentWnd,
    UINT iSelectPage)
    :CPropertySheet(nIDCaption, pParentWnd, iSelectPage)
{
}

CPSProperties::CPSProperties(LPCTSTR pszCaption, CWnd* pParentWnd,
     UINT iSelectPage)
    :CPropertySheet(pszCaption, pParentWnd, iSelectPage)
{   //konstruktor
    m_pp1=new CPPPrva;
    m_pp2=new CPPDruga;
    AddPage(m_pp1);
    AddPage(m_pp2);
    //CMainFrame *pMF=(CMainFrame *)AfxGetApp()->m_pMainWnd;
    Create(AfxGetApp()->m_pMainWnd,WS_SYSMENU|WS_POPUP
        |WS_CAPTION|WS_VISIBLE|DS_MODALFRAME);
}

CPSProperties::~CPSProperties()
{
```

```
}

BEGIN_MESSAGE_MAP(CPSProperties, CPropertySheet)
    //{{AFX_MSG_MAP(CPSProperties)
    // NOTE-the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////
// CPSProperties message handlers
void CPSProperties::PostNcDestroy()
{   // TODO:Add your specialized code here and/or call the base class
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    pApp->m_prop_en=FALSE;
    delete m_pp1;
    delete m_pp2;
    delete this;
    //CPropertySheet::PostNcDestroy();
}
```

- **B4-2: PPPrva.h**

```
#if !defined
(AFX_PPPRVA_H__47441661_A415_11D4_B72F_8C97951B8C97__INCLUDED_)
#define AFX_PPPRVA_H__47441661_A415_11D4_B72F_8C97951B8C97__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PPPrva.h : header file
//

/////////////////////////////////////////////////////////////////////////
// CPPPrva dialog
class CPPPrva : public CPropertyPage
{   DECLARE_DYNCREATE(CPPPrva)
// Construction
    public:
        CPPPrva();
        ~CPPPrva();
// Dialog Data
    //{{AFX_DATA(CPPPrva)
        enum { IDD = IDD_PP_1 };
        int       m_par1;
    //}}AFX_DATA
// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPPPrva)
    public:
        virtual BOOL OnSetActive();
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);
            // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
    protected:
    // Generated message map functions
    //{{AFX_MSG(CPPPrva)
        afx_msg void OnUpdate();
        virtual BOOL OnInitDialog();
    //}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
// immediately before the previous line.
#endif
// !defined
// (AFX_PPPRVA_H__47441661_A415_11D4_B72F_8C97951B8C97__INCLUDED_)
```

- **B4-2: PPPrva.cpp**

```
// PPPrva.cpp : implementation file
//
#include "stdafx.h"
#include "prvi.h"
#include "PPPrva.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////
// CPPPrva property page
IMPLEMENT_DYNCREATE(CPPPrva, CPropertyPage)
CPPPrva::CPPPrva() : CPropertyPage(CPPPrva::IDD)
{   //{{AFX_DATA_INIT(CPPPrva)
    m_par1 = 0;
    //}}AFX_DATA_INIT
}

CPPPrva::~CPPPrva()
{
}

void CPPPrva::DoDataExchange(CDataExchange* pDX)
{   CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPPPrva)
    DDX_Text(pDX, IDC_PAR1, m_par1);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPPPrva, CPropertyPage)
    //{{AFX_MSG_MAP(CPPPrva)
        ON_BN_CLICKED(IDC_UPDATE, OnUpdate)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////
// CPPPrva message handlers
void CPPPrva::OnUpdate()
{   // TODO: Add your control notification handler code here
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    UpdateData(TRUE);
    pApp->m_par1 = m_par1;
}

BOOL CPPPrva::OnInitDialog()
{   CPropertyPage::OnInitDialog();
```

```
    // TODO: Add extra initialization here
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    m_par1 = pApp->m_par1;
    UpdateData(FALSE);
    return TRUE; //return TRUE unless you set the focus to a control
                 //EXCEPTION: OCX Property Pages should return FALSE
}

BOOL CPPPrva::OnSetActive()
{   // TODO:Add your specialized code here and/or call the base class
    //UpdateData(FALSE);
    return CPropertyPage::OnSetActive();
}
```

- **B4-3: PPDruga.h**

```
#if !defined
(AFX_PPDRUGA_H__47441662_A415_11D4_B72F_8C97951B8C97__INCLUDED_)
#define
AFX_PPDRUGA_H__47441662_A415_11D4_B72F_8C97951B8C97__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PPDruga.h : header file
//

/////////////////////////////////////////////////////////////////////////
// CPPDruga dialog
class CPPDruga : public CPropertyPage
{   DECLARE_DYNCREATE(CPPDruga)
// Construction
    public:
        CPPDruga();
        ~CPPDruga();
// Dialog Data
    //{{AFX_DATA(CPPDruga)
        enum { IDD = IDD_PP_2 };
        int       m_par2;
    //}}AFX_DATA
// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CPPDruga)
    public:
        virtual BOOL OnSetActive();
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);
            // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
    protected:
// Generated message map functions
    //{{AFX_MSG(CPPDruga)
        afx_msg void OnUpdate();
        virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
```

```
// immediately before the previous line.
#endif
// !defined
// (AFX_PPDRUGA_H__47441662_A415_11D4_B72F_8C97951B8C97__INCLUDED_)
```

- **B4-3: PPDruga.cpp**

```
// PPDruga.cpp : implementation file
//
#include "stdafx.h"
#include "prvi.h"
#include "PPDruga.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////
// CPPDruga property page
IMPLEMENT_DYNCREATE(CPPDruga, CPropertyPage)
CPPDruga::CPPDruga() : CPropertyPage(CPPDruga::IDD)
{   //{{AFX_DATA_INIT(CPPDruga)
    m_par2 = 0;
    //}}AFX_DATA_INIT
}

CPPDruga::~CPPDruga()
{
}

void CPPDruga::DoDataExchange(CDataExchange* pDX)
{   CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPPDruga)
    DDX_Text(pDX, IDC_PAR2, m_par2);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPPDruga, CPropertyPage)
    //{{AFX_MSG_MAP(CPPDruga)
    ON_BN_CLICKED(IDC_UPDATE, OnUpdate)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////
// CPPDruga message handlers
void CPPDruga::OnUpdate()
{   // TODO: Add your control notification handler code here
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    UpdateData(TRUE);
    pApp->m_par2 = m_par2;
}

BOOL CPPDruga::OnInitDialog()
{   CPropertyPage::OnInitDialog();
    // TODO: Add extra initialization here
    CPrviApp *pApp=(CPrviApp *)AfxGetApp();
    m_par2 = pApp->m_par2;
    UpdateData(FALSE);
    return TRUE; //return TRUE unless you set the focus to a control
```

```
                //EXCEPTION: OCX Property Pages should return FALSE
}

BOOL CPPDruga::OnSetActive()
{    //TODO:Add your specialized code here and/or call the base class
    //UpdateData(FALSE);
    return CPropertyPage::OnSetActive();
}
```

Prilog C. Dinamička biblioteka MojDLL

U ovom prilogu je dat kompletan C izvorni kod za generisanje dinamičke biblioteke MojDLL.dll koju, u lekciji 7, koristi glavni program Prvi.exe. Zaglavlje MojDLL.h se ne koristi prilikom prevođenja dinamičke biblioteke, već se uključuje u prevođenje glavne aplikacije. Za generisanje dll-a potrebno je kompajirati MojDLL.c.

- **Zaglavlje MojDLL.h**

```
/* MojDLL.h */
extern "C" double Aritmetika(double operand1, double operand2
    ,long operacija);
```

- **Izvorni fajl MojDLL.c**

```
/* MojDLL.c */
__declspec(dllexport) double Aritmetika(double operand1,
    double operand2, long operacija)
{   double rezultat;
    switch(operacija)
    {   case 0:
            rezultat = operand1 + operand2;
            break;
        case 1:
            rezultat = operand1 - operand2;
            break;
        case 2:
            rezultat = operand1 * operand2;
            break;
        case 3:
            if(operand2==0) rezultat = 0;
            else rezultat = operand1 / operand2;
            break;
        default: rezultat = 0;
    }
    return rezultat;
}
```

Prilog D. Adresni prostor procesa

U ovom prilogu prikazan je izgled adresnog prostora jednog stvarnog procesa. Program VMMAP kojim je ovo generisano napisao je autor knjige [2].

Adresa	*Status*	*Veličina*	*Atributi*		*Fajl*
BFFFF000	Free	4096			
BFF70000	Private	585728	5	----	C:\WINDOWS\SYSTEM\KERNEL32.DLL
BFF61000	Free	61440			
BFF50000	Private	69632	3	----	C:\WINDOWS\SYSTEM\USER32.DLL
BFF46000	Free	40960			
BFF20000	Private	155648	5	----	C:\WINDOWS\SYSTEM\GDI32.DLL
BFF1F000	Free	4096			
BFEB0000	Private	454656	3	----	
BFEA0000	Private	65536	3	----	C:\WINDOWS\SYSTEM\ADVAPI32.DLL
BFE96000	Free	40960			
BFE90000	Private	24576	3	----	
BFE86000	Free	40960			
BFE80000	Private	24576	3	----	
BFE20000	Free	393216			
BFE10000	Private	65536	3	----	
BFB73000	Free	2740224			
BFB50000	Private	143360	7	----	
835A0000	Free	1012596736			
834A0000	Private	1048576	3	----	
8348F000	Private	69632	1	----	
83470000	Free	126976			
83461000	Private	61440	1	-RW-	
8340C000	Private	348160	1	-RW-	
833ED000	Private	126976	1	----	
83398000	Private	348160	1	-RW-	
83368000	Private	196608	1	----	
83329000	Private	258048	1	-RW-	
832EA000	Private	258048	1	-RW-	
832D5000	Private	86016	1	-RW-	
83295000	Private	262144	1	-RW-	
8326A000	Private	176128	1	-RW-	
8325D000	Private	53248	1	-RW-	
83208000	Private	348160	1	-RW-	
831D9000	Private	192512	1	----	
831B3000	Private	155648	1	----	
83184000	Private	192512	1	----	
83153000	Private	200704	1	----	
830A3000	Free	720896			
830A2000	Private	4096	1	----	
83072000	Free	196608			
83053000	Private	126976	3	----	
83034000	Private	126976	3	----	
83015000	Private	126976	3	----	
82E01000	Private	2179072	3	----	
82DD0000	Private	200704	1	----	
82DA0000	Private	196608	1	----	
82D91000	Private	61440	1	-RW-	
82D83000	Private	57344	1	----	
82D64000	Private	126976	3	----	
82CEA000	Private	499712	1	----	
82CCB000	Private	126976	3	----	

```
82CB7000 Private      81920 1  ----
82C98000 Private     126976 1  ----
82C7D000 Private     110592 1  ----
82C62000 Private     110592 1  ----
82C36000 Private     180224 1  ----
82B34000 Private    1056768 3  ----
82B2F000 Private      20480 1  ----
82B2E000 Private       4096 1  ----
82B2A000 Free         16384
82B29000 Private       4096 1  ----
82B28000 Private       4096 1  ----
82B27000 Private       4096 1  ----
82B25000 Private       8192 1  ----
82B22000 Private      12288 1  ----
82AFD000 Private     151552 1  ----
82AFC000 Private       4096 1  ----
82AFB000 Private       4096 1  ----
82AFA000 Private       4096 1  ----
82AF8000 Private       8192 1  ----
82AF7000 Private       4096 1  ----
82AF5000 Private       8192 1  ----
82AF3000 Private       8192 1  ----
82AF2000 Private       4096 1  ----
82AF1000 Private       4096 1  ----
82AED000 Private      16384 1  ----
82AE9000 Private      16384 1  ----
82AE1000 Private      32768 1  ----
82AE0000 Private       4096 1  ----
82ADF000 Private       4096 1  ----
82ADE000 Private       4096 1  ----
82ADC000 Private       8192 1  ----
82ADB000 Private       4096 1  ----
82AD9000 Private       8192 1  ----
82AD7000 Private       8192 1  ----
82AD6000 Private       4096 1  ----
82AD5000 Private       4096 1  ----
82AD4000 Private       4096 1  ----
82AD3000 Private       4096 1  ----
82AD1000 Private       8192 1  ----
82AD0000 Private       4096 1  ----
82ACF000 Private       4096 1  ----
82ACD000 Private       8192 1  ----
82ACB000 Private       8192 1  ----
829CA000 Private    1052672 3  ----
829C9000 Private       4096 1  ----
829C8000 Private       4096 1  ----
829C7000 Private       4096 1  ----
829C6000 Free          4096
829C0000 Private      24576 1  -RW-
829BA000 Private      24576 1  -RW-
829AA000 Private      65536 1  -RW-
8299A000 Private      65536 1  -RW-
8289A000 Private    1048576 3  ----
82864000 Private     221184 1  -RW-
82763000 Private    1052672 3  ----
8268B000 Private     884736 1  -RW-
8258B000 Private    1048576 3  ----
8258A000 Private       4096 1  ----
82589000 Private       4096 1  ----
82588000 Private       4096 1  ----
8255B000 Private     184320 1  ----
```

```
8245A000 Private    1052672 3  ----
82459000 Private       4096 1  ----
82458000 Private       4096 1  ----
82258000 Private    2097152 3  ----
82257000 Private       4096 1  ----
82256000 Private       4096 1  ----
8224E000 Private      32768 1  -RW-
82246000 Private      32768 1  -RW-
8223F000 Private      28672 1  -RW-
82238000 Private      28672 1  -RW-
82237000 Private       4096 1  ----
82236000 Private       4096 1  ----
82235000 Private       4096 1  ----
82234000 Private       4096 1  ----
82233000 Private       4096 1  ----
82232000 Private       4096 1  ----
82231000 Private       4096 1  ----
8222F000 Private       8192 1  ----
8222D000 Private       8192 1  ----
8222C000 Private       4096 1  ----
8222B000 Private       4096 1  ----
8222A000 Private       4096 1  ----
82229000 Private       4096 1  ----
82228000 Private       4096 1  ----
82227000 Private       4096 1  ----
82226000 Private       4096 1  ----
82225000 Private       4096 1  ----
82224000 Private       4096 1  ----
82223000 Private       4096 1  ----
82222000 Private       4096 1  ----
82221000 Private       4096 1  ----
8221C000 Private      20480 1  ----
8221B000 Private       4096 1  ----
8200B000 Private    2162688 5  -RW-
81DEB000 Private    2228224 5  -RW-
81D6B000 Private     524288 3  ----
81D6A000 Private       4096 1  ----
81D69000 Private       4096 1  ----
81D68000 Private       4096 1  ----
81D67000 Private       4096 1  ----
81B47000 Private    2228224 5  -RW-
81AC7000 Private     524288 3  ----
81AC6000 Private       4096 1  ----
81AC5000 Private       4096 1  ----
81ABC000 Private      36864 1  ----
81A91000 Private     176128 1  ----
81A8E000 Private      12288 1  ----
81A8B000 Private      12288 1  ----
81A87000 Private      16384 3  ----
81A83000 Private      16384 2  -RW-
81683000 Private    4194304 3  ----
81583000 Private    1048576 3  ----
81561000 Private     139264 3  ----
80161000 Private   20971520 45 ----
80131000 Private     196608 1  ----
800AA000 Private     552960 1  ----
800A9000 Private       4096 1  ----
80019000 Private     589824 1  ----
80018000 Private       4096 1  ----
80017000 Private       4096 1  ----
80016000 Private       4096 1  ----
```

```
80015000 Private       4096 1  ----
80014000 Private       4096 1  ----
80004000 Private      65536 2  ----
80003000 Private       4096 1  ----
80002000 Private       4096 1  ----
80001000 Private       4096 1  ----
80000000 Private       4096 1  ----
7D670000 Free      43581440
7D660000 Private      65536 4  ---- C:\WINDOWS\SYSTEM\INDICDLL.DLL
78040000 Free      90308608
78000000 Private     262144 3  ---- C:\WINDOWS\SYSTEM\MSVCRT.DLL
00870000 Free    2004418560
00770000 Private    1048576 2  -RW-
00660000 Private    1114112 4  ----
00540000 Private    1179648 6  ---- Thread Stack
00530000 Private      65536 2  -RW-
00420000 Private    1114112 4  ----
00400000 Private     131072 4  ---- C:\WINDOWS\TEMP\VMMAP.EXE
00000000 Free       4194304
```

Prilog E. Import i eksport sekcija DLL-a

U ovom prilogu dat je izgled import i eksport sekcije DLL-a, kao i izgled import sekcije aplikacije (EXE). Izlaz je generisan programom DUMPBIN, koji se isporučuje uz Visual C++. To je konzolni program, a za njegovo uspešno startovanje treba pozvati batch proceduru VCVARS32.BAT koja postavlja promenljive VC okruženja kada se radi u komandnoj liniji. Import sekcija se dobija komandom

```
dumpbin -imports imefajla.ekstenzija >redir_fajl.txt
```

a eksport sekcija komandom

```
dumpbin -exports imefajla.ekstenzija >redir_fajl.txt
```

- **MojDLL.dll - import sekcija**

```
Microsoft (R) COFF Binary File Dumper Version 5.00.7022
Copyright (C) Microsoft Corp 1992-1997. All rights reserved.


Dump of file mojdll.dll

File Type: DLL

         Section contains the following Imports

            KERNEL32.dll
                27B   WriteFile
                116   GetProcAddress
                 FE   GetModuleHandleA
                14C   GetVersion
                 6B   ExitProcess
                246   TerminateProcess
                 D3   GetCurrentProcess
                 D6   GetCurrentThreadId
                24B   TlsSetValue
                248   TlsAlloc
                249   TlsFree
                21E   SetLastError
                24A   TlsGetValue
                 F4   GetLastError
                21B   SetHandleCount
                12A   GetStdHandle
                 EF   GetFileType
                128   GetStartupInfoA
                 4C   DeleteCriticalSection
                 FC   GetModuleFileNameA
                 A3   GetCPInfo
                 9D   GetACP
                109   GetOEMCP
                 96   FreeEnvironmentStringsA
                1AB   MultiByteToWideChar
                 97   FreeEnvironmentStringsW
                 E1   GetEnvironmentStrings
                 E3   GetEnvironmentStringsW
                26E   WideCharToMultiByte
                16C   HeapDestroy
                16A   HeapCreate
                25E   VirtualFree
                 AA   GetCommandLineA
```

```
                17B   InterlockedDecrement
                17E   InterlockedIncrement
                179   InitializeCriticalSection
                 58   EnterCriticalSection
                18F   LeaveCriticalSection
                168   HeapAlloc
                16E   HeapFree
                25B   VirtualAlloc
                190   LoadLibraryA
                12B   GetStringTypeA
                12E   GetStringTypeW
                18D   LCMapStringA
                18E   LCMapStringW
                 F6   GetLocaleInfoA
                 F7   GetLocaleInfoW
                 8E   FlushFileBuffers
                 18   CloseHandle
                229   SetStdHandle
                219   SetFilePointer

     Summary

        5000 .data
        1000 .idata
        1000 .rdata
        1000 .reloc
        7000 .text
```

- **MojDLL.dll - eksport sekcija**

```
Microsoft (R) COFF Binary File Dumper Version 5.00.7022
Copyright (C) Microsoft Corp 1992-1997. All rights reserved.


Dump of file mojdll.dll

File Type: DLL

          Section contains the following Exports for MojDLL.dll

                    0 characteristics
               3A013B8C time date stamp Thu Nov 02 11:01:48 2000
                   0.00 version
                      1 ordinal base
                      1 number of functions
                      1 number of names

               ordinal hint   name

                     1    0   Aritmetika  (00001000)

     Summary

        5000 .data
        1000 .idata
        1000 .rdata
        1000 .reloc
        7000 .text
```

- **Prvi.exe - import sekcija**

```
Microsoft (R) COFF Binary File Dumper Version 5.00.7022
Copyright (C) Microsoft Corp 1992-1997. All rights reserved.


Dump of file prvi.exe

File Type: EXECUTABLE IMAGE

          Section contains the following Imports

             MojDLL.dll
                   0   Aritmetika

             MFC42D.DLL
                Ordinal  1298
                Ordinal  3578
                Ordinal  3463
                Ordinal  2651
                Ordinal  3891
                Ordinal   955
                Ordinal  4141
                Ordinal  2687
                Ordinal  2239
                Ordinal  2238
                Ordinal  3326
                Ordinal  4543
                Ordinal  3261
                Ordinal  1426
                Ordinal  3539
                Ordinal  4036
                Ordinal  1819
                Ordinal  3982
                Ordinal  4828
                Ordinal  3506
                Ordinal  4073
                Ordinal  2101
                Ordinal  1373
                Ordinal  1891
                Ordinal  3009
                Ordinal  3819
                Ordinal  3553
                Ordinal  2100
                Ordinal  1612
                Ordinal  4830
                Ordinal  2945
                Ordinal  3930
                Ordinal  1405
                Ordinal  1865
                Ordinal  1670
                Ordinal  4067
                Ordinal  2335
                Ordinal  2478
                Ordinal  2565
                Ordinal  3577
                Ordinal  2474
                Ordinal  2566
                Ordinal  2336
                Ordinal  2437
                Ordinal  2334
                Ordinal  3073
                Ordinal  3074
```

```
Ordinal  3072
Ordinal  2436
Ordinal  3266
Ordinal  3670
Ordinal  3548
Ordinal  4348
Ordinal   384
Ordinal   707
Ordinal   594
Ordinal  4047
Ordinal  1328
Ordinal  2019
Ordinal  4057
Ordinal  3517
Ordinal  3823
Ordinal  3887
Ordinal  1896
Ordinal  4540
Ordinal  3262
Ordinal  1425
Ordinal  4038
Ordinal  1821
Ordinal  4070
Ordinal  2102
Ordinal  1374
Ordinal  3557
Ordinal  4053
Ordinal  1866
Ordinal  3546
Ordinal  3942
Ordinal  3946
Ordinal  2829
Ordinal   314
Ordinal   570
Ordinal  1812
Ordinal  1795
Ordinal  3416
Ordinal  1111
Ordinal  4809
Ordinal  3713
Ordinal  1633
Ordinal   627
Ordinal   642
Ordinal  1574
Ordinal  3311
Ordinal  3458
Ordinal  4318
Ordinal  1757
Ordinal   410
Ordinal  1651
Ordinal   392
Ordinal   640
Ordinal   672
Ordinal   591
Ordinal  1696
Ordinal  2429
Ordinal  4757
Ordinal  4666
Ordinal  3034
Ordinal  4320
Ordinal  1775
```

```
Ordinal   344
Ordinal   470
Ordinal  4020
Ordinal  4408
Ordinal   406
Ordinal  3564
Ordinal  3685
Ordinal  3689
Ordinal  1493
Ordinal  3426
Ordinal  3097
Ordinal  1675
Ordinal  1825
Ordinal  3944
Ordinal  3947
Ordinal  3579
Ordinal  2650
Ordinal   954
Ordinal  2240
Ordinal  3327
Ordinal  1820
Ordinal  4074
Ordinal  3554
Ordinal  3549
Ordinal   680
Ordinal   669
Ordinal   493
Ordinal   462
Ordinal   385
Ordinal  4418
Ordinal  1590
Ordinal  1889
Ordinal  2082
Ordinal  2081
Ordinal  4396
Ordinal  2276
Ordinal  4490
Ordinal  1663
Ordinal  1253
Ordinal  3345
Ordinal  1666
Ordinal  3571
Ordinal  4048
Ordinal  1329
Ordinal  2020
Ordinal  1060
Ordinal  1981
Ordinal  1824
Ordinal  2420
Ordinal  3980
Ordinal  3981
Ordinal  3979
Ordinal  3854
Ordinal  3728
Ordinal  3869
Ordinal  3508
Ordinal  3518
Ordinal  3824
Ordinal  4075
Ordinal  3817
Ordinal  2033
```

```
Ordinal  1339
Ordinal   423
Ordinal   649
Ordinal  1492
Ordinal  3431
Ordinal  3875
Ordinal  3663
Ordinal  3660
Ordinal  3658
Ordinal  3443
Ordinal  4829
Ordinal  3588
Ordinal  1913
Ordinal  1894
Ordinal  4273
Ordinal  3152
Ordinal  1008
Ordinal  3994
Ordinal  1828
Ordinal  2637
Ordinal  4089
Ordinal  4091
Ordinal  3265
Ordinal  3708
Ordinal  4101
Ordinal  4077
Ordinal  4266
Ordinal  3668
Ordinal  3545
Ordinal  2048
Ordinal  1353
Ordinal  2930
Ordinal   505
Ordinal   689
Ordinal  4818
Ordinal   563
Ordinal  4090
Ordinal  4006
Ordinal   305
Ordinal   974
Ordinal   396
Ordinal  2788
Ordinal  3341
Ordinal  4602
Ordinal  2079
Ordinal  2127
Ordinal  4850
Ordinal  1461
Ordinal  3419
Ordinal  3707
Ordinal  2419
Ordinal  2422
Ordinal  4811
Ordinal  1892
Ordinal  1909
Ordinal  4040
Ordinal  4278
Ordinal  1411
Ordinal  4166
Ordinal  2481
Ordinal  4245
```

```
Ordinal  3544
Ordinal  3870
Ordinal  3825
Ordinal  1835
Ordinal  3524
Ordinal  2722
Ordinal  2504
Ordinal  4536
Ordinal  3272
Ordinal  4569
Ordinal  4674
Ordinal  3669
Ordinal  3547
Ordinal   325
Ordinal   578
Ordinal  3813
Ordinal   742
Ordinal   765
Ordinal  3280
Ordinal  1302
Ordinal  1985
Ordinal  3662
Ordinal  3661
Ordinal  1544
Ordinal  3442
Ordinal  3643
Ordinal  3851
Ordinal  3933
Ordinal  3499
Ordinal  3504
Ordinal  3718
Ordinal  3844
Ordinal  3615
Ordinal  3625
Ordinal  3624
Ordinal  3612
Ordinal  3614
Ordinal  3611
Ordinal  3874
Ordinal  3872
Ordinal  3278
Ordinal  4037
Ordinal  4078
Ordinal  2944
Ordinal  1404
Ordinal  3551
Ordinal   687
Ordinal   503
Ordinal  4052
Ordinal  2187
Ordinal   474
Ordinal  1017
Ordinal  1904
Ordinal  1352
Ordinal  2047
Ordinal  1016
Ordinal  3268
Ordinal  3770
Ordinal  3216
Ordinal  3748
Ordinal  3863
```

```
    Ordinal  3726
    Ordinal  2107
    Ordinal  1377
    Ordinal  3990
    Ordinal  4804
    Ordinal  1515
    Ordinal  3714
    Ordinal  1385
    Ordinal  1901
    Ordinal  4076
    Ordinal  3555
    Ordinal  1611
    Ordinal  3550
    Ordinal  2034
    Ordinal  1340
    Ordinal   426
    Ordinal   650
    Ordinal  1656
    Ordinal   991
    Ordinal   427
    Ordinal  3432
    Ordinal   702
    Ordinal  2126
    Ordinal  1650
    Ordinal  3943
    Ordinal  3425
    Ordinal  1514
    Ordinal  4716
    Ordinal  3006
    Ordinal  1807
    Ordinal  1262
    Ordinal  1073

MSVCRTD.dll
     B2   _adjust_fdiv
     7E   __p__commode
     86   __p__fmode
     97   __set_app_type
     E2   _except_handler3
     99   __setusermatherr
     5D   _XcptFilter
    129   _initterm
     EB   _exit
    1A1   _onexit
     6A   __dllonexit
    2B5   memset
    265   exit
    266   exp
    2CC   sin
     6D   __getmainargs
     7C   __p__acmdln
     CC   _controlfp
    10B   _ftol
     5E   __CxxFrameHandler
    1C6   _setmbcp

KERNEL32.dll
    29B   lstrcpyA
    1AA   MulDiv
    160   GlobalLock
    155   GlobalAlloc
```

```
              FE   GetModuleHandleA
             128   GetStartupInfoA
             166   GlobalUnlock
              98   FreeLibrary
             190   LoadLibraryA
             116   GetProcAddress

         USER32.dll
             15E   IsClipboardFormatAvailable
              AF   EmptyClipboard
              E6   GetClipboardData
             1E9   SetClipboardData
              38   CloseClipboard

         GDI32.dll
              17   CopyMetaFileA
              45   DeleteMetaFile

         MFCO42D.DLL
            Ordinal   899

Summary

    1000 .data
    2000 .idata
    2000 .rdata
    1000 .reloc
    5000 .rsrc
    7000 .text
```