S A D R Ž A J:

1. UVOD

Microsoft Excel je zasigurno jedan od najkompleksnijih programskih paketa, koji se nalazi u širokoj upotrebi među korisnicima. Excel sadrži više od 300 standardno uključenih tabličnih funkcija različitih tipova (matematičkih, financijskih, statističkih i dr.) Osim toga u njemu je integriran programski jezik Visual Basic za aplikacije (VBA) koji omogućava programiranje specijalnih funkcija, koje nisu standardno uključene u Excel.

Pored toga Excel omogućava komunikaciju sa drugim aplikacijama – povezivanje sa programima i različitim datotekama koje sadrže tekst, sliku, grafove, tablice, itd. Tako se postiže integriranje više programa i datoteka u jednu cjelinu formiranu oko Excela kao središnje aplikacije za unos i prikaz različitih tipova podataka. Također, sposobnost Excela za povezivanje s drugim aplikacijama omogućava iskorištavanje već postojećih skupova podataka iz drugih datoteka, te ih nije potrebno ponovno pisati u Excelu.

Posebno je značajna sposobnost Excela za povezivanje s izvršnim datotekama izrađenim u različitim programskim jezicima, kao npr. C, C++, Pascal, Fortran, itd. Takva komunikacija potrebna je kada već postoji samostalan funkcionalan program napisan u nekom od programskih jezika. Umjesto dugotrajnog zahtjevnog prevođenja programskog koda u Visual Basic, jednostavnije je povezivati Excel s gotovim programom.
Također, mogućnosti samog VBA su znatno manje od mogućnosti profesionalnih programskih paketa, pa je izrada kompleksnih programa brža i jednostavnija, a ponekad i jedino moguća s takvim naprednijim programerskim alatima.

Postoji više načina povezivanja Excela s drugim programima i datotekama, a u ovom radu su podrobnije objašnjena dva oblika komunikacije i to sa programima izrađenim programskim jezikom Visual C++ koji se trenutno najčešće primjenjuje za izradu novih aplikacija.

Prvi, najjednostavniji način predstavlja povezivanje Excela sa izvršnom datotekom preko tekstualnih datoteka. To je indirektni način povezivanja u kojemu C++ program ne pristupa izravno podacima napisanim u Excelu, već te iste podatke čita/piše u/iz tekstualne datoteke.

Drugi često korišteni način povezivanja Excela sa C++ je direktnim putem primjenom biblioteka za dinamičko povezivanje (DLL – *Dynamic Link Library*), pri čemu se umjesto C++ izvršne datoteke izrađuje DLL napisan u Visual C++.

Cilj ovog rada je prikaz mogućnosti i osnovnih principa rada navedenih načina povezivanja Excel dokumenta sa drugim programima i utvrđivanje složenosti izrade, te brzine i stabilnosti rada pri njihovoj komunikaciji.

2. MICROSOFT EXCEL

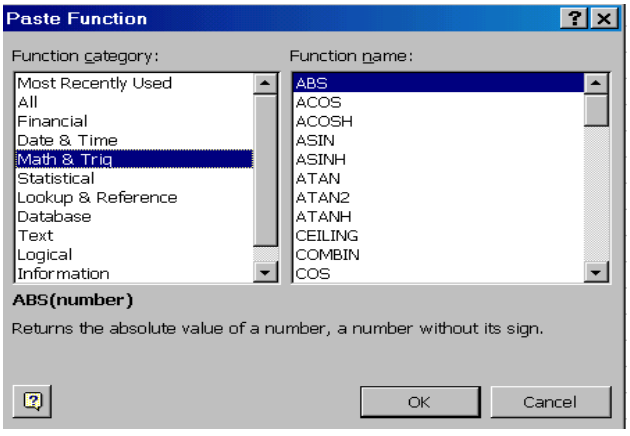
Microsoft Excel je najraširenija i najčešće upotrebljavana aplikacija za tablične proračune. Vrlo je jednostavan za upotrebu i omogućava korištenje kalkulatora s brojnim ugrađenim funkcijama, formiranje (programiranje) novih funkcija korištenjem VBA (*Visual Basic for Application*) alata, a također omogućava komunikaciju sa raznim eksternim aplikacijama.

2.1 Općenito o Microsoft Excelu

Excel ima vrlo raznoliku primjenu, ali najviše se upotrebljava kao tablični kalkulator, te za grafičke prikaze tabličnih proračuna. Excel podržava razne tipove podataka koji se nalaze u tablici, tj. razlikuje brojeve, tekst, datum, vrijeme, valutu, postotke i razlomke. Na osnovu toga o kojem se tipu radi, Excel dozvoljava upotrebu određenih računskih operacija i primjenu funkcija koje prihvaćaju te tipove podataka. Excel pruža još mnogo mogućnosti kao što je izrada scenarija, sumarne tablice, statističke analize podataka, itd.

2.1.1 Upotreba ugrađenih funkcija

U izborniku *Insert* nalazi se naredba *Function* koja ispisuje rezultat izabrane funkcije u selektiranu tabličnu ćeliju. Nakon što je odabrana kategorija funkcije, npr. matematička funkcija, odabire se podfunkcija iz te kategorije. Na Sl. 2.1.1 prikazan je odabir matematičke funkcije (*Math & Trig*) koja daje apsolutnu vrijednost zadanog broja (*ABS*).



Sl. 2.1.1 Okvir za dijalog *Paste Function*

U prozor, koji je otvoren nakon odabira funkcije, unosi se vrijednost parametara funkcije (u ovom slučaju broj kojem se traži apsolutnu vrijednost). Unos vrijednosti parametara funkcije može se napraviti na dva načina: direktnim unosom broja ili odabirom ćelije u Excelu gdje se taj broj nalazi.

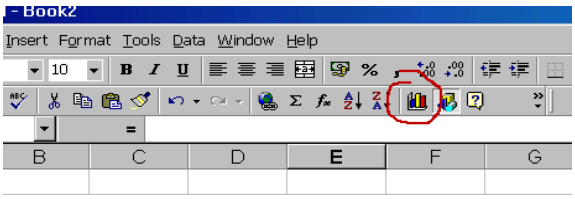
2.1.2 Izrada grafikona

Grafički prikaz podataka iz Excel tablice omogućen je bez obzira da li su podaci samo uneseni u tablicu ili su izračunati preko raznih formula i funkcija. Grafikon može biti prikazan na osnovu podataka smještenih po stupcima (*columns*) ili po redovima (*rows*).

stupac1	stupac2	stupac3
23	35	40
21	34	36
18	30	30
15	28	25
11	25	0
0	1	

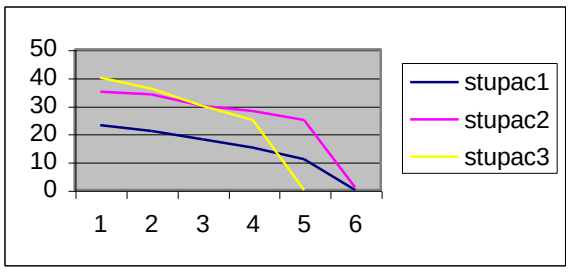
Sl. 2.1.2 Radna tablica

Nakon što su podaci uneseni u tablicu, crtanje grafa započinje odabirom naredbe *Chart* iz izbornika *Insert* ili odabirom ikone sa trake sa alatima kao na Sl. 2.1.3:



Sl. 2.1.3 Ikona *Chart* (graf)

Moguć je prikaz različitih vrsta grafova kao što su linijski, stupčani, kružni, horizontalni, trodimenzionalni, itd. Grafički se prikaz može odnositi na cijelu tablicu ili samo na selektirane dijelove. Jedna od mogućnosti prikaza tablice je graf na Sl. 2.1.4.



Sl. 2.1.4 Grafikon na osnovu radne tablice

U svakom trenutku pristup bilo kojem dijelu grafa omogućen je pritiskom na desnu tipku miša, i na taj način moguće je promijeniti izgled grafa, njegovih osi, legende i niza drugih parametara za oblikovanje grafova.

2.2 Visual Basic za aplikacije (VBA) u Excelu

Visual Basic je programski jezik koji se temelji na najpopularnijem programskom jeziku BASIC-u, ali ga svojim poboljšanim i proširenim mogućnostima, te novom koncepcijom s grafičkim sučeljem daleko nadilazi. Excel sadrži posebnu verziju Visual Basica koju je Microsoft nazvao *Visual Basic for Applications* – VBA (Visual Basic za aplikacije).
VBA kod se u Excelu nalazi unutar tzv. VBA makroa. Za izradu makro programa nije potrebno predznanje o programiranju te se oni vrlo jednostavno kreiraju snimanjem niza aktivnosti korisnika u samom Excelu, pri čemu Excel automatski generira Visual Basic kod. Za preuređivanje koda makro programa potrebno je aktivirati Visual Basic editor, te u njemu napraviti željene izmjene programa.

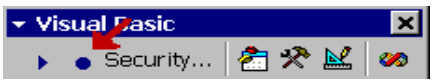
2.2.1 Snimanje makroa

Snimanje makroa je najlakše objasniti pomoću primjera. Formatiranje određenog raspona ćelija započinje unosom podataka u radnu tablicu (Sl. 2.2.1):

	A	B	C	D	E
1					
2		32	54	6	667
3		333	66	44	1
4		14	434	1	22

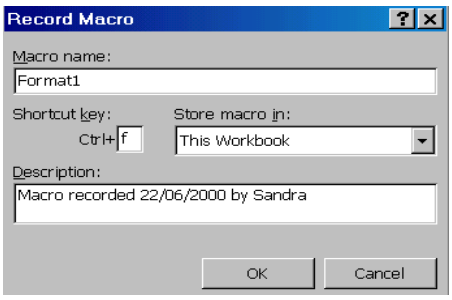
Sl. 2.2.1 Podaci za snimanje makroa

Nakon selektiranja jedne od ćelija iz radne tablice, kreiranje makroa započinje snimanjem operacija korištenjem *Tools – Record Macro – Record New Macro* naredbe ili *Record Macro* alata iz Visual Basic retka s alatima (Sl. 2.2.2).



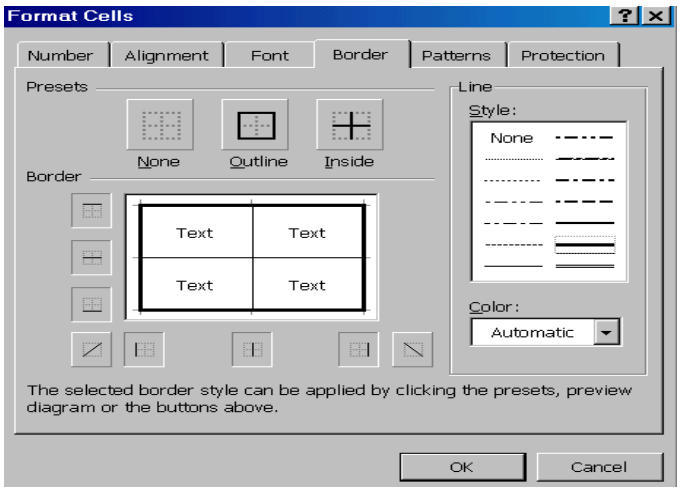
Sl. 2.2.2 Visual Basic redak sa alatima

Nakon pokretanja operacije snimanja makroa, otvara se okvir za dijalog u kojem se unosi ime i opis makroa, te *Shortcut Key* (Ctrl+slovo), tj. definira se kombinacija tipki koje se mogu upotrijebiti za brzo pokretanje makroa. Brojevi nisu dozvoljeni u kombinaciji.



Sl. 2.2.3 Okvir za dijalog *Record Macro*

U sljedećem koraku na zaslonu se prikazuje ikona *Stop Macro* alata, kojim se može u svakom trenutku zaustaviti snimanje makroa. Prije zaustavljanja snimanja makroa, snimaju se operacije koje će taj makro obavljati. Da bi se pri svakom ponovnom korištenju makro odnosio uvijek na cijelu tablicu, bez obzira na veličinu, odabrana je opcija *Current Region* smještena u izborniku *Edit–Go To – Special*. Formatiranje ćelija radi se u *Format Cells* okviru za dijalog, a tipovi rubnih i unutrašnjih linija postavljaju se u *Border* pretincu (Sl. 2.2.4).



Sl. 2.2.4 Format Cells okvir za dijalog

Isključivanje opcije *Gridlines* u *View* pretincu iz izbornika *Tools–Options* omogućava prikaz selektirane tablice na čistom radnom listu.

Nakon snimanja svih potrebnih operacija, snimanje makroa zaustavljeno je pomoću *Stop Macro* alata ili odabirom *Stop Recording* naredbe iz *Tools–Record Macro* podizbornika.

Sl. 2.2.5 *Stop Macro* alat

Rezultat izvršavanja makroa je prikaz kao na Sl. 2.2.6:

	A	B	C	D	E	F
1						
2		32	54	6	667	
3		333	66	44	1	
4		14	434	1	22	
5						

Sl. 2.2.6 Izgled radne tablice nakon izvođenja makroa

Prilikom snimanja, makro inicijalno koristi apsolutne reference. Korištenje relativnih referenci omogućeno je odabirom naredbe *Tools–Record Macro–Use Relative Reference* ili odabirom *Relative Reference* sa alata *Stop Macro*. Prednost korištenja relativnih referenci je mogućnost izvođenja makroa na bilo kojem mjestu u radnoj tablici, dok sa druge strane, upotreba apsolutnih referenci omogućava izvođenje makroa na istim ćelijama bez obzira na trenutni položaj u radnoj tablici.

Nakon što je makro snimljen njegov Visual Basic kod može se u svakom trenutku pogledati odabirom naredbe *Tools–Macro–Visual Basic Editor* ili odabirom tipki CTRL + F11.
U Visual Basic modulu *Module1* nalazi se makro koji izgleda ovako:

```
Sub Format1()
' Format1 Macro
' Keyboard Shortcut: Ctrl+f

    Selection.CurrentRegion.Select
    Selection.Borders(xlDiagonalDown).LineStyle = xlNone
    Selection.Borders(xlDiagonalUp).LineStyle = xlNone
    With Selection.Borders(xlEdgeLeft)
        .LineStyle = xlContinuous
        .Weight = xlThick
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlEdgeTop)
        .LineStyle = xlContinuous
        .Weight = xlThick
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlEdgeBottom)
        .LineStyle = xlContinuous
        .Weight = xlThick
```

```
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlEdgeRight)
        .LineStyle = xlContinuous
        .Weight = xlThick
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlInsideVertical)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlInsideHorizontal)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
    ActiveWindow.DisplayGridlines = False
End Sub
```

U ovom kodu napisani su svi gore navedeni postupci, tj. selektirano je cijelo područje tablice (*Selection.CurrentRegion.Select*), te su naznačene vrste linija tablice i to svih vanjskih (*xlEdgeLeft, xlEdgeTop, xlEdgeBottom, xlEdgeRight*) pa onda i svih unutrašnjih (*xlInsideVertical, xlInsideHorizontal).* Opcija *Gridlines* isključena je tako što joj je postavljena vrijednost *False* (*ActiveWindow.DisplayGridlines = False*).

2.2.2 Izvođenje makroa

Nakon što je makro snimljen, može se pokrenuti u bilo kojem trenutku na više načina:

- pokretanjem *Tools–Macro* naredbe, otvara se *Macro Name/Reference* okvir u kojem se odabire ime makroa, a makro se pokreće pomoću naredbe *Run*,
- pritiskom na kombinaciju tipki za pokretanje makroa (u ovom slučaju Ctrl+f),
- pokretanjem *Run Macro* alata iz Visual Basic retka sa alatima,
- pokretanjem alata, objekta *button* ili nekog drugog grafičkog objekta kojem je pridružen makro.

Bilo koja tablica sada se može formatirati kao u prethodnom primjeru, jednostavnim pokretanjem već snimljenog makroa na jedan od gore navedenih načina.

2.2.3 Pridruživanje makroa objektu *button*

Button je grafički objekt koji sadrži tekst koji se može uređivati i formirati kao i svaki drugi. Kreira se pomoću *Create Button* alata koji se nalazi u *Toolbar–Forms* retku sa alatima.
Makro se vrlo jednostavno pridružuje *button* objektu. Nakon što je iz *Forms* retka sa alatima pokrenut *Button* alat, te je pomoću miša selektirano mjesto i veličina objekta, na zaslonu se automatski nakon otpuštanja tipke miša pojavljuje *Assign Macro* okvir za dijalog.

Sl. 2.2.7 *Forms* redak sa alatima

Sl. 2.2.8 Kreiranje objekta

U *Macro Name/Reference* okviru odabere se ime makroa koji se pridružuje ili se može snimiti i novi makro odabirom funkcije *Record*. Naziv kreiranog objekta se mijenja pritiskom na sliku objekta sa desnom tipkom miša te odabirom *Edit text* naredbe.

2.2.4. Organizacija Visual Basic modula

Većina Visual Basic modula sastoji se od dva dijela:

- područje deklaracija
- područje procedura i funkcija.

Modul započinje deklaracijom niza varijabli i konstanti te se definiraju pojedine Visual Basic opcije. Deklaracije se mogu poredati u bilo kojem redoslijedu, ali preporučuje se pridržavanje konvencije po kojoj se najprije navode opcije, zatim deklariraju varijable i na kraju konstante.

Područje deklaracija	*Option Explicit*	Visual Basic opcije
	Dim x As Integer	Deklaracija varijabli
	Const Delta=10	Deklaracija konstanti
Visual Basic procedure	*Sub kreiraj()* *...* *End Sub*	procedura
	Function izračunaj(x) *...* *End Function*	funkcija

Sl. 2.2.9 Dijelovi Visual Basic modula

U bilo kojem dijelu koda smije se napisati komentar koji povećava razumljivost procedura, a posebno je bitan ako se procedura prilagođava nekim novim uvjetima. Komentar započinje s apostrofom (') i nastavlja se do kraja retka. Ako se *kod* nalazi iza apostrofa, on se smatra dijelom komentara i neće se izvoditi. Npr.:

' Ovo je komentar

2.2.5. Tipovi podataka u Visual Basicu

Visual Basic podržava nekoliko osnovnih tipova podataka.

- Boolean – 2 bajta
- Integer – 2 bajta
- Long – 4 bajta
- Single – 4 bajta
- Double – 8 bajta
- Currency – 8 bajta
- Date – 8 bajta
- String – 1bajt/znaku

Pored navedenih postoje i dva posebna tipa podataka: objektni tip – Object – koristi se za čuvanje referenci objekata, a zauzima 4 bajta; varijantni tip – Variant – može predstavljati različite vrijednosti (do maksimalne vrijednosti kao za double tip, te datuma i vremena) ili tekst. Kod Variant tipa varijabli programer ne vodi računa o tipu podataka koji sadrži. Ipak, ako se izvode automatske operacije ili funkcije sa Variant podacima, onda oni moraju biti brojevi. Variant varijabla ne može sadržavati korisnički definirani tip.

Kombiniranjem osnovnih tipova definiraju se i dvije vrste složenih tipova: korisnički definirani tipovi – definiraju se pomoću naredbe *Type* i polja – *Array*

2.2.6. Deklaracija varijabli i konstanti

Naredba za deklariranje varijabli ima sljedeći oblik:

Dim *ime* As *tip*

Varijable u VBA nije potrebno eksplicitno deklarirati: automatski se podrazumijeva da se radi o varijabli Variant tipa. Ako je varijabla deklarirana, a nije naveden njen tip (npr. *Dim x*), također se podrazumijeva da je varijabla Variant tipa. Međutim, korištenje eksplicitne deklaracije omogućava otkrivanje pogrešaka u vezi s imenima varijabli (npr. krivo utipkano ime). Eksplicitno deklariranje svake varijable obavezno je ako se na početku VBA modula navede naredba

Option Eksplicit

ili ako je uključena opcija *Require Variable Declaration* u *Module General* pretincu *Tools – Options* okvira za dijalog.

Deklaracija korisnički definiranih tipova podataka vrši se na vrhu VBA modula. Korisnički definirani tipovi podataka odgovaraju strukturama u programskom jeziku C. Radi se o tipu podataka čije varijable sadržavaju nekoliko različitih tipova podataka.
Npr. deklaracijom:

```
Type Student
        Ime As String
        Adresa As String
        Položio As Currency
        Broj_prodanih As Integer
End Type
Dim St As Student
```

definira se tip *Student*, koji se sastoji od 4 osnovna tipa. Elementima varijable pristupa se pomoću operatora točke, npr.

```
St.Ime = "Marin"
St.Položio = 20
```

Polje (*array*) je skup koji služi za pohranjivanje niza istovrsnih podataka. Svako polje ima svoju gornju i donju granice, a veličina mu se može mijenjati pomoću *ReDim* naredbe. Primjer deklaracije polja:

Dim x(25) As Integer

Indeksi elemenata polja počinju od 0 (donja granica) i kreću se do N–1 (gornja granica), tj. u ovom slučaju do 24. Međutim, donja i gornja granicu polja može se i eksplicitno navesti, npr.

Dim x(1 To 25) As Integer

U VBA dozvoljena je deklaracija polja do 60 dimenzija. Primjer deklaracije dvodimenzionalnog polja sa eksplicitno deklariranim granicama:

Dim A (1 To 5, 2 To 10) As Double

Ako se tijekom izvođenja programa trebaju mijenjati dimenzije polja, tada se polje deklarira kao dinamičko polje:

Dim Polje()

a alokacija memorijskog prostora za dinamičko polje, odnosno promjena dimenzije polja vrši se *ReDim* naredbom unutar procedure:

ReDim Polje (5,10)

Pri svakom izvođenju *ReDim* naredbe gube se vrijednosti spremljene u polju. Ako je potrebno pri promjeni dimenzije polja sačuvati podatke, koristi se ključna riječ *Preserve*:

ReDim Preserve Niz(Ubound(Niz)+10)

Kod višedimenzionalnih polja uz korištenje *Preserve* ključne riječi, smije se mijenjati samo veličina posljednje dimenzije i to njezina gornja granica. Vrijednost gornje granice daje funkcija *Ubound.*

Visual Basic za aplikacije sadrži mnoge ugrađene konstante. Npr. *xlWorksheet* je konstanta koja identificira radnu tablicu kao tip lista radne bilježnice. Sve ugrađene konstante koje se odnose na Excelove objekte počinju s *xl*, a sve ostale počinju sa *vb*.

Definicija vlastitih konstanti omogućena je korištenjem naredbe *Const* sljedećeg oblika:

Const *IMEKONSTANTE*= izraz

Uobičajeno je da se imena konstanti pišu velikim slovima, npr.:

Const PI = 3.14159265
Const MAX_BROJ = 100

2.2.7 Visual Basic procedure

Pravila za imena procedura, varijabli, parametara i konstanti su: prvi znak mora biti slovo, VBA ne razlikuje velika i mala slova u imenima i ne smiju se koristiti ključne riječi i imena koja označavaju ćelijske reference kao imena funkcijskih procedura.

Razlikuju se dva osnovna tipa procedure:

Potprogrami ili obične procedure – izvode operacije, ali ne vraćaju vrijednosti. Počinju sa *Sub* i završavaju s *End Sub*, npr.:

```
Sub TabličnaMreža (mreža)
        If mreža = True Then
          ActiveWindow.DisplayGridlines = True
        Else
          ActiveWindow.DisplayGridlines = False
        End If
End Sub
```

Funkcije ili funkcijske procedure – izvode operacije i vraćaju vrijednost. Počinju sa *Function*, a završavaju s *End function*, npr.:

```
Function Faktorijel(n)
        If n = 0 Then
          Fakorijelt = 1
Else
         Faktorijel = n * Faktorijel(n-1)
End If
End Function
```

Osnovni dijelovi procedure su:

- ključne riječi (*Sub* – *End Sub* ili *Function* – *End Function*) koje označavaju početak i završetak procedure,
- ime (u primjerima *TabličnaMreža* i *Faktorijel*) – jednoznačno identificira proceduru,
- argumenti (*mreža* i *n*) – vrijednosti koje se prosljeđuju proceduri, a predstavljaju početnu točku za računanje ili određivanje operacija koje treba izvesti,
- tijelo procedure – niz VBA naredbi,
- povratna vrijednost – vrijednost koju vraća funkcijska procedura.

Procedure mogu međusobno pozivati jedne druge i to na način da ako procedura ne prima argument navodi se samo ime procedure, a ako prima argument onda se poziva s imenom i s vrijednosti argumenta, npr.:

```
Sub Izvještaj()
        Kreiraj
        Faktorijel (3)
End Sub
```

Postoje dvije metode za prijenos podataka između procedura po vrijednosti i po referenci. Inicijalno se u VBA koristi prijenos po referenci (prenosi se adresa varijable), dok se za prijenos po vrijednosti uz ime argumenta mora koristiti izraz *ByVal*, npr.:

```
Function UčitajStr (s As String, ByVal n As Integer)
....
End Function
```

Upotrebom izraza *ByRef* može se naglasiti da se neka varijabla prenosi po referenci.

2.2.8. Objekti, svojstva i metode

Objekt je entitet kojim se može upravljati iz Visual Basica. Svaki objekt ima karakteristike koje ga čine upotrebljivim – svojstva (*properties*). Uz objekt se vezuje i pojam metode (*method*). Metode su akcije (operacije) koje objekti izvršavaju.
U Excelu, VBA razlikuje preko 100 različitih objekata. Neki od najvažnijih su: *Application, Workbook, Worksheet, Module, Chart, Range, Window, Font, Style, Name, Toolbar, Menu*...
Ćelija nije zaseban objekt u Visual Basicu, već se za rad s njom koristi objekt raspona (*Range*) veličine jedne ćelije. Veliki dio VBA koda odnosi se na upravljanje objektima kojima se postavljaju ili očitavaju neka određena svojstva.
Naredba za postavljanje vrijednosti svojstva ima sljedeću sintaksu:

```
objekt.svojstvo = izraz
```

gdje je *objekt* – referenca objekta, *svojstvo* – ime svojstva objekta i *izraz* – vrijednost koja se pridružuje svojstvu, npr.:

```
Raspon1. ColumnWidth = 20
```

Stupcima u rasponu *Raspon1* postavlja se širina od 20 znakova.
Naredba za očitavanje vrijednosti svojstva:

```
varijabla = objekt.svojstvo
```

gdje je *varijabla* – varijabla ili drugo svojstvo u koje se sprema očitana vrijednost svojstva, *objekt* – referenca objekta i *svojstvo* – ime svojstva, npr.

```
x = Dohodak.Value
```

Različiti tipovi objekta imaju različita svojstva a najčešće se koriste: *ActiveCell, ActiveSheet, ActiveWorkbook, Bold, Column, Row, Width, Value, Visible...*

Metode pripadaju objektima kao i svojstva, ali one predstavljaju operacije koje objekt izvršava.
U VBA kodu moguća su dva oblika korištenja metoda, i to:

- Objekt.metoda – u slučaju metode bez argumenata
- Objekt. metoda. argumenti – u slučaju metode s argumentima.

Da bi se sačuvala vrijednost koju vraća metoda, koristi se lista argumenata u zagradama, npr. za Cells metodu:

```
Rn = Sheet1.Cells(1,1)          ' varijabli Rn pridružuje se referenca ćelije A1
```

VBA razlikuje preko 100 različitih metoda objekata, a najčešće se koriste:
Activate, Add, Calculate, Cells, Clear, Close, Open, Range, Save, Undo, Run, Worksheets...

2.2.9. Kontrola izvođenja programa

Visual Basic posjeduje skup naredbi kojima kontrolira odvijanje programa: naredbe za izvođenje skupa naredbi, grananje i realiziranje višeznačnih odluka.

If – Then i If – Then – Else naredbe:
If – Then i If – Then – Else su naredbe za izvršavanje odluka. Ovisno o ispunjenju ili neispunjenju uvjeta izvodi se određeni skup naredbi. Sintaksa je sljedeća:

```
If izraz Then
        Blok naredbi
End If
```

Ako je uvjet izražen izrazom ispunjen, izvodi se skup naredbi između *If* i *End If*.
Za definiranje bloka naredbi od kojih se uvijek jedan izvodi koristi se If – Then – Else naredba:

```
If izraz1 Then
        Blok1 naredbi
ElseIf izraz2 Then
        Blok2 naredbi
ElseIf izraz3 Then
        Blok3 naredbi
…
Else
        BlokN naredbi
End If
```

Select Case naredba:

Select Case je najpogodnija naredba za realiziranje višeznačne odluke na temelju konstantnih vrijednosti. Sintaksa je:

```
Select Case izraz
Case vrijednost1
        Blok1
Case vrijednost2
        Blok2
…
Case Else
        BlokN
End Select
```

U *Select Case* strukturi testira se samo jedan izraz. Ako vrijednost izraza odgovara jednoj od konstantnih vrijednosti, vrši se grananje na pripadajući blok naredbi.
Case Else je posljednji element naredbe i predstavlja slučaj koji se izvodi ako nijedan od prethodnih nije ispunjen.

Do…Loop petlja:

U VBA može se koristiti nekoliko tipova petlji za ponavljanje izvođenja skupa naredbi.

```
Do While izraz
        Blok naredbi
Loop
```

Skup naredbi se izvršava sve dok je izraz istinit. Ako je na samom početku izraz u petlji neistinit, blok naredbi se uopće neće izvršiti.
Drugi oblik *Do… Loop* naredbe je takav da se uvjet testira na kraju petlje. Time se osigurava izvođenje petlje barem jedanput. Sintaksa je:

```
Do
        Blok naredbi
Loop While izraz
```

Osim ovih oblika koji izvode blok naredbi sve dok je izraz istinit, postoje i druga dva oblika koji izvode blok naredbi dok je izraz neistinit.
Sintakse su sljedeće.

```
Do Until izraz
        Blok naredbi
Loop
```

```
Do
        blok naredbi
Loop Until izraz
```

For petlja:

Također omogućava ponavljanje dijelova programa, ali sada je točno određeno koliko puta se petlja izvodi. *For* petlja se izvodi dok se ne dosegne konačna ili granična vrijednost:

```
For brojač = početni uvjet To završni uvjet [Step promjena]
        Blok naredbi
Next [brojač]
```

Dijelovi petlje u uglatoj zagradi nisu obavezni. Ako nije napisan koliki je korak promjena, VBA inicijalno uzima korak od 1

Primjer:

```
Suma = 0
For i = 0 To 100 Step 5
        Suma = Suma + i
Next i
```

Postoji i petlja *For Each… Next* koja je varijacija *For* petlje u kojoj se niz naredbi ponavlja za svaki element koji mora biti ili Variant ili Object tipa.

```
For Each element In grupa
        Blok naredbi
Next element
```

Izlaz iz kontrolnih struktura i procedura:

Exit For i Exit Do:

Ove naredbe osiguravaju prekid izvođenja i izlaz iz *For*, odnosno *Do* petlji.

Primjer:

```
For i = 0 to n
        If godina(i) = 1994 Then
                Izabrano = True
                Exit For
        End If
Next i
```

GoTo naredba:

Općenito se izbjegava u programiranju, ali može biti vrlo korisna.

```
For i = 0 to m
        For j = 0 to n
                If A(i,j) = x GoTo Pronašli
        Nextj
Next i
…
Pronašli:
```

Izlaz iz procedure prije njenog regularnog kraja ostvaruje se s *Exit Sub* ili *Exit Function*.

```
Sub Proc1()
        If Izlaz = True Then
                Exit Sub
        End If
...
EndSub
```

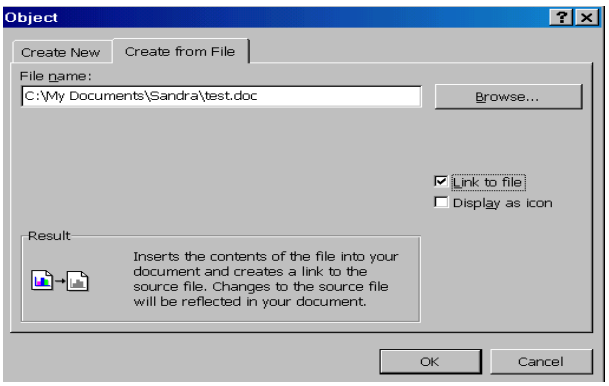
2.3 Komunikacija Excela sa eksternim aplikacijama

Primjenom Visual Basica za aplikacije moguće je automatizirati komunikaciju s drugim aplikacijama i ostvariti njihovo upravljanje iz Excela, te na taj način kreirati korisničke aplikacije koje integriraju različite programe i datoteke.

2.3.1 OLE aplikacije

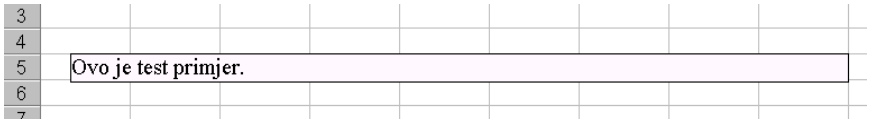
OLE (*Object Linking and Embendding*) je tehnologija razvijena u Microsoftu, koja se sve više prihvaća kao uobičajeni način dijeljenja podataka različitih aplikacija. Za pristup nekoj aplikaciji putem OLE-a, potrebno je da su zadovoljena dva uvjeta. Prvo, aplikacija mora biti napisana tako da zadovoljava OLE specifikaciju, i drugo, aplikacija mora biti instalirana i ispravno ubilježena u *OLE Registry* na sustavu.

OLE omogućava smještanje ili ugradnju nekog objekta izrađenog u jednoj aplikaciji u datoteku izrađenu u drugoj aplikaciji. Npr. može se ugraditi Word dokument u Excelovu radnu tablicu. Nakon što je postavljena točka umetanja u Excelu, odabere se naredba *Insert–Object*, a zatim *Create From File*. Pojavljuje se okvir kao na Sl. 2.3.1 u kojem se pomoću naredbe *Browse* određuje pozicija dokumenta na disku.



Sl. 2.3.1 *Object* okvir za dijalog

Za povezivanje trenutno aktivnog Excel dokumenta s umetnutim Word dokumentom potrebno je uključiti opciju *Link To File*.



Sl. 2.3.2 Umetnuti Word dokument u Excelu

Najjednostavniji način prijenosa manje količine podataka ili grafike iz jedne aplikacije u drugu je korištenje *Copy/Paste* tehnike. U tom slučaju oba dokumenta moraju biti otvorena. Nakon selektiranja područja sa mišem u dokumentu iz kojeg se kopiraju podaci naredbom *Edit–Copy* (ili pomoću tipaka CTRL+C). Podaci su sada kopirani na privremenu lokaciju (*clipboard*). Za kopiranje podataka u Excel definira se mjesto umetanja u tablici i pomoću naredbe *Edit–Paste* podaci se upišu na izabrano mjesto (ili pomoću tipaka CTRL+V).

OLE automatizacija je jedno od najvažnijih svojstava OLE-a. Ona omogućava pisanje makroa u jednoj aplikaciji za nadgledanje objekata u drugoj aplikaciji. Postoje dva različita načina podrške za OLE automatizaciju: objektna ili kontrolna aplikacija. Objektna aplikacija izlaže svoj model drugim aplikacijama i dozvoljava kontrolnim aplikacijama da je nadgledaju. U Microsoft Office paketu samo neke aplikacije mogu biti kontrolne. To su: Excel, Access, Project i Visual Basic.

Za nadziranje OLE automatizacijskog objekta putem makroa, potrebno je:

- postaviti varijablu koja će predstavljati OLE objekt,
- koristiti tu varijablu za pristup objektnom modelu prilikom postavljanja ili dohvaćanja svojstava objekta i pozivanja metoda,
- osloboditi OLE objekt kada više nije potreban.

Primjer pristupanja Word objektu iz Excela: kreiran je makro koji otvara novu datoteku u Wordu, daje joj naslov *Izvještaj*, a zatim formatira naslov masno otisnutim pismom, veličine 32 i centrira naslov vodoravno na stranici:

```
Dim WordAplik As Object
Sub Excelmakro( )
Set WordAplik = CreateObject("Word.Basic")
With WordAplik
        If Ucase (Left(Application.OperatingSystem,3))< > "MAC" Then
                .AppRestore
                .AppMaximaze 1
        Else
                AppActivate "Microsoft Word"
        End If
        .FileNewDefault
        .InsertPara
```

```
        .Insert "Izvještaj"
        .StartOfLine 1
        .Bold
        .FontSize 32
        .CenterPara
    End With
    End Sub
```

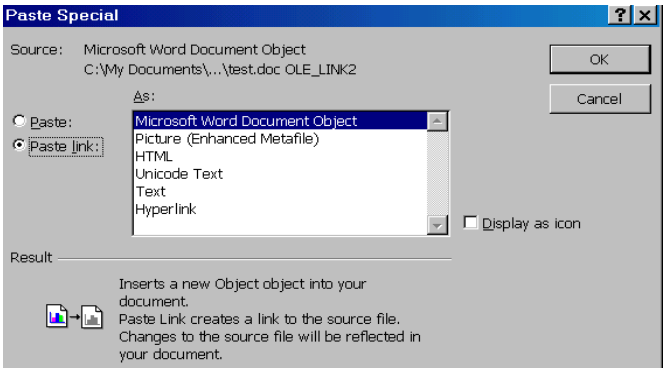
VBA podržava dinamičku razmjenu podataka između dviju Windows aplikacija, koristeći se OLE automatizacijom. Između aplikacija kreiran je komunikacijski kanal preko kojeg se šalju podaci nakon što je napravljena promjena u izvornoj aplikaciji. Podaci se šalju automatski ili na zahtjev korisnika (manualno).
Osnovne karakteristike ovakvog načina povezivanja su:

- moguće je uspostavljanje veze jednog izvornog dokumenta s više različitih prijemnih dokumenata,
- podaci se spremaju u dokumentu izvorne aplikacije,
- podaci i grafika nisu uključeni u Excel dokument, već se prikazuje samo njihov rezultat, pa su dokumenti manji od onih s uključenim objektima,
- moguće je korištenje samo dijela izvorne datoteke,
- ažuriranja su prilikom promjene izvornih podataka automatska ili na zahtjev korisnika.

Osim navedenih prednosti, moraju se uzeti u obzir i nedostaci:

- promjena imena ili premještanje izvorne datoteke može narušiti vezu,
- sve Windows aplikacije ne podržavaju ovakav način povezivanja podataka,
- automatsko ažuriranje usporava rad računala.

Za kreiranje veze izvorni dokument treba biti snimljen i obje Windows aplikacije (npr. Word i Excel) moraju biti otvorene. U izvornom dokumentu se nakon selektiranja podataka koji se povezuju, naredbom *Edit–Copy* kopiraju podaci na privremenu lokaciju. Sada se u Excel dokumentu pokreće *Edit–Paste Special* naredba koja otvara okvir za dijalog kao na Sl. 2.3.3:



Sl. 2.3.3 *Paste Special* okvir za dijalog

U okviru *As* odabran je tip podatka koji se dodaje u dokument i opcija *Paste Link*. Opcija *Paste Link* nije dostupna kada se u odlagalištu nalazi datoteka grafičkog formata (slika). Za povezivanje

slike, upotrebljava se naredba *Paste Picture Link* iz izbornika *Edit*. Ova naredba zamjenjuje naredbu *Paste Special* kad se otvori izbornik *Edit*, a istovremeno je pritisnuta tipka SHIFT.

Na ovaj način se uspostavlja dinamička veza između Word i Excel aplikacije. Ako se promijeni podatak u Wordu koji je dinamički povezan sa Excelom, tada će se podaci ažurirati automatski. Podaci se mogu mijenjati i manualno (na zahtjev korisnika) ako je u izbornik *Edit–Links* odabrana opcija *Manual*.

2.3.2 Dinamička razmjena podataka (DDE)

Neke aplikacije dostupne na tržištu još uvijek ne podržavaju OLE komunikaciju. Za razmjenjivanje podataka sa tim aplikacijama ili za njihov nadzor, koristi se DDE (*Dynamic Data Exchange*) komunikacija.
VBA u Excelu nudi sljedeća svojstva i metode DDE komunikacije:

- *DDEAppReturnCode*: svojstvo objekta *Application*. Vraća DDE povratni kod koji se nalazio u zadnjoj DDE poruci potvrde koju je Excel primio,
- *DDEExecute*: metoda *Application* objekta. Pokreće naredbu ili obavlja radnju u drugoj aplikaciji putem navedenog DDE kanala,
- *DDEInitiate*: metoda *Application* objekta. Otvara DDE kanal prema drugoj aplikaciji,
- *DDEPoke*: metoda *Application* objekta. Šalje podatke drugoj aplikaciji putem DDE kanala,
- *DDERequest*: metoda *Application* objekta. Traži informacije od aplikacije putem DDE kanala,
- *DDETerminate*: metoda *Application* objekta. Zatvara DDE kanal prema drugoj aplikaciji.

Ako se upotrebljava verzija Worda starija od Word 6.0, onda se mora koristiti DDE za nadzor Word datoteke. U novijim verzijama, bolje je koristiti OLE automatizaciju.

Primjer DDE komunikacije koji radi potpuno isti postupak kao i primjer *Excelmakro()* napisan za OLE komunikaciju, tj. otvara novu datoteku u Wordu, daje joj naslov *"Izvještaj"*, a zatim formatira naslov masno otisnutim pismom, veličine 32 i centrira naslov vodoravno na stranici:

```
Sub KomunikacijaDDE()
Dim DDE1 As Integer
With Application
        .ActivateMicrosoftApp xlMicrosoftWord
        DDE1 = .DDEInitiate("WinWord", "Document1")
        .DDEExecute DDE1, "[AppMaximaze 1]"
        .DDEExecute DDE1, "[InsertPara]"
        .DDEExecute DDE1, "[InsertPara]"
        .DDEExecute DDE1, "[Insert " " Izvještaj" "]"
        .DDEExecute DDE1, "[StartOfLine]"
```

```
        .DDEExecute DDE1, "[EndOfLine 1]"
        .DDEExecute DDE1, "[CenterPara]"
        .DDEExecute DDE1, "[Bold]"
        .DDEExecute DDE1, "[FontSize 32]"
        .DDEExecute DDE1, "[EndOfLine]"
        .DDEExecute DDE1, "[InsertPara]"
        .DDETerminate DDE1
End With
End Sub
```

2.3.3 Funkcije za rad sa datotekama i direktorijima

VBA sadrži nekoliko svojstava, metoda i funkcija koje se koriste za izradu i brisanje datoteka i direktorija, za izmjenu i određivanje putanja te za upisivanje i čitanje iz datoteka na tvrdom disku: *AltStartupPath, ChDir, ChDrive, CurDir, DefaultFilePath, Dir, Kill, LibraryPath, MkDir, PathSeparator, RmDir, SetAttr, StartupPath*

Primjer zapisivanja podataka u tekstualnu datoteku pomoću makro programa *NapišiuDatoteku* napisanog u VBA:

```
Sub NapišiuDatoteku()
Dim Broj As Integer
        Broj = FreeFile()
        Open "C:\Test.txt" For Output As #Broj
        Print # Broj, "Ovo je primjer." &Chr(14)
        Write # Broj, 10*10
        Close # Broj
End Sub
```

Makro *PročitajizDatoteke* služi za učitavanje podataka iz tekstualne datoteke i njihov prikaz u okviru za dijalog:

```
Sub PročitajizDatoteke()
Dim x As String
Dim St As String
Dim Broj As Integer
        Broj = FreeFile()
        Open "C:\Test.txt" For Input Access Read As # Broj
        Do While Not EOF(Broj)
                Line Input # Broj, x
                St = St & x
        Loop
        Close # Broj
        MsgBox St
EndSub
```

2.3.4 Biblioteke za dinamičko povezivanje (DLL)

Biblioteke za dinamičko povezivanje (*Dynamic Link Libraries* – DLL) su jedan od najvažnijih strukturnih elemenata Windows operativnog sustava. DLL datoteke nisu izravno izvodive i ne primaju poruke. To su posebne datoteke s funkcijama koje se pozivaju iz programa ili drugih DLL-ova zbog obavljanja određenog posla. Iako biblioteke za dinamičko povezivanje mogu imati bilo koji nastavak imena, kao što su .EXE ili .FON, njihov standardni nastavak je .DLL. Jedino se .DLL automatski čita od strane operativnog sustava. Ako datoteka ima drugačiji nastavak, učitavanje modula se mora eksplicitno navesti upotrebom funkcija *LoadLibrary* ili *LoadLibraryEx*.

Biblioteke za dinamičko povezivanje se upotrebljavaju u toku izvođenja programa. Prilikom njihovog učitavanja, DLL se mora nalaziti u jednom od direktorija:

- direktoriju s .EXE programom,
- trenutno aktivnom direktoriju,
- sistemskom direktoriju Windowsa,
- Windows direktoriju,
- direktoriju koji se nalazi na navedenoj putanji.

2.3.4.1 Upotreba Declare naredbe

Prije pozivanja funkcija iz DLL-a u VBA, upotrebljava se naredba *Declare* koja identificira funkciju koja se koristiti, ime DLL-a u kojem se funkcija nalazi, i njene tipove argumenata. Nakon što se funkcija deklarira u VBA modulu, poziva se kao da je dio koda.
Funkcija se deklarira na sljedeći način:

[Public | Private] Declare Sub *ime* Lib "*imebiblioteke*" [Alias "*drugoime*"] **(**[*lista_argumenata*]**)**]
ili
[Public | Private] Declare Function *ime* Lib "*imebiblioteke*" [Alias "*drugoime*"] **(**[*lista_argumenata*]**)**] [As *tip*]

Dijelovi koji su navedeni u uglatim zagradama smiju se izostaviti. Funkcija ili procedura može biti *Public* (može se pozvati iz bilo kojeg VBA modula) ili *Private* (može se pozvati samo iz VBA modula gdje je deklarirana).
'*ime'* je ime procedure ili funkcije iz DLL-a. Mada VBA ne radi razliku između velikih i malih slova, kod deklariranja DLL-a ih razlikuje, jer poziva DLL napisan u programskom jeziku (C++) koji razlikuje velika i mala slova.
'*imebiblioteke''* je ime DLL-a u kojem se nalazi procedura ili funkcija.
Alias '*drugoime'* upotrebljava se ako se ime procedure ili funkcije poklapa sa imenom neke varijable, konstante ili neke druge procedure u istom VBA modulu. Upotrebljava se također i ako ime procedure sadrži neki znak koji ne zadovoljava DLL konvenciju pisanja imena.

'lista_argumenata' je lista argumenata koja se predaje proceduri ili funkciji i mora se navesti točan tip varijable koja se predaje. Ako se poziva funkcija, poželjno je navesti i tip koji funkcija vraća - As *tip* – kako bi se vidjelo je li ispravno pridružena varijabli.

VBA učitava DLL pri prvom pozivu funkcije iz DLL-a i on ostaje u memoriji dok se ne zatvori radna bilježnica u kojoj se nalazi taj VBA modul.

2.3.4.2 Predaja argumenata po vrijednosti i po referenci

VBA kod inicijalno predaje argumente preko referenci, a ne preko njihovih vrijednosti. Ako funkcija očekuje pokazivač tada se argument predaje po referenci (*ByRef*), inače se predaje po vrijednosti (*ByVal*).

a) Primjer predaje argumenta *ByVal* – sljedeća funkcija je samo dio DLL-a i napisana je u programskom jeziku C++, a računa opseg kruga za zadani radijus:

```
double WINAPI opseg(double dRadius)
{
        return dRadius * 2 * 3.14159;
}
```

U Windowsima, DLL funkcije koriste *__stdcall* konvenciju pozivanja funkcije. U ovom primjeru je upotrijebljena konstanta *WINAPI* koja u sebi već sadrži konvenciju *__stdcall*.

Visual Basic kod koji koristi funkciju *opseg* za ispis u Excel tablici vrijednosti radijusa i njegovog odgovarajućeg opsega:

```
Declare Function opseg Lib "primjer" _
   (ByVal radius As Double) As Double

Sub TablicaOpsega( )
   Dim x As Double
   Worksheets(1).Activate
   Range("a1:c11").Clear
    Cells(1, 1).Value = "Radius"
   Cells(1, 2).Value = "Opseg Kruga"
   For i = 1 To 10
      Cells(i + 1, 1).Value = i
      Cells(i + 1, 2).Value = opseg(i)
   Next
   Columns("a:b").AutoFit
End Sub
```

Declare naredba koristi *ByVal* ključnu riječ, jer se argument prenosi po vrijednosti.

b) Primjer prenošenja argumenata po referenci:

Funkcija *Pomnoži* je dio DLL-a i mijenja argument koji primi tako da ga pomnoži s brojem 2 ako je argument>0. Funkcija vraća vrijednost *False* ako je argument manji od nule, inače vraća vrijednost *True*:

```
bool WINAPI Pomnoži(short *pn)
{
        if (*pn < 0)
        return 0;           // FALSE u Visual Basic

        *pn *= 2;
        return -1;                  // TRUE u Visual Basic
}
```

VBA deklaracija funkcije sada više ne uključuje *ByVal* ključnu riječ nego *ByRef*. VBA inicijalno predaje argumente po referenci, pa ključnu riječ *ByRef* nije ni potrebno navesti.

```
Declare Function Pomnoži Lib "primjer" _
   (d As Integer) As Boolean

Sub PomnožiBroj()
   Dim n As Integer
   n = CInt(InputBox("Unesite broj?"))
   r = Pomnoži(n)
   MsgBox n & ":" & r
End Sub
```

2.3.4.3 Korištenje tipa Variant

Prilikom predaje Variant tipa argumenta postupak je vrlo sličan predaji argumenta bilo kojeg drugog tipa. Ipak ovaj postupak naglašen je prije svega zbog specifičnog korištenja Variant tipa u DLL-u (programski jezik C++). U DLL-u se koristi struktura podataka VARIANT pomoću koje je omogućen pristup podacima koji su sadržani u argumentu. Npr., sljedeća funkcija određuje o kojem se tipu argumenta radi (sadržanom u Variant strukturi):

```
short WINAPI VariantPrimjer(VARIANT vt)
{
        if (vt.vt == VT_DISPATCH)                   // variant je  objekt
                return -1;
        else if (vt.vt == VT_BSTR)                  // variant je string
                return _wtoi(vt.bstrVal);
        else if (vt.vt == VT_I2)                // variant je integer
                return vt.iVal;
        else
```

```
            // variant je nešto drugo
        return -3;
}
```

U VBA kodu deklarirana je i kasnije pozvana *VariantPrimjer* funkcija:

```
Declare Function VariantPrimjer Lib "primjer" _
   (ByVal v As Variant) As Integer

Sub VariantTest()
        MsgBox VariantPrimjer (Worksheets(1))        '-1 (objekt)
        MsgBox VariantPrimjer ("25")                 '25 (vrijednost)
        MsgBox VariantPrimjer (5)                    '5 (vrijednost)
        MsgBox VariantPrimjer (3.2)                  '-3 (nešto drugo)
End Sub
```

Oznake variant konstanti za određeni Variant tip argumenta koje podržava VBA kod.

Boolean	VT_BOOL
Currency(scaled integer)	VT_CY
Date	VT_DATE
Double(double floating-point)	VT_R8
Integer	VT_I2
Long Integer	VT_I4
Object	VT_DISPATCH
Single(floating-point)	VT_R4
String	VT_BSTR

2.3.4.4 Korištenje tipa String

Kada VBA predaje string po referenci u DLL, tada upotrebljava poseban OLE tip podataka nazvan BSTR. U većini slučajeva BSTR se može tretirati kao pokazivač na string koji završava s nul - bajtom (\0).
Funkcija u C-u treba deklarirati argument kao pokazivač na BSTR. Pokazivač neće nikada biti NULL. Ako VBA stringu nije ništa pridruženo, BSTR na koji pokazuje pokazivač će biti nula. Ako je pridružen prazan string, prvi karakter će biti 0 i dužina stringa će biti 0.

```
short WINAPI STip(BSTR *pbstr)
{       if (pbstr == NULL)              // pokazivač je nula, neće se dogoditi
                return 1;
        if (*pbstr == NULL)             // string  je alociran u VB  sa Dim naredbom,ali mu
                return 2;                       //još  nije pridružena vrijednost
        if (*pbstr[0] == 0)         // string je alociran i pridružen mu je prazan string (" ")
                return 3;
        return 4;                       // string ima vrijednost
}
```

Primjer VBA koda koji deklarira i poziva *STip* funkciju:

```
Declare Function STip Lib "primjer" _
   (s As String) As Integer

Sub STipTest()
   Dim s As String
   MsgBox STip(s)                    'ispisuje 2
   s = ""
   MsgBox STip(s)                    ' ispisuje 3
   s = "primjer"
   MsgBox STip(s)                    ' ispisuje 4
End Sub
```

2.3.4.5 Korištenje korisnički definiranih struktura

Naredba *Type* koristi se u VBA kodu za kreiranje korisnički definiranih struktura. Na primjer, sljedeći VBA tip podatka i struktura u C-u su identični.

U VBA:

```
Type ARG
   i As Integer
   str As String
End Type
```

U C-u:

```
typedef struct {
        short i;
        BSTR bstr;
} ARG;
```

Korisnički tipovi podataka na smiju biti predani po vrijednosti, nego po referenci. Primjer funkcije u C-u koja deklarira argument kao pokazivač na strukturu:

```
short WINAPI ArgStruktura(ARG *parg, char *szArg)
{       BSTR bstr;
        if (parg == NULL)
                return -1;
        if ((bstr = SysAllocString((BSTR)szArg)) == NULL)
                return -1;                              // alocira lokalni string
        if (parg->bstr != NULL)   // stringu je već pridružena vrijednost
                SysFreeString(parg->bstr);
        parg->i = SysStringByteLen(bstr);          //dužina stringa
        parg->bstr = bstr;
        return parg->i;
}
```

Deklaracija i poziv iz VBA:

```
Declare Function ArgStruktura Lib "primjer" _
   (a As ARG, ByVal s As String) As Integer

Sub ArgStrukturaTest( )
   Dim x As ARG
   MsgBox ArgStruktura (x, "abracadabra")
   MsgBox x.str & ":" &str&( x.i)                    'ispisuje string i dužinu
End Sub
```

*2.3.4.6 Korištenje polja (*array*)*

OLE automatizacija osigurava poseban tip podataka polja koji se prenosi iz VBA u DLL. Ovaj tip podataka naziva se SAFEARRAY. Kada OLE automatizacija predaje SAFEARRAY u DLL, tada DLL prima pokazivač na pokazivač polja. Kao i BSTR pokazivači, i SAFEARRAY pokazivači mogu pokazivati na nulto polje, ako je polje alocirano, ali još nije dimenzionirano.
Pomoću SAFEARRAY tipa podataka predaje se polje DLL-u, koji onda koristeći OLE funkcije za rad sa SAFEARRAY–om očita dimenziju polja, njegovu gornju i donju granicu, te elemente tog polja.

Primjer funkcije *SumaPolja* koja je dio DLL-a i prima dva parametra. Prvi parametar je pokazivač na SAFEARRAY, a drugi parametar je pokazivač na long. Funkcija provjerava broj dimenzija polja, očitava gornju i donju granicu polja, te zbraja elemente polja:

```
typedef SAFEARRAY * FPSAFEARRAY;

short WINAPI SumaPolja(FPSAFEARRAY *ppsa, long *plResult)
{       short Elem;
        long donjagr, gornjagr, l, rezultat;

        if (*ppsa == NULL)                  // polje još nije inicijalizirano
                return -4;
        if ((*ppsa)->cDims != 1)            // provjeri broj dimenzija
                return -5;
                        // očitavanje gornje i donje granice polja
        if (FAILED(SafeArrayGetLBound(*ppsa, 1, &donjagr)) ||
                FAILED(SafeArrayGetUBound(*ppsa, 1, &gornjagr)))
                return -1;

                                //zbrajanje elemenata
        for (l =donjagr, rezultat = 0; l <=gornjagr; l++)
        {
                if (FAILED(SafeArrayGetElement(*ppsa, &l, &Elem)))
                        return -2;
```

```
            rezultat += Elem;
        }
        *plResult = rezultat;
        return 0;
}
```

Primjer deklaracije i poziva funkcije *Sumapolja* iz VBA:

```
Declare Function Sumapolja Lib "primjer" _
   (a( ) As Integer, r As Long) As Integer

Sub SumaPoljaTest()
   Dim n(5) As Integer
   Dim suma As Long

                'Vrijednost  prvih pet ćelija u Excelu spremi  u polje
   For i = 0 To 4
      n(i) = Cells(i+1)
   Next
   x = SumArray(n, suma)          'Poziv funkcije SumArray
   MsgBox x & ":" & suma
End Sub
```

VBA radi minimalnu provjeru tipa i veličine polja. Zbog toga je vrlo važno uočiti kako je deklarirana funkcija, jer pozivom DLL-a sa krivim tipom podatka polja, najvjerojatnije će doći do rušenja aplikacije (Excela).

2.3.5 Poziv funkcije iz Excel radne bilježnice

Funkcije iz DLL-a se mogu pozvati i direktno iz radne bilježnice, pomoću funkcije CALL.
Sintaksa CALL naredbe je sljedeća:

CALL(modul_tekst,procedura,tip_tekst,argument1,argument2...)

modul_text je ime DLL-a, koji se mora nalaziti na putanji traženja DLL-a
procedura – ime funkcije u DLL-u,
tip_tekst – tipovi argumenata koje funkcija vraća i koje prima
argument1,argument2... – vrijednosti argumenata koje funkcija prima

U trećem argumentu *tip_tekst* u *CALL* funkciji prvo slovo predstavlja tip podatka koji funkcija vraća, a sljedeća slova predstavljaju redom sve tipove argumenata koje funkcija prima.

Tipovi podataka koji se najčešće koriste imaju sljedeće oznake:

Tip	Opis	C deklaracija
A	logical	short Int
B	8 byte	double
C	string	char*
D	string	unsigned char*
E	8 byte	double*
I	2 byte	short int
J	4 byte	long int
K	array	FP*

Primjer – kreiran je DLL *mnozenje* koji sadrži funkciju *funk* koja množi predanu vrijednost sa 2:

```
int WINAPI funk(int x)
{
        return x*2;
}
```

U Excelu je pozvana funkcija *funk* tako da je u ćeliju upisana naredba:

= CALL ("mnozenje", "funk", "JJ", 10)

a kao rješenje funkcije će se u toj ćeliji ispisati broj 20.

3. PROGRAMSKI JEZIK C / C++

C++ je programski jezik koji je izgrađen na osnovama C programskog jezika i opisan je u mnogo slučajeva kao C sa klasama, tj. C++ podržava objektno orijentirano programiranje.

3.1 Općenito o C/C++

Popularnost C/C++ danas je sasvim očigledna. Većina novih aplikacijskih programa napisana je u C/C++, a to se odnosi i na većinu operacijskih sustava. Razlog ovakve rasprostranjenosti je da, pored toga što se smatra višim programskim jezikom, C/C++ je dovoljno niskog nivoa da posjeduje mogućnosti rezervirane samo za asembler (direktan pristup memoriji i manipuliranje bitovima). Zbog ovih svojstava pogodan je za sistemsko programiranje i pisanje operacijskih sustava. Pored toga omogućuje pisanje racionalnog koda koji je vrlo brz uz odgovarajuće optimizacije kao i programa prenosivih između različitih računala, pa se primjenjuje za pisanje aplikacijskih programa.

3.1.1 main( ) ili WinMain( ) funkcija

Svaki C++ program mora u sebi imati *main()* ili *WinMain()* funkciju. Te funkcije služe kao ulazne točke programa, tj. od te točke se program započinje izvršavati, a isto tako one određuju i završnu točku programa.

main() funkcija je funkcija koja se koristi u programskom jeziku C, ali ona se pojavljuje i u C++ ako se program izvršava kao konzolna aplikacija.
Prilikom kreiranja konzolne aplikacije Visual C++ inicijalno napiše *main()* funkciju koja prima dva parametra, a vraća integer vrijednost.

```
int main (int argc, char* argv[ ])
{
        return 0;
}
```

argc – je integer koji pokazuje koliko argumenata je predano programu iz komandne linije. Pošto se ime programa smatra za argument, *argc* je minimalno broj 1,

argv –pokazuje na polje znakovnih nizova (stringova) koji sadrže argumente. Prvi string *argv[0]* je ime programa, i svaki sljedeći je argument predan programu iz komandne linije. Posljednji pokazivač *argv[argc]* je NULL.

U većini programa vrijednost koju vraća *main()* nije bitna, pa se tada može koristiti jedan od sljedećih oblika *main()* funkcije:

```
main()
int main()
int main(void)
void main()
void main(int argc, char *argv[ ])
```

U Windows (Win32) aplikaciji se umjesto *main()* funkcije nalazi *WinMain* funkcija koja također definira početnu i krajnju točku programa:

```
int WINAPI WinMain (HINSTANCE hInstance,
        HINSTANCE hPrevInstance,
        LPSTR     lpCmdLine,
        int       nCmdShow)
{
        //Ovdje treba napisati kod.
        return 0;
}
```

Funkcija također vraća integer vrijednost, ali sada prima 4 parametra. Funkciji *WinMain* dodijeljen je tip WINAPI. Taj identifikator definiran je kao *__stdcall*, što se odnosi na specijalni niz pozivanja funkcija koji se odvija između operativnog sistema i aplikacije.
U Windowsima postoji nekoliko identifikatora (napisani velikim slovima) različitih tipova "rukovatelja":

- HINSTANCE - rukovatelj primjerkom istog programa,
- HWND - rukovatelj prozorom,
- HDC - rukovatelj kontekstom uređaja,
- HICON - rukovatelj ikonom,
- HCURSOR - rukovatelj mišem,
- HBRUSH - rukovatelj kistom.

Tip podataka LPSTR koji je treći parametar funkcije *WinMain* je pokazivač na niz znakova (string) i pokazivač je tipa long.

Argumenti funkcije *Winain*:

- *hInstance* – rukovatelj na trenutnu instancu aplikacije je broj koji jednoznačno označava program za vrijeme njegova izvođenja u Windowsima.
- *hPrevInstance* – rukovatelj na prijašnju instancu aplikacije. U Win32 aplikacijama ovaj parametar uvijek iznosi NULL.
- *lpCmdLine* – pokazivač na string koji određuje komandnu liniju aplikacije i ne uključuje ima programa. Da bi funkcija vratila cijelu komandnu liniju treba upotrijebiti funkciju *GetComandLine*.
- *nCmdShow* – pokazuje način prikazivanja prozora aplikacije u okružju Windowsa (npr. *SW_HIDE, SW_MINIMAZE, SW_SHOWNORMAL*….)

3.1.2 Mađarska notacija

Mnogi programeri koji programiraju za Windowse koriste dogovor o imenovanju varijabli poznat kao mađarska notacija. Ime varijabli započinje malim slovom ili slovima koja označavaju tip podataka kojem varijabla pripada. Na primjer, u imenu *lpCmdLine* prefiks *lp* dolazi od long pointer.

Prefiksi koji se često koriste u Windows programiranju su:

c	char
by	BYTE
n	short
i	int
x, y	int (korišten kao x ili y – koordinata)
b ili f	Bool(int), f dolazi od "flag" ili zastavica
w	word(unsigned short)
l	long
fn	function
s	string
sz	string koji završava nul – bajtom
h	handle
p	pointer

3.1.3 C++ klase i objektno orijentirano programiranje

Klasa je skup podataka i funkcija koje zajedno obrađuju posebnu programersku zadaću. Svaka klasa mora imati deklaraciju. Definicija i deklaracija klase se obično piše u tzv. *header fileu* ili zaglavlju zbog preglednosti koda. Ako se deklaracija navede u zaglavlju, tada se to zaglavlje uključi u kod pomoću naredbe *include*, npr.

```
#include "primjer1.h"
```

U jednostavnim slučajevima se i definicija i deklaracija klase navodi u samom C++ kodu.

Klase imaju tri nivoa pristupa: *private*, *public* i *protected*. Programer nema pristup funkcijama i varijablama iz klase koje su navedene kao *private*, već se njima koristi interno sama klasa. Ako je nivo pristupa *public* tada programer ima potpun pristup tim članovima klase. *Protected* varijablama i funkcijama klase također ne može pristupiti korisnik klase, ali se njima može koristiti klasa koja je izvedena od te iste klase (nasljeđuje klasu i dodaje joj još neke nove funkcije)
Inicijalno je nivo pristupa klasi *private*, pa ako nije naglašeno da su neki članovi klase *public* ili *protected* tada se toj klasi ne može pristupiti.

3.1.3.1 Konstruktor

Konstruktor (*constructor*) je funkcija –član koja se automatski poziva prilikom kreiranja instanci klase, tj. objekta. Koristi se za inicijalizaciju varijabli klase i za rezerviranje memorijskog prostora korištenjem operatora *new*.
Konstruktor treba imati isto ime kao i klasa. Klasa može imati više od jednog konstruktora (kod *overloaded* funkcija – funkcije koje imaju isto ime, ali primaju različite tipove podataka).
Klasa *Pravokutnik* sadrži dva konstruktora; jedan konstruktor postavlja sve članove na 0, a drugi konstruktor omogućava postavljanje članova klase na željene vrijednosti:

```
class Pravokutnik {
public:
        Pravokutnik ();                                 //konstruktor 1
        Pravokutnik (int _lijevo, int _vrh, int _desno, int _dno);          //konstruktor 2
        int UzmiŠirinu();
        int UzmiVisinu();
        void PostaviPravokutnik(int _lijevo, int _vrh, int _desno, int _dno);
private:
        int lijevo;
        int vrh;
        int desno;
        int dno;
};
```

Definicija konstruktora treba izgledati ovako:

```
Pravokutnik:: Pravokutnik ()
{
        lijevo = 0;
        vrh = 0;
        desno = 0;
        dno = 0;
}

Pravokutnik :: Pravokutnik (int _lijevo, int _vrh, int _desno, int _dno);
{
        lijevo = _lijevo;
        vrh = _vrh;
        desno =_desno;
        dno = _dno;
}
```

Prvi konstruktor je inicijalni konstruktor koji ne prima parametre. Drugi konstruktor prima parametre i pridružuje ih odgovarajućim članovima klase. Imena varijabli su lokalna za konstruktor i svako ime varijable počinje sa _ kako bi se razlikovale lokalne varijable od članova klase.

Konstruktor se upotrebljava za kreiranje instanci klase, odnosno za kreiranje objekta:

```
Pravokutnik pravokutnik1;                  //objekt kreiran pomoću prvog konstruktora
Pravokutnik pravokutnik2 (0,0,100,100);    //objekt kreiran pomoću drugog konstruktora
```

3.1.3.2 Destruktor

Destruktor je posebna funkcija koja se poziva prije uništavanja objekta. Destruktor je suprotan od konstruktora. Koristi se za oslobađanje memorijskog prostora koju je rezervirao konstruktor klase. Ime destruktora mora biti isto kao i ime klase, a ispred imena destruktora nalazi se znak ~.

```
class Pravokutnik {
public:
        Pravokutnik ();                    //konstruktor 1
        Pravokutnik (int _lijevo, int _vrh, int _desno, int _dno);      //konstruktor 2
        ~Pravokutnik();            // destruktor
        int UzmiŠirinu();
        int UzmiVisinu();
        void PostaviPravokutnik(int _lijevo, int _vrh, int _desno, int _dno);
private:
        int lijevo;
        int vrh;
        int desno;
        int dno;
        char *tekst;               //dodan novi član klase
};
```

Definicija konstruktora:

```
Pravokutnik:: Pravokutnik ()
{
        lijevo = 0;
        vrh = 0;
        desno = 0;
        dno = 0;
        tekst = new char[256];             //rezerviranje memorijskog prostora
        strcpy (tekst, "Bilo koja boja");
}
```

Primjer pozivanja destruktora:

```
Pravokutnik::~Pravokutnik()
{
        delete[ ] tekst;                   //brisanje teksta iz memorije
}
```

Pravokutnik klasa rezervira memorijski prostor za polje znakova nazvano *tekst* s konstruktorom, a destruktor oslobađa taj memorijski prostor prije nego što se objekt uništi, tj. briše element iz memorije.

3.1.3.3 Pokazivač "this"

Sve klase sadrže skriveni član koji se zove *this*. *This* je pokazivač na instancu klase u memoriji. Svaka instanca klase ima svoju vlastitu kopiju članova klase, ali sve instance klase dijele isti skup funkcija. Kompajler na osnovu parametra *this*, određuje koji objekt klase ide sa kojim pozivom funkcije.

Primjer načina upotrebe operatora **this*:

```
void Datum :: postaviMjesec (int mj)
{
  mjesec = mj;
  this -> mjesec = mj;
  (*this).mjesec = mj;
}
```

Ova tri izraza su ekvivalentna. Zabranjeno je mijenjati vrijednost pokazivača *this*, on uvijek treba pokazivati na instancu klase, tj. na objekt.

3.1.4 Kompajler i linker

Nakon što je napisan izvorni (*source*) kod i zaglavlje (*header file*), taj kod se treba prevesti, povezati i izvršiti.
Naredba *Compile* pokreće preprocesor koji kreira ulazne podatke za kompajler, koji prevodi izvorni kod u objektni oblik i dodjeljuje mu ekstenziju .OBJ. Taj oblik sadrži mašinski kod, upute za linker, te imena funkcija i varijabli kreiranih iz izvornog koda.
U sljedećem koraku linker povezuje .OBJ file sa statičkim bibliotekama (uključene u kod pomoću *#include*) i drugim objektnim modulima, i kreira izvršnu verziju programa (.EXE).

Opcije preprocesora, kompajlera i linkera mogu se promijeniti u izborniku *Project–Settings* u Visual C++.

Error! Not a valid link.

Sl. 3.1.1 Kreiranje .EXE programa

3.2 Biblioteke za dinamičko povezivanje (DLL)

Biblioteka za dinamičko povezivanje (*Dynamic Link Library* – DLL) je kod smješten u datoteci sa .DLL ekstenzijom. DLL nije izravno izvodiv pa se poziva ili iz izvršnog programa ili iz drugog DLL-a.

Postoje dva tipa DLL-a:

- DLL koji sadrži kod
- DLL koji sadrži samo resurse.

Kod koji se nalazi u DLL-u napisan je ili u obliku funkcija ili u obliku C++ klasa. Prije pozivanja funkcija i korištenja klasa koje su dio DLL-a iz neke aplikacije, DLL treba biti učitan u memoriju. Tada se te funkcije i klase ponašaju kao da su dio same aplikacije.

3.2.1 Načini učitavanja DLL u memoriju

DLL se može učitati u memoriju na dva načina: statički i dinamički.

Statičko učitavanje: prilikom pokretanja aplikacije koja poziva DLL, DLL se automatski učita u memoriju. Prednost ovakvo načina pozivanja DLL-a je njegova jednostavnost, a nedostatak je u tome da ako program ne uspije ispravno pozvati DLL, program se neće moći ni pokrenuti.

Dinamičko učitavanje: DLL se učitava u memoriju tek kad aplikacija zatreba neku njegovu klasu ili funkciju, a kada prestane njegova uporaba DLL se briše iz memorije. Prednost ovakvog načina je bolje iskorištenje memorijskog prostora, pa se zbog toga aplikacija brže pokreće, a glavni nedostatak ovakvog postupka je dodatan posao za programera (učitavanje DLL-a sa *LoadLibrary* i oslobađanje memorije nakon njegove upotrebe sa *FreeLibrary*).

3.2.2 Kreiranje DLL-a

Postoji nekoliko vrsta DLL-ova, tzv. *regular DLL* , *extension DLL*, *non-MFC (Microsoft Foundation Class) DLL* i *resource DLL*. Ovisno o tome na koji način i gdje će se DLL koristiti kreira se jedan od njih.

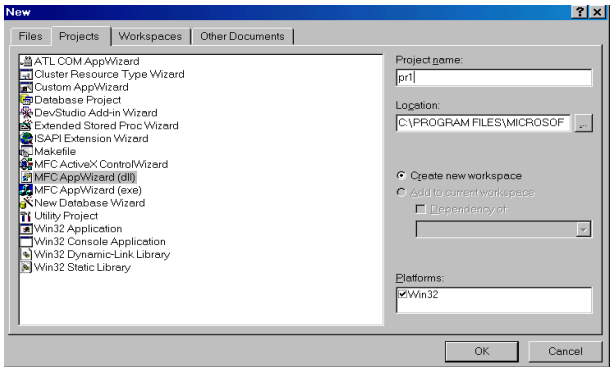
Regular DLL može biti statički i dinamički povezan s MFC (*Microsoft Foundation Class*). MFC koristi interno, a funkcije koje se eksportiraju mogu se pozvati ili sa MFC izvršnim programima, ili sa izvršnim programima koji ne koriste MFC. Izvršni program koji poziva DLL može biti napisan u bilo kojem programskom jeziku koji podržava korištenje DLL-a (C, C++, Pascal, Visual Basic...).

MFC extension DLL je DLL koji implementira klase nastale od postojećih MFC klasa. Izvršna aplikacija mora biti MFC aplikacija ili *regular DLL* dinamički povezan sa MFC. *Extension DLL* mora imati *DllMain* funkciju i sva potrebna inicijalizacija se mora nalaziti u njoj.

Non-MFC DLL je DLL koji ne koristi interno MFC, a funkcije koje se eksportiraju mogu se pozivati od MFC i ne-MFC izvršnih programa. Funkcije se obično eksportiraju iz DLL-a koristeći standardno C sučelje.

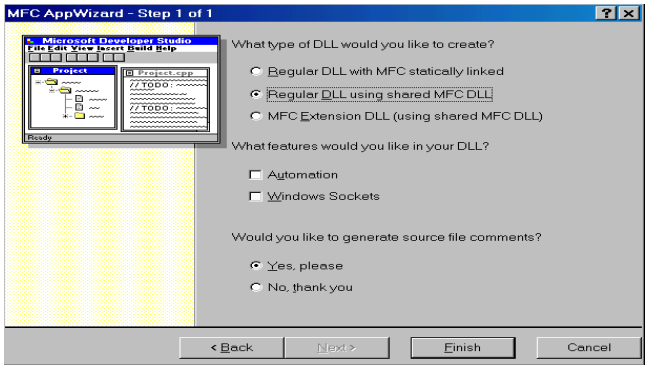
Resource DLL je DLL koji sadrži samo resurse, kao što su ikone, bitmape, stringove i okvire za dijalog.

U Visual C++ kreiranje DLL-a započinje odabirom opcije *MFC AppWizard* (kreira *regular DLL* i *extension DLL*), ili *Win32DynamicLinkLibrary* (kreira *non-MFC DLL* i *resource DLL*) koje se nalaze u izborniku *File – New*.



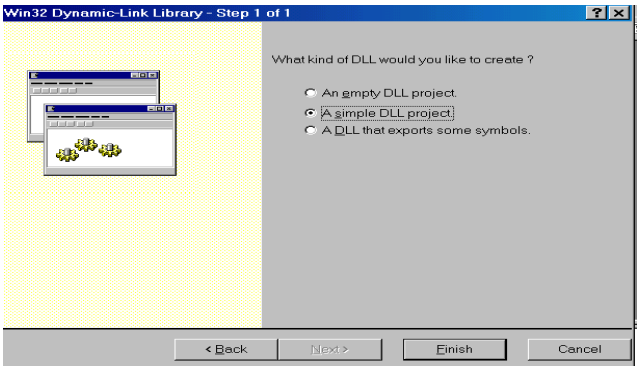
Sl. 3.2.1 Okvir za dijalog New Projects

MFC AppWizard u sljedećem koraku pruža mogućnost izbora DLL-a (prikazano na Sl. 3.2.2). Na osnovu odabranog DLL-a, *AppWizard* kreira tipične datoteke sa osnovnim inicijalizacijskim postupkom.



Sl. 3.2.2 MFC AppWizard

Izborom *Win32 Dynamic – Link Library* kreira se ili *non-MFC DLL* ili *resource DLL*. Ovisno o odabiru u sljedećem koraku (Sl. 3.2.3), DLL projekt kojeg kreira *Win32 Dynamic – Link Library* može biti prazan DLL (bez ikakvog napisanog koda) ili DLL koji sadržava potreban kod za njegovu inicijalizaciju.



Sl. 3.2.3 Win32 DLL

Resource DLL se kreira tako da se u izborniku *Project*, opciji *Add To Project* kreira novi *Resource Script* za DLL snimljen kao datoteka s ekstenzijom .RC. Da bi DLL sadržavao samo resurse, potrebno je u izborniku *Project–Settings*, u podizborniku *Link* dodati */NOENTRY* opciju.

3.2.3 Eksportiranje iz DLL-a korištenjem modulski definirane datoteke (.DEF)

Modulski definirana datoteka (.DEF) je tekstualna datoteka koja sadrži jednu ili više naredbi koje opisuju različite atribute DLL-a. Prva naredba mora biti *LIBRARY* naredba. Ona identificira pripadanje DEF datoteke DLL-u i linker smješta njegovo ime u DLL-ovu ulaznu biblioteku.
Naredba *EXPORTS* ispisuje imena funkcija i njihove ordinalne vrijednosti koje eksportira DLL tako da iza imena funkcije piše znak @ i njen ordinalni broj. Ordinalne vrijednosti moraju biti u rasponu od 1 do N, gdje je N ukupan broj funkcija koje DLL eksportira.
Mada nije nužno, .DEF datoteka često sadrži i naredbu opisa *DESCRIPTION*, koja opisuje svrhu DLL-a.

Na primjer jedna datoteka sa ekstenzijom .DEF izgleda ovako:

```
LIBRARY BINSTABLO
DESCRIPTION " Kreira binarno stablo"
EXPORTS
        Unesi @1
        Izbriši @2
        Član @3
        Min @4
```

AppWizard kreira osnovnu DEF datoteku i automatski je dodaje DLL projektu. U tako kreiranu datoteku dodaju se samo imena funkcija s njihovim ordinalnim brojevima.
Za razliku od *AppWizarda*, opcija *Win32 DLL* ne kreira DEF datoteku, ali se datoteka može naknadno kreirati i dodati projektu (*Add To Project*).

Prilikom eksportiranja funkcija iz DLL-a u izvršne datoteke napisane u Visual C++, C++ kompajler automatski kreira tzv. dekorativna imena funkcija koja se moraju navesti u .DEF datoteci umjesto stvarnih imena funkcija, ili se funkcije moraju eksportirati sa standardnom C vezom (naredba *extern "C"*).
Dekorativna imena funkcija mogu se pročitati upotrebom naredbe *DUMPIN* ili korištenjem linker naredbe */MAP* (naredbe se navode u izborniku *Project–Settings*). Aplikacija koja poziva DLL (C++ kod) mora biti kreirana sa istom verzijom Visual C++ da bi mogla pročitati dekorativna imena.

Ako *extension DLL* sadržava klase koje se moraju eksportirati, tada se na početku i na kraju zaglavlja mora nalaziti sljedeći kod:

```
#undef AFX_DATA
#define AFX_DATA AFX_EXT_DATA
        // tijelo zaglavlja
#undef AFX_DATA
#define AFX_DATA
```

Ove linije koda osiguravaju da su MFC varijable, koje DLL koristi interno, dodane klasama i eksportirane sa DLL-om.

3.2.4 Eksportiranje funkcija iz DLL-a korištenjem ključne riječi __declspec(dllexport)

Pomoću ključne riječi *__declspec(dllexport)* može se eksportirati podatke, funkcije i klase iz DLL-a. Ako se upotrebljava naredba *__declspec(dllexport)* tada nije potrebna .DEF datoteka.
Da bi eksportirali funkcije, *__declspec(dllexport)* se postavlja lijevo od ključne riječi koja predstavlja konvenciju poziva DLL-a (*__stdcall* ili *WINAPI*), ako je ključna riječ navedena:

```
void __declspec(dllexport) __stdcall Funkcija1 (void);
```

Da bi eksportirali sve *public* podatke i funkcije klase, ključna riječ mora se napisati lijevo od imena klase:

```
Class __declspec(dllexport) CPrimjer :public CObjekt
{  definicija klase  };
```

Da bi DLL bio čitljiviji i pregledniji, ponekad se definira makro za naredbu *__declspec(dllexport)* i tada se taj makro koristi sa svakim simbolom koji se eksportira:

```
#define DllExport __declspec(dllexport)
```

3.2.5 Eksportiranje iz DLL-a korištenjem AFX_EXT_CLASS

Extension DLL koristi makro AFX_EXT_CLASS za eksportiranje klasa, a izvršni programi koji pozivaju taj DLL koriste taj isti makro za importiranje klasa.
U zaglavlju DLL-a dodaje se AFX_EXT_CLASS ključna riječ u deklaraciji klase na slijedeći način:

```
class AFX_EXT_CLASS CmojaKlasa: public Cdokument
{    tijelo klase     };
```

Ovaj makro je definiran od MFC kao *__declspec(dllexport)* kada su definirani preprocesorski simboli _AFXDLL i _AFXEXT, a definiran je kao *__declspec(dllimport)* kada je _AFXDLL definirano, a _AFXEXT nije definirano.

Pošto je AFX_EXT_CLASS definirano kao *__declspec(dllexport)* tada se može eksportirati klasu iz DLL-a bez korištenja .DEF datoteke, i na taj način se ne moraju pisati dekorativna imena za sve simbole klase. Mada .DEF datoteka nije nužna, ta metoda je ponekad efikasnija jer se imena simbola mogu eksportirati po njihovim ordinalnim brojevima.

3.2.6 Eksportiranje C++ funkcija za upotrebu u C izvršnim programima i obratno

Funkcije iz DLL-a napisane u programskom jeziku C++, kojima se pristupa iz C programskog modula, moraju se deklarirati sa C vezom, npr.:

```
extern "C" __declspec (dllexport) int MojaFunkcija (long parametar1);
```

Na sličan način, za funkcije DLL-a napisane u programskom jeziku C, kojima se pristupa iz C ili C++ programskog modula, koristi se *__cplusplus* preprocesorski makro na način da se prvo odredi koji se programski jezik kompajlira i tada se koristi C vezu ako se funkcije pozivaju iz C++ modula:

```
#ifdef __cdplusplus
extern "C"
{                       //samo za eksportiranje C sučelja
                        //ako ga koristi C++ kod
__declspec(dllimport) void MojaFunkcija();
__declspec (dllimport) void DrugaFunkcija();
#ifdef __cplusplus
}
#endif
```

3.2.7 Inicijalizacija DLL-a

DLL sadržava inicijalizacijski kod koji se izvršava prilikom pozivanja DLL-a iz nekog izvršnog programa ili iz drugog DLL-a. Mjesto u kodu gdje se dodaje inicijalizacijski kod ovisi o vrsti DLL-a. U Win32 DLL ulazna točka funkcije je najčešće *DllMain* koja služi i za inicijalizaciju i za terminaciju DLL-a.
Prilikom kreiranja DLL-a *AppWizard* ili *Win32 DLL* u nekim slučajevima (ovisno o odabranim opcijama) napišu sav potreban kod za inicijalizaciju i terminaciju DLL-a.
Ako kod nije naveden, treba se dodati, jer mada će se DLL uspješno kreirati, neće se moći ispravno pozvati iz druge aplikacije.

Vrste DLL-a	Mjesto dodavanja koda za inicijalizaciju i terminaciju DLL-a
Regular DLL	U *CwinApp* objektu upotrebom *InitInstance* i *ExitInstance*
Extension DLL	U *DllMain* funkciji koju generira *AppWizard*
Non – MFC DLL	U funkciji *DllMain* koji smo sami kreirali

Tablica 3.2.1 Mjesto dodavanja koda za inicijalizaciju i terminaciju DLL-a

Primjer inicijalizacije i terminacije DLL-a koja se obavlja u funkciji *DllMain*:

```
extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
```

```
	UNREFERENCED_PARAMETER(lpReserved);
	if (dwReason == DLL_PROCESS_ATTACH)
	{
		TRACE0("BB.DLL Initializing!\n");
					// Extension DLL inicijalizacija
		if (!AfxInitExtensionModule(BbDLL, hInstance))
			return 0;
		new CDynLinkLibrary(BbDLL);
	}
	else if (dwReason == DLL_PROCESS_DETACH)
	{
		TRACE0("BB.DLL Terminating!\n");
				// Terminacija biblioteka prije poziva destruktora
		AfxTermExtensionModule(BbDLL);
	}
	return 1;
}
```

4. PRIMJER POVEZIVANJA EXCELA SA PROGRAMSKIM JEZIKOM C++

U programskom jeziku C/C++ programira se dosta novih programa kod kojih ponekad predstavlja veći problem napraviti sučelje za unos podataka i ispis rješenja, nego riješiti sam problem. Ako cilj programiranja nije kreiranje određenog sučelja, tada je moguće upotrijebiti neku od gotovih aplikacija za rad sa podacima.
Microsoft Excel je jedna od aplikacija koja se može koristiti kao sučelje za unos podataka i ispis rezultata proračuna, tj. za tabelarni i grafički prikaz rezultata i odgovarajućih izvještaja.

Jedna od najčešćih varijanti korištenja kombinacije Excela i C/C++ ili nekog drugog programskog jezika je slučaj da određena aplikacija zahtijeva složeni i obimni proračunski dio, a ulazni i izlazni parametri su dani u tabelarnoj i grafičkoj formi. U tom slučaju takva kombinacija ima za prednost praktički gotovo korisničko sučelje (poznato velikom broju korisnika), uz mogućnost efikasnog razvoja proračunskog dijela aplikacije i njeno brzo izvršavanje.

Među specifičnim primjerima ovakve kombinacije može se navesti upravo primjer starijih numeričkih aplikacija u elektronici (i općenito tehnici) koje su dobre sa stanovišta brzine i točnosti proračuna, ali im nedostaje odgovarajuće korisničko sučelje. U tom je slučaju opisana kombinacija sa Excelom jedno od najčešće korištenih rješenja.
Kao ilustracija načina povezivanja Excela i C++ (pomoću datoteka i pomoću DLL-a) napravljen je jednostavan primjer sortiranja podataka u matrici.

4.1 Rješavanje problema u programskom jeziku C++

Potrebno je riješiti problem sortiranja brojeva od najvećeg broja prema najmanjem i obrnuto.
Problem sortiranja riješen je pomoću metode ljusaka (*Shells Method*).

Općenito, ideja sortiranja pomoću metode ljusaka izgleda sljedeće. Npr. za sortiranje 16 brojeva, n1 do n16, potrebno je prvo grupirati brojeve u 8 grupa po 2 broja koje treba sortirati (n1, n9), (n2, n10), ..., (n8, n16). Zatim se sortira svaka od 4 grupe sa 4 elementa (n1, n5, n9, n13), ..., (n4, n8, n12, n16), pa 2 grupe od 8 elemenata (n1, n3, n5, n7, n9, n11, n13, n15) i (n2, n4, n6, n8, n10, n12, n14, n16), i na kraju se sortira cijeli niz od svih 16 brojeva.
Za sortiranje elemenata nužno je samo posljednje sortiranje, ali parcijalna sortiranja omogućavaju puno brže i efikasnije sortiranje brojeva, zbog toga što se pomoću njih brojevi efikasno pomiču na poziciju koja je bliža njihovoj krajnjoj poziciji.

Odnos između brojeva koji se sortiraju u svakoj sljedećoj ljusci (8, 4, 2, 1 u ovom primjeru) naziva se inkrement, a metoda ljusaka se ponekad naziva metoda opadajućeg inkrementa.

Provedena su mnoga istraživanja na koji način treba izabrati dobar skup inkremenata. Skup 8, 4, 2, 1 nije baš najbolji izbor.
Mnogo bolji skup inkremenata je

$(3^k - 1)/2$ ..., 40, 13, 4, 1

koji se generira pomoću formula:

$i_1 = 1$ $i_{k+1} = 3 \cdot i_k + 1$ $i_k = 1, 2, ...$

Funkcija koja sortira niz elemenata a[1...n] od manjeg prema većem broju

```
void metljusakam(long n, int a[])
{       inc=1;                  //Određivanje početnog inkremenat.
        do {
                inc *= 3;
                inc++;
        } while (inc <=n);

        do {                            // parcijalna soriranja.
                inc /= 3;
                for (i=inc+1;i<=n;i++)
                {                       // Vanjska petlja.
                        v=a[i];
                        j=i;
                        while (a[j-inc] > v)
                        {               //Unutrašnja petlja.
                                a[j]=a[j-inc];
                                j -= inc;
                                if (j <= inc) break;
                        }
                        a[j]=v;
                }
        } while (inc > 1);
}
```

Funkcija koja sortira niz elemenata a[1...n] od većeg prema manjem broju

```
void metljusakav(long n, int a[])
{
        inc=1;                  //Određivanje početnog inkrementa.
        do {
                inc *= 3;
                inc++;
        } while (inc <=n);
```

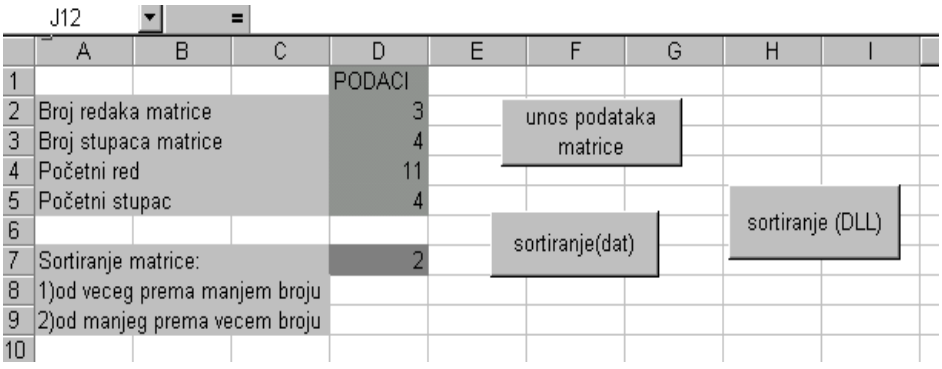
```
        do {                                    // parcijalna soriranja.
                inc /= 3;
                for (i=inc+1;i<=n;i++)
                {                               // Vanjska petlja.
                        v=a[i];
                        j=i;
                        while (a[j-inc]< v)
                        {                       //Unutrašnja petlja.
                                a[j]=a[j-inc];
                                j -= inc;
                                if (j <= inc) break;
                        }
                        a[j]=v;
                }
        } while (inc > 1);
}
```

Funkcije sortiranja *metljusakam* i *metljusakav* razlikuju se samo u tzv. unutrašnjoj petlji koja uspoređuje podatke i sortira ih od manjih prema većim ili obrnuto.

4.2 Povezivanje Excela sa C++ pomoću datoteka

Izvršna datoteka (.EXE) napisana u Visual C++ ne može direktno čitati podatke iz Excelove radne tablice, pa se zbog toga može koristiti prijenos podataka preko tekstualnih datoteka.

Podaci koji se sortiraju i način sortiranja unose se u Excel dokumentu. Podaci se unose u obliku matrice koju korisnik sam dimenzionira zadajući broj redaka i stupaca matrice, te mjesto nalaženja početnog elementa matrice, kao što je prikazano na Sl. 4.2.1.
Osim tih podataka potrebno je odabrati i način sortiranja podataka.



Sl. 4.2.1 Prikaz unosa početnih podataka matrice koja se sortira

Nakon što su navedeni osnovni podaci o matrici, potrebno je napisati i njene elemente. Unos elemenata matrice može biti napravljen ručno ili pomoću makroa *unesipodatke( )* napisanog u VBA koji u zadani prostor matrice upisuje slučajno odabrane brojeve. Makro *unesipodatke( )* pridružen je *button* objektu *"unos podataka matrice"* (Sl. 4.1.), a izgleda ovako:

```
Dim i, j, n, m, x, y As Integer
Sub unesipodatke()
        ucitaj
        Range("A10:aZ60").Clear
        For i = x To (x + n - 1)
                For j = y To (y + m - 1)
                        Cells(i, j).Interior.ColorIndex = 17
                        Cells(i, j) = Int(100 * Rnd())
                Next j
        Next i
End Sub
```

Podaci: *n* – broj redaka matrice, *m* – broj stupaca matrice, *x* – početni red matrice, *y* – početni stupac matrice su podaci koji se učitavaju pomoću potprograma *ucitaj( )*:

```
Sub ucitaj()
        Worksheets("upis podataka").Activate
        n = Cells(2, 4)
        m = Cells(3, 4)
        x = Cells(4, 4)
        y = Cells(5, 4)
End Sub
```

U sljedećem koraku se podaci iz Excel tablice smještaju u polje podataka:

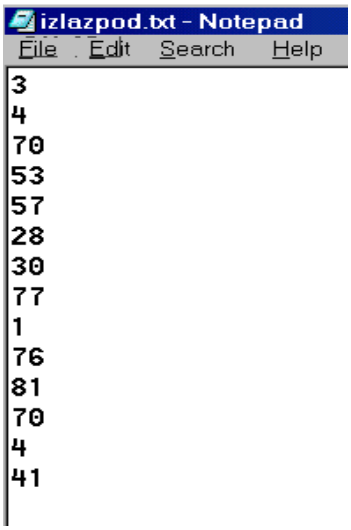
```
Sub poljeizExcela()
        ucitaj
                ' podaci iz Excel tablice koje treba sortirati
        For i = x To (x + n - 1)
                For j = y To (y + m - 1)
                        a(i - x, j - y) = Cells(i, j)
                Next j
        Next i
End Sub
```

Zajedno sa podacima o veličini matrice (*n, m*) ispisuje se polje podataka matrice u datoteku *"izlazpod.txt"* koja se kreira pomoću naredbe *open.*

```
Sub ispisudatoteku()
Dim broj As Integer
        poljeizExcela
                        'ispis podataka u datoteku izlazpod
```

```
        broj = FreeFile()
        Open "C:\izlazpod.txt" For Output As #broj
        Write #broj, n
        Write #broj, m
        For i = 0 To n - 1
                For j = 0 To m - 1
                        Write #broj, a(i, j)
                Next j
        Next i
        Close #broj
End Sub
```

U datoteci *"izlazpod.txt"* prvi podatak je vrijednost *n* (broj redova), drugi podatak je *m* (broj stupaca), a zatim slijede podaci polja koji su smješteni jedan ispod drugog.(Sl. 4.2.2)



Sl. 4.2.2 Ispis podataka u datoteku

Nakon što su podaci ispisani u tekstualnu datoteku, pokreće se izvršna .EXE datoteka ovisno o vrsti sortiranja. Ako je sortiranje od većih prema manjim brojevima pokreće se datoteka *veci.exe*, a u drugom slučaju datoteka *manji.exe*.

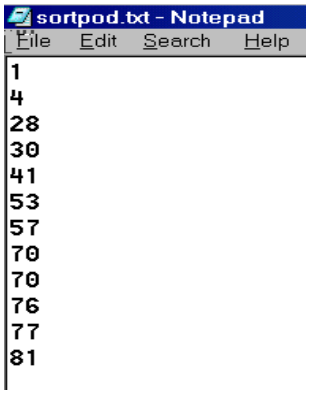
Programi su napisani kao konzolne aplikacije i koriste metodu ljusaka (*shells*) za sortiranje niza. Izvorni kod (*source*) programa *manji.cpp* glasi:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#pragma hdrstop
#define NMAX 450
FILE *stream;
```

```
int main(int argc, char **argv)
{
	int n,m;
	int a[NMAX],v;
	int i,j,inc;
		//čitanje podataka iz datoteke
	if((stream = fopen( "izlazpod.txt", "r+t" )) != NULL )
	{
		fscanf( stream, "%d\n",&n );
		fscanf( stream, "%d\n",&m );
		for (i=0;i<n;i++)
			for(j=0;j<m;j++)
			fscanf( stream, "%d\n",&a[i*m+j+1] );
		fclose( stream );
	}
	else
	{
		printf( "File could not be opened\n" );
		return 0;
	}

		//sortiranje (metoda ljuski)
	inc=1;.
	do {
		inc *= 3;
		inc++;
	} while (inc <=n*m);

	do {
	inc /= 3;
	for (i=inc+1;i<=n*m;i++)
	   {
		v=a[i];
		j=i;
		while (a[j-inc] > v)
		{
			a[j]=a[j-inc];
			j -= inc;
			if (j <= inc) break;
		}
		a[j]=v;
	   }
	}while (inc > 1);

		//ispis u datoteku
	stream = fopen( "sortpod.txt", "w+t" );
```

```
        for (i=0;i<n;i++)
                for(j=0;j<m;j++)
                fprintf( stream, "%d\n",a[i*m+j+1] );
        fclose( stream );
        return 0;
}
```

Program *manji.exe* otvara tekstualnu datoteku *sortpod.txt* za pisanje i čitanje podataka pomoću funkcije *fopen* (Sl. 4.2.3). U datoteci *sortpod.txt* nalaze se samo sortirani podaci (u ovom slučaju sortirani od manjih brojeva prema većim).



Sl. 4.2.3 Sortirani podaci u datoteci *sortpod.txt*

Posljednji korak predstavlja čitanje podataka iz datoteke i njihovo smještanje u Excelovu tablicu. Podaci se ispisuju u obliku sortirane matrice na istoj poziciji kao i početna matrica, samo u novom radnom listu *"rez(dat)"*.

```
Sub procitajizdat()
Dim broj As Integer
        ucitaj
                'sortirani podaci iz datoteke sortpod
        broj = FreeFile()
        Open "C:\sortpod.txt" _
        For Input Access Read As #broj
        For i = 0 To n - 1
                For j = 0 To m - 1
                        Input #broj, a(i, j)
                Next j
        Next i
        Close #broj
                'ispis sortiranih podataka u Excelov radni list rez(dat)
        Worksheets("rez(dat)").Activate
        Range("a1:az60").Clear
        For i = 0 To n - 1
                For j = 0 To m - 1
```

```
                    Cells(i + x, j + y) = a(i, j)
                    Cells(i + x, j + y).Interior.ColorIndex = 6
                Next j
            Next i
        End Sub
```

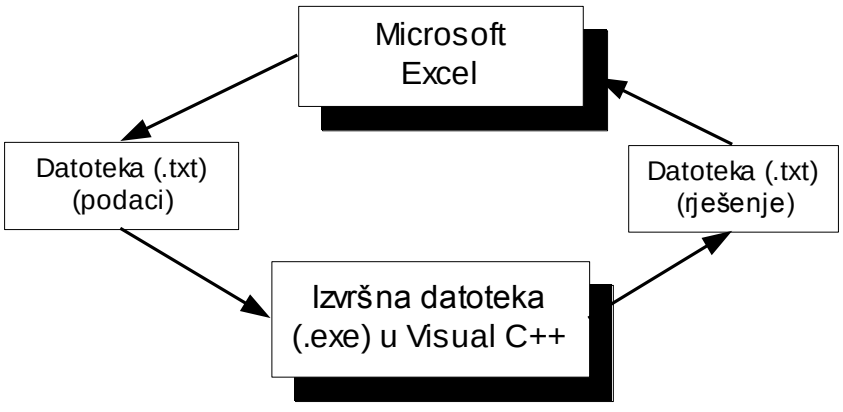
Cijeli postupak sortiranja pomoću datoteka sažet je u jednoj proceduri (*svedat*) koja obavlja sve gore navedene postupke:

- ispisuje podatke iz Excelove tablice u datoteku,
- ovisno o izboru sortiranja poziva programe koji sortiraju podatke (*veci.exe* ili *manji.exe*),
- nakon što su podaci sortirani, čita ih iz datoteke i smješta u Excelovu tablicu.

```
Sub svedat()
Dim vrijednost
Dim d As Integer
        Worksheets("upis podataka").Activate
        d = Cells(7, 4)
        ispisudatoteku
        If (d = 1) Then
                vrijednost = Shell("C:\veci.EXE", 1)
        ElseIf (d = 2) Then
                vrijednost = Shell("C:\manji.EXE", 1)
        Else: MsgBox ("Pogrešan unos podataka")
        End If
        procitajizdat
End Sub
```

Buton objektu *"sortiranje(dat)"* (Sl. 4.2.1) pridružen je makro *svedat()* koji pokreće i koordinira cijeli postupak sortiranja podataka matrice pomoću datoteka.

Općenito se povezivanje Excela sa izvršnom datotekom napisanom u Visual C++ preko tekstualnih datoteka može shematski prikazati kao na Sl. 4.2.4



Sl. 4.2.4 Povezivanje Excela sa C++ pomoću datoteka

4.3 Povezivanje Excela sa C++ pomoću DLL-a

Pomoću biblioteka za dinamičko povezivanje (DLL) omogućeno je uspostavljanje direktne veze između programskog jezika C++ i Excela.
Prilikom kreiranja DLL-a potrebno je prije svega odabrati tip DLL-a koji se kreira. Zbog toga što Microsoft Excel treba pozivati taj DLL, potrebno je odabrati tip DLL-a koji podržava rad sa MFC (*Microsoft Foundation Class*). I *regular DLL* i *extension DLL* će se bez problema pozivati iz Excela. Prednost u odabiru ipak je pripala *extension DLL*-u, jer prilikom njegovog kreiranja nije potrebno voditi računa da li će se DLL statički ili dinamički povezivati sa MFC.

Pomoću *AppWizard(DLL)* kreiran je *MFC Extension DLL*. *AppWizard* kreira osnovne datoteke, a među njima i .CPP datoteku i .DEF datoteku.

U .CPP datoteci nalazi se *DllMain* funkcija koja sadrži kod za inicijalizaciju i terminaciju DLL-a, tj. predstavlja ulaznu i izlaznu točku pri pozivu DLL-a. U toj datoteci također su smještene i sve ostale funkcije koje je potrebno da DLL obavlja. U ovom slučaju to su funkcije za sortiranje niza podataka i to *sortpoljaV* (sortira podatke od većih prema manjim) i *sortpoljaM* (sortira podatke od manjih prema većim).

```
#include <windows.h>
#include "stdafx.h"
#include <afxdllx.h>
#include <afxpriv.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define NMAX 400
static AFX_EXTENSION_MODULE BbDLL = { NULL, NULL };

extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
        UNREFERENCED_PARAMETER(lpReserved);
        if (dwReason == DLL_PROCESS_ATTACH)
        {       TRACE0("BB.DLL Initializing!\n");
                        // Extension DLL inicijalizacija
                if (!AfxInitExtensionModule(BbDLL, hInstance))
                        return 0;
                new CDynLinkLibrary(BbDLL);
        }
        else if (dwReason == DLL_PROCESS_DETACH)
        {       TRACE0("BB.DLL Terminating!\n");
```

```
		// Terminacija biblioteka prije poziva destruktora
		AfxTermExtensionModule(BbDLL);
	}
	return 1;
}

typedef SAFEARRAY * FPSAFEARRAY;

short WINAPI sortpoljaV(FPSAFEARRAY *ppsa, int vel)
{	short Elem;
	long donjagr, gornjagr, l;
	int a[NMAX],inc,i,j,v;

	if (*ppsa == NULL)		// polje nije inicijalizirano
		return -4;
	if ((*ppsa)->cDims != 1)	// broj dimenzija polja
		return -5;
					//donja i gornja granica polja
	if (FAILED(SafeArrayGetLBound(*ppsa, 1, &donjagr)) ||
		FAILED(SafeArrayGetUBound(*ppsa, 1, &gornjagr)))
		return -1;
					// čitanje elemenata polja
	for (l =donjagr; l <= gornjagr; l++)
	{	if (FAILED(SafeArrayGetElement(*ppsa, &l, &Elem)))
			return -2;
		a[l]=Elem;
	}

	// sortiranje od većeg prema manjem elementu
	inc=1;
	do {
		inc *= 3;
		inc++;
	}while (inc <= vel);

	do {
		inc /= 3;
		for (i=inc+1;i<=vel;i++)
		{	v=a[i];
			j=i;
			while (a[j-inc] < v)
			{
				a[j]=a[j-inc];
				j -= inc;
				if (j <= inc) break;
			}
			a[j]=v;
		}
```

```
        }while (inc > 1);
                                        //zapisivanje elemenata sortiranog polja
        for (l = donjagr ; l <= gornjagr; l++)
        {       Elem =a[l];
                if (FAILED(SafeArrayPutElement(*ppsa, &l, &Elem)))
                        return -3;
        }
        return 0;
}

short WINAPI sortpoljaM(FPSAFEARRAY *ppsa, int vel)
{       short Elem;
        long donjagr, gornjagr, l;
        int a[NMAX],inc,i,j,v;

        if (*ppsa == NULL)      // polje nije inicijalizirano
                return -4;
        if ((*ppsa)->cDims != 1)  // broj dimenzija polja
                return -5;
                        //donja i gornja granica polja
        if (FAILED(SafeArrayGetLBound(*ppsa, 1, &donjagr)) ||
                FAILED(SafeArrayGetUBound(*ppsa, 1, &gornjagr)))
                return -1;
                                        // čitanje elemenata polja
        for (l =donjagr; l <= gornjagr; l++)
        {
                if (FAILED(SafeArrayGetElement(*ppsa, &l, &Elem)))
                        return -2;
                a[l]=Elem;
        }
                // sortiranje od manjeg prema većem elementu
        inc=1;
        do {    inc *= 3;
                inc++;
        }while (inc <=vel);
        do {    inc /= 3;
                for (i=inc+1;i<=vel;i++)
                {       v=a[i];
                        j=i;
                        while (a[j-inc] > v)
                        {       a[j]=a[j-inc];
                                j -= inc;
                                if (j <= inc) break;
                        }
                a[j]=v;
                }
        } while (inc > 1);
                //zapisivanje elemenata sortiranog polja
```

```
        for (l = donjagr ; l <= gornjagr; l++)
        {       Elem =a[l];
                if (FAILED(SafeArrayPutElement(*ppsa, &l, &Elem)))
                        return -3;
        }
        return 0;
}
```

Pri prijenosu polja iz Excela u DLL koristi se SAFEARRAY struktura, a vrijednosti elemenata polja čitaju se pomoću *SafeArrayGetElement* funkcije. Nakon što su podaci sortirani smještaju se u polje podataka pomoću *SafeArrayPutElement* funkcije. Sortiranje je napravljeno pomoću metode ljusaka (*shell method*).

Osim .CPP datoteke u kojoj je smješten kod funkcija, potrebno je promijeniti i modulski definiranu datoteku (.DEF) u kojoj se navode imena funkcija sa njihovim ordinalnim brojevima:

```
LIBRARY     "bb"
DESCRIPTION  'bb Windows Dynamic Link Library'
        EXPORTS
        sortpoljaV @1
        sortpoljaM @2
```

Nakon što je DLL kreiran, potrebno je da ga se prije poziva iz Excela ili kopira na neko od mjesta na putanji traženja DLL-a (direktorij s .EXE datotekom, trenutno aktivni direktorij, sistemski direktorij Windowsa, Windows direktorij) ili da se kod deklaracije DLL-a u Excelu točno navede njegova lokacija na disku.

Deklaracija funkcija *sortpoljaV* i *sortpoljaM* u Excelu napravljena je pomoću naredbe *Declare* i glasi:

```
Declare Function sortpoljaV Lib "bb" (a() As Integer, ByVal r As Integer) As Integer

Declare Function sortpoljaM Lib "bb" (a() As Integer, ByVal r As Integer) As Integer
```

Nakon što je DLL deklariran, sada se njegove funkcije pozivaju u Visual Basicu za aplikacije kao da su dio VBA koda.
U proceduri *sortdll( )* napisanoj u VBA pročitana je matrica podataka koja se sortira, te se ovisno o izboru sortiranja poziva ili funkcija *sortpoljaV* ili *sortpoljaM*. Sortirani podaci ispisuju se u novi list Excelove radne bilježnice *"rez(DLL)"*.

```
Sub sortdll()
Dim b(250) As Integer
Static c As Integer
        ucitaj
        poljeizExcela
        For i = 0 To n - 1
```

```
                For j = 0 To m - 1
                        b(i * m + j + 1) = a(i, j)
                Next j
        Next i
        c = m * n
        Worksheets("upis podataka").Activate
        d = Cells(7, 4)
        If (d = 1) Then
                z = sortpoljaV(b, c)
                Worksheets("rez(dll)").Activate
                For i = 0 To n - 1
                        For j = 0 To m - 1
                                Cells(i + x, j + y) = b(i * m + j + 1)
                                Cells(i + x, j + y).Interior.ColorIndex = 6
                        Next j
                Next i
        ElseIf (d = 2) Then
                z = sortpoljaM(b, c)
                Worksheets("rez(dll)").Activate
                For i = 0 To n - 1
                        For j = 0 To m - 1
                                Cells(i + x, j + y) = b(i * m + j + 1)
                                Cells(i + x, j + y).Interior.ColorIndex = 6
                        Next j
                Next i
        Else: MsgBox ("Pogrešan unos podataka")
        End If
End Sub
```

Pokretanje makroa *sortdll( )* koji omogućava povezivanje Excela sa C++ pomoću DLL-a, omogućeno je preko *button* objekta *"sortiranje (DLL)"* (Sl. 4.2.1).

5. ZAKLJUČAK

U ovom diplomskom radu obrađene su mogućnosti komuniciranja MS Excela sa eksternim aplikacijama. Detaljnije je opisan najjednostavniji način preko odgovarajućih ulazno/izlaznih datoteka, te najefikasniji način ukoliko se radi o izradi vlastitih aplikacija, tj. povezivanje korištenjem vanjskih aplikacija kao DLL biblioteke. Također je pomoću primjera za sortiranje podataka matrice po njihovoj veličini, prikazano povezivanje Excel dokumenta sa programima izrađenim programskim jezikom Visual C++.

Komunikacija između Excel dokumenta i izvršnih datoteka (*manji.exe* i *veci.exe*) napisanih u Visual C++, preko tekstualnih datoteka (*izlazpod.txt* i *sortpod.*txt), koordinirana je iz Excela u makrou *svedat()* napisanom u VBA.

Drugi način komunikacije između Excela i Visual C++ ostvaren je pomoću biblioteka za dinamičko povezivanje (DLL-a). Sortiranje podataka napravljeno je u funkcijama *sortpoljaV* i *sortpoljaM* iz biblioteke za dinamičko povezivanje *bb.dll,* a deklaracija i pozivanje tih funkcija napravljeno je iz VBA makroa *sortdll( ).*

Komunikacija Excela sa Visual C++ indirektnim načinom pomoću tekstualnih datoteka ima nedostatak da se prilikom prebacivanja na neko novo računalo mora u samom kodu programa korigirati njegovo novo odredište. Taj problem je kod DLL-a riješen na taj način da se DLL kopira na neku lokaciju koja se nalazi na stalnoj putanji traženja DLL-a.

Povezivanje preko DLL-a također ima svoju manu. Nakon što je DLL kreiran testirano je njegovo pozivanje sa različitih operativnih sistema i iz različitih verzija Excela. Uočeno je da prilikom pozivanja DLL-a, kreiranog u Visual C++ 6.0, iz starijih verzija Excela (npr. Excel 95) dolazi do problema sa inicijalizacijom DLL-a. Isto tako ako je DLL kreiran u starijoj verziji Visual C++ (npr. Visual C++ 4.2.) tada ga je problem pozvati iz novijih verzija Excela.
Do tih problema dolazi zbog toga što i Excel i DLL pozivaju interno podatke iz MFC (*Microsoft Foundaution Class*), pa je potrebno da MFC bude ista i za Excel i za DLL, tj. da bi povezivanje radilo bez problema potrebno je da su Excel i Visual C++ programi istog vremenskog razdoblja.

LITERATURA:

C. Petzold: *Kako programirati Windows 95*, Microsoft Press – Znak, 1996.

T. Čukman, V. Bolt: *C/C++ kroz primjere* – Procon, 1994.

K. Reisdorph: *Teach yourself Borland C++ Builder 3 in 21 days* – Sams Publishing, 1998.

V. Bolt, D. Bolt: *Excel 5.0 korak dalje* – Procon, 1995.

E. Wells: *Excel for Windows 95*, Microsoft Press – Znak, 1996.

Microsoft Developer Network – CD-ROM

S A D R Ž A J: