



Tema 08

Pregled standardne i STL biblioteke jezika C++

Prof. dr Miodrag Živković

Tehnički fakultet

OBJEKTNO ORIJENTISANO PROGRAMIRANJE 2



Sadržaj

1. Uvod
2. Generičke funkcije i klase
3. Standardna biblioteka jezika C++
4. Standardna biblioteka šablona (STL)
5. Vektorska grafika u jeziku C++



1. Uvod

- Standardna biblioteka jezika C++
- Standardna biblioteka šablona jezika C++



Standardna biblioteka jezika C++

- Standardna biblioteka predstavlja *skup klasa i funkcija* koje predstavljaju osnovu i sastavni deo ISO standarda jezika C++
 - biblioteka obezbeđuje nekoliko generičkih kontejnera (struktura podataka), funkcije za njihovu upotrebu, funkcijske objekte, generičke nizove, tokove podataka (interaktivni ulaz-izlaz i rad s fajlovima), kao i neke često potrebne funkcije, npr. kvadratni koren
 - biblioteka uključuje zaglavlja *C standarda* (ISO C90) sa sufiksom *.h* koja se više ne koriste
- Mogućnosti Standardne biblioteke deklarisanе su u prostoru imena (*namespace*) *std*
 - biblioteka je veoma obimna, tako da se uključivanje celog prostora imena *std* izbegava

Standardna biblioteka šablona jezika C++

- **Standardna biblioteka šablona** (*Standard Template Library*, STL) uticala je na razvoj mnogih delova Standardne biblioteke jezika C++
 - Alexander Stepanov (od 1979), Meng Lee (od 1992) iz kompanije HP
 - kompanija HP je 1994. godine izvorni kod biblioteke STL dala na besplatno korišćenje
 - biblioteka STL je 1998. godine uključena u **standard jezika C++**
- *Obe programske biblioteke imaju veliki broj zajedničkih svojstava, iako nijedna ne predstavlja striktni podskup druge*
 - standardna biblioteka šablona je po obimu znatno manja, ali od nje zavise mnogi delovi standardne biblioteke jezika C++

2. Generičke funkcije i klase

1. Upotreba šablona u jeziku C++
2. Generisanje funkcija i klasa



2.1 Upotreba šablona u jeziku C++

- Preklapanje operatora omogućava korišćenje istog naziva funkcije za operacije s različitim tipovima podataka, npr.

```
char max (char i, char j) { return i>j ? i : j; }  
int  max (int  i, int  j) { return i>j ? i : j; }  
float max (float i, float j) { return i>j ? i : j; }
```

Funkcije se tipično razlikuju samo po *tipovima* podataka

- U jeziku C++ moguće je definisanje **šablona** (*template*) za opis takvih funkcija, gde se *tip* podataka zadaje kao **parametar** šablona (tzv. *generičke funkcije*)
- Na isti način mogu se opisati i klase, za koje se tipovi nekih polja i parametri metoda mogu definisati naknadno (*generičke klase*)

Definisanje šablona

- Šablon generičke funkcije ima oblik

```
template <parametar1, parametar2, ...> opis
```

gde *opis* može biti *deklaracija* (prototip) ili *definicija* generičke funkcije ili klase, a *parametri* označavaju tipove ili konstante:

```
class naziv_tipa  
typename naziv_tipa  
naziv_tipa naziv_konstante
```

- Primeri definicije šablona klase i šablona novog tipa podataka

```
template <class T> T max (T a, T b) { return a>b ? a : b; }
```

```
template <typename T, int n>  
class Vektor {  
    T niz [n];  
public:  
    T& operator[] (int i) { return niz[i]; }  
};
```


2.2 Generisanje funkcija i klasa

- Funkcije i klase definisane šablonima generišu se za konkretne tipove i konstante na mestima njihovog pozivanja
- Pozivanje se vrši izrazima oblika

```
naziv_funkcije <argument1, argument2, ...>
```

```
naziv_Klase <argument1, argument2, ...>
```

gde *argumenti* mogu biti

naziv_tipa

konstantni_izraz

- Primeri

```
int a= max<int>(1,2);           // generiše se int max(int,int)
char b=max<char>('1','2');      // generiše se char max(char,char)
Vektor<int,10> vekt1;           // niz od 10 elemenata tipa int
```

3. Standardna biblioteka jezika C++

- Osnovne komponente Standardne biblioteke
- Implementacija Standardne biblioteke



Osnovne komponente Standardne biblioteke

- Skup višestruko upotrebljivih *komponenti* zasnovanih na *šablonima* koje implementiraju veliki broj opštih struktura podataka i algoritama za njihovo korišćenje
- Osnovni elementi mogu se koristiti pomoću zaglavlja
 - kontejneri
 - opšte funkcije
 - lokalizacija
 - stringovi
 - tokovi
 - niti
 - numerička
 - C biblioteka



Implementacija Standardne biblioteke

- [Apache C++ Standard Library](#) - besplatna biblioteka otvorenog koda, implementacija međunarodnog standarda za C++ ISO/IEC 14882
 - Apache Software Foundation
- [Microsoft C++ Standard Library](#) - biblioteka otvorenog koda kompanije Microsoft za Visual C++
 - GitHub, od 2019



4. Standardna biblioteka šablona (STL)

1. Komponente Standardne biblioteke šablona
2. Kontejneri
3. Iteratori
4. Algoritmi
5. Funkcijski objekti



4.1 Komponente Standardne biblioteke šablona

- Osnovne komponente Standardne biblioteka šablona (template) jezika C++ su:
 - kontejneri
 - iteratori
 - algoritmi
 - funkcijski objekti



4.2 Kontejneri

- **Kontejneri** su objekti (strukture podataka) koji služe za smeštanje i organizovanje drugih objekata
- Npr. šablon `vector<T>` je kontejner za jednodimenzionalna polja, koji po potrebi automatski povećava svoje dimenzije
- Tip `T` može biti osnovni tip ili neka korisnička klasa
 - klase treba pamtit i pomoću pokazivača da se izbegne potencijalni gubitak informacija o izvedenim klasama (*object slicing*)
- Primeri upotrebe (prostor imena `std`)

```
std::vector<std::string> stringovi; // objekti tipa string
std::vector<double> podaci;         // vrednosti tipa double
```
- Šabloni kontejnerskih klasa definisani su u zaglavljima:
 - `vector`, `array`, `deque`, `list`, `forward_list`, `map`, `unordered_map`, `set`, `unordered_set` i `bitset`

Zaglavlja kontejnerskih klasa

Zaglavlje	Opis
<code>vector</code>	<code>vector<T></code> je proširivo polje, novi elementi dodaju se na kraj
<code>array</code>	<code>array<T,N></code> je polje fiksnih dimenzija od N elemenata (efikasnije)
<code>deque</code>	<code>deque<T></code> je red koji se može ažurirati s obe strane
<code>list</code>	<code>list<T></code> je dvostruko povezana lista elemenata tipa T
<code>forward_list</code>	<code>forward_list<T></code> je jednostruko povezana lista elemenata tipa T
<code>map</code>	<code>map<K,T></code> je asocijativna lista objekata tipa <i>pair<K,T></i> (K je ključ)
<code>unordered_map</code>	<code>unordered_map<K,T></code> je asocijativna lista objekata tipa <i>pair<K,T></i> za koje nije definisan poredak
<code>set</code>	<code>set<T></code> je kontejner <i>map</i> , gde je element liste istovremeno i ključ
<code>unordered_set</code>	<code>unordered_set<T></code> je kontejner <i>set</i> za čije elemente nije definisan poredak
<code>bitset</code>	<code>bitset<T></code> je klasa koja predstavlja niz bitova, npr. flegova

Alokatori i komparatori

- Većina kontejnerskih struktura automatski se prilagođava broju elemenata koji se u njih smeštaju

- Npr. stvarna forma šablona `vector<T>` je

```
vector<T, Allocator=allocator<T>>
```

tako da se može definisati sopstveni alokator (šablon klase)

- Neki kontejneri pretpostavljaju poredak svojih elemenata, kao npr. kontejner *map*

```
map<K, T, Compare=less<K>, Allocator=allocator<pair<K,T>> >
```

- Element strukture *Compare* je funkcijski objekt koji poredi ključeve tipa K i određuje njihov redosled
- Moguće je definisati sopstveni komparator, npr. "veći od"

Primer: Upotreba šablona vektora

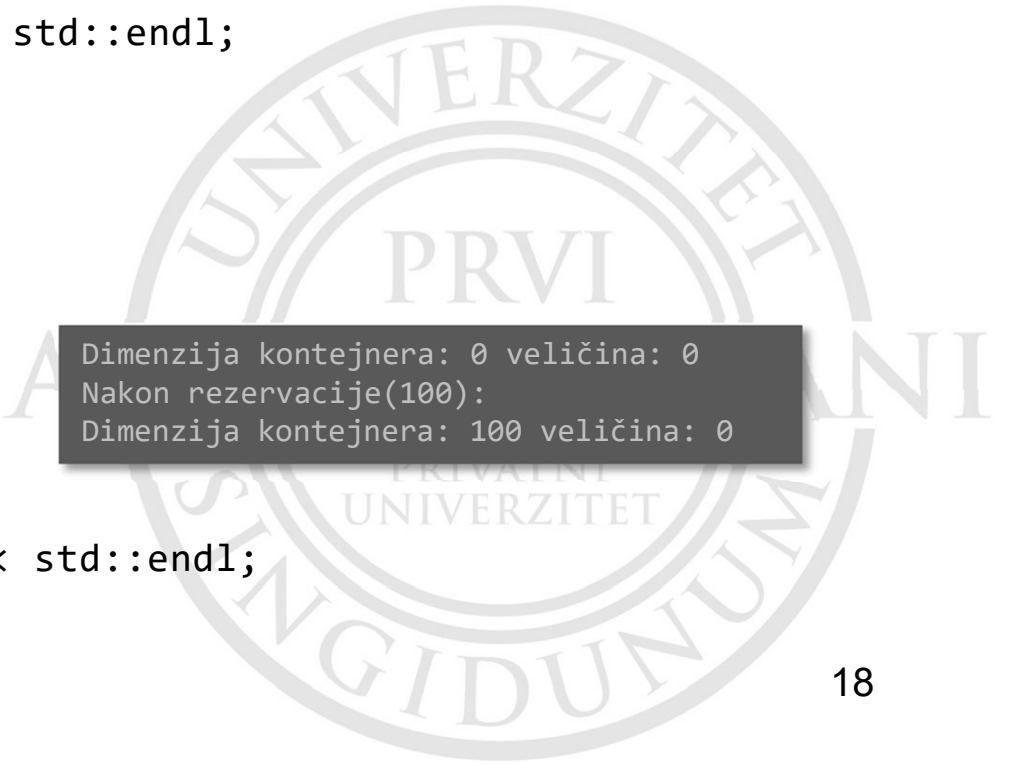
(1/2)

```
#include <iostream>
#include <vector>
using std::vector;

// Šablonska funkcija za prikaz dimenzija i broja elemenata vektora
template<class T>
void listInfo(const vector<T>& v) {
    std::cout << "Dimenzija kontejnera: " << v.capacity()
               << " velicina: " << v.size() << std::endl;
}

int main() {
    // Kreiranje osnovnog vektora
    vector<double> podatak;
    listInfo(podatak);

    // Kreiranje vektora od 100 elemenata
    podatak.reserve(100);
    std::cout << "Nakon rezervacije (100):" << std::endl;
    listInfo(podatak);
}
```



```
Dimenzija kontejnera: 0 velicina: 0
Nakon rezervacije(100):
Dimenzija kontejnera: 100 velicina: 0
```

Primer: Upotreba šablona vektora

(2/2)

```
// Kreiranje i inicijalizacija vektora od 10 elemenata (-1)
vector<int> brojevi(10,-1);
std::cout << "Inicijalna vrednost vektora je: ";
for (auto n : brojevi) std::cout << " " << n; // petlja nad vektorom
std::cout << std::endl << std::endl;
```

auto - tip vrednosti određuje inicijalizator, u ovom slučaju tip elementa kontejnera

```
// Provera da li dodavanje elemenata utiče na obim vektora
auto staraDim = brojevi.capacity(); // stari obim
auto novaDim = staraDim; // novi obim, nakon dodavanja elementa
```

```
listInfo(brojevi);
for (int i=0; i<1000; i++) {
    brojevi.push_back(2*i);
    novaDim = brojevi.capacity();
    if (staraDim < novaDim) { // povećanje
        staraDim = novaDim;
        listInfo(brojevi);
    }
}
return 0;
}
```

```
Dimenzija kontejnera: 0 velicina: 0
Nakon rezervacije (100):
Dimenzija kontejnera: 100 velicina: 0
Inicijalna vrednost vektora je: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Dimenzija kontejnera: 10 velicina: 10
Dimenzija kontejnera: 15 velicina: 11
Dimenzija kontejnera: 22 velicina: 16
Dimenzija kontejnera: 33 velicina: 23
Dimenzija kontejnera: 49 velicina: 34
Dimenzija kontejnera: 73 velicina: 50
Dimenzija kontejnera: 109 velicina: 74
Dimenzija kontejnera: 163 velicina: 110
Dimenzija kontejnera: 244 velicina: 164
Dimenzija kontejnera: 366 velicina: 245
Dimenzija kontejnera: 549 velicina: 367
Dimenzija kontejnera: 823 velicina: 550
Dimenzija kontejnera: 1234 velicina: 824
```

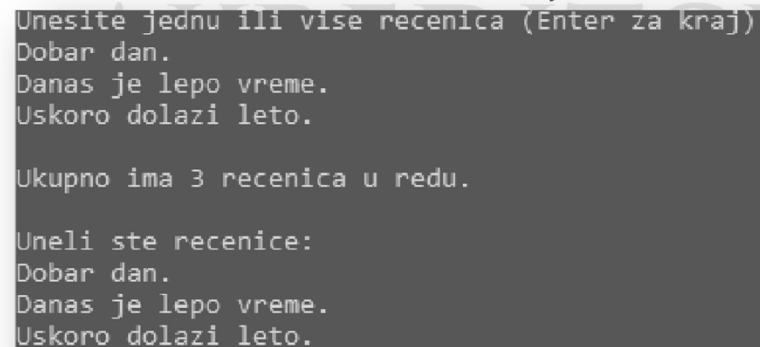
automatsko povećanje dimenzija vektora (capacity) vrši se na način koji zavisi od implementacije; povećanje je na $k \cdot N$ u odnosu na postojeći obim N , gde je k obično 1.5 ili 2

Kontejnerski adapteri

- **Kontejnerski adapteri** su šabloni klasa koji se definišu na osnovu *postojećih* kontejnerskih klasa, obično ograničavanjem njihovih svojstava
- Primeri kontejnerskih adaptera su **red** (*queue*) i **stek** (*stack*), koji se definišu ograničavanjem operacija samo na jednu stranu nekog osnovnog kontejnera
 - **red** se može definisati na osnovu kontejnera `deque<T>` ili `list<T>`
 - **stek** se može definisati na osnovu kontejnera `deque<T>`, `vector<T>` ili `list<T>`
- Adapter *queue* (dostupan preko zaglavlja **<queue>**) dodaje elemente na kraj, a izuzima s početka reda
 - ima metode : `empty()`, `size()`, `front()`, `back()`, `push_back()` i `pop_front()`

Primer: Upotreba šablona kontejnerskog adaptera *queue*

```
#include <iostream>
#include <queue>
#include <string>
using namespace std;
int main() {
    queue<string> recenice;
    string recenica;
    cout << "Unesite jednu ili vise recenica (Enter za kraj)" << endl;
    while (true) {
        getline(cin, recenica);
        if (recenica.empty())
            break;
        recenice.push(recenica);
    }
    cout << "Ukupno ima " << recenice.size() << " recenica u redu.\n" << endl;
    cout << "Uneli ste recenice:" << endl;
    while (!recenice.empty()) {
        cout << recenice.front() << endl;
        recenice.pop();
    }
    return 0;
}
```



```
Unesite jednu ili vise recenica (Enter za kraj)
Dobar dan.
Danas je lepo vreme.
Uskoro dolazi leto.

Ukupno ima 3 recenica u redu.

Uneli ste recenice:
Dobar dan.
Danas je lepo vreme.
Uskoro dolazi leto.
```

4.3 Iteratori

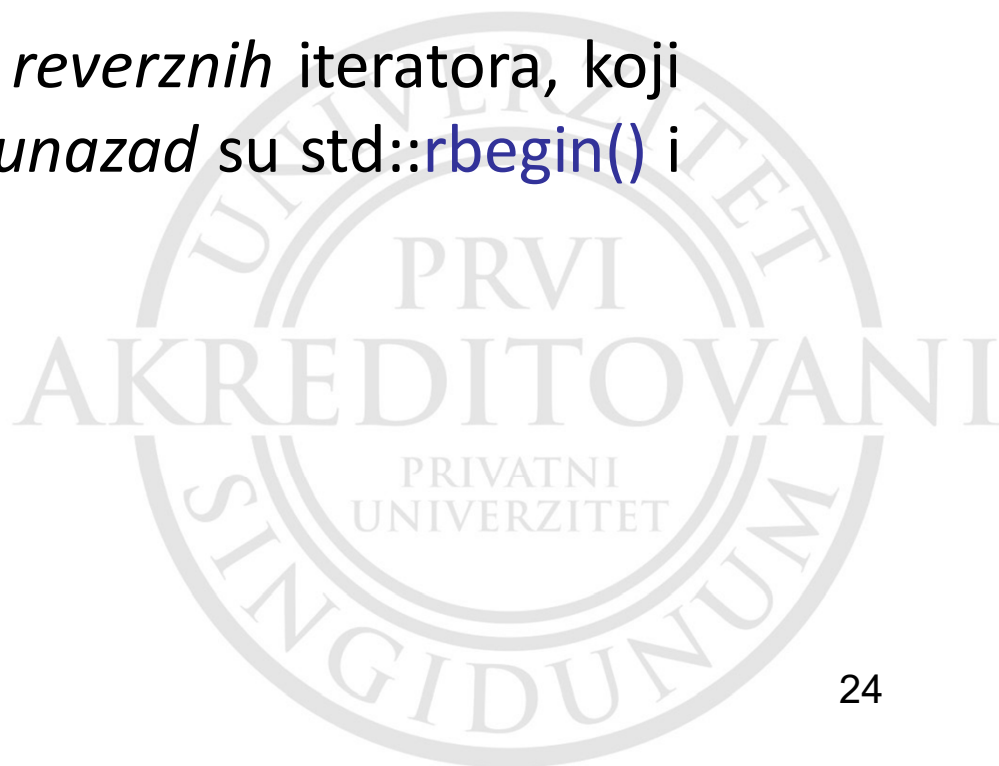
- **Iteratori** su objekti tipa pokazivača, koji se koriste za pristup objektima kontejnera koji nisu definisani pomoću adaptera
 - iterator za pristup objektima može se dobiti na osnovu kontejnera
 - može se kreirati iterator za ulaz i izlaz objekata ili podataka određenog tipa preko odgovarajućih tokova
 - iteratori imaju iste neke osnovne funkcije, npr. za njihovo poređenje (`==`, `!=` i `=`) i promenu pozicije (inkrement `++` i dekrement `--`)
- Iterator je promenljiva koja pokazuje na podatak u okviru kontejnera
- Svaka kontejnerska klasa ima svoj tip iteratora, ali se iteratori svih kontejnerskih klasa koriste na isti način

Operacija nad iteratorima

- Osnovne operacije nad iteratorom su:
 - prefiksni i postfiksni **inkrement** (**++**) i **dekrement** (**--**) koji pomera iterator na sledeći ili prethodni element kontejnera
 - operatori **==** i **!=**, za ispitivanje da li iteratori pokazuju na isti element
 - operator **indirekcije** *****, koji obezbeđuje pristup podatku na koji je iterator pozicioniran (zavisno od konkretne klase kontejnera, pristup može biti za čitanje, pisanje ili i čitanje i pisanje)
- Postoje četiri kategorije iteratora
 - ulazni i izlazni iteratori (*input and output*), samo za čitanje ili upis
 - za obilazak objekata (*forward*), i za čitanje i za upis u jednom smeru
 - bidirekcionni iteratori, omogućavaju **--iter** i **iter++**
 - iteratori za direktni pristup (*random access*), npr. **iter[n]**, **iter+n**, ...

Funkcije koje vraćaju iteratore

- Mnoge kontejnerske klase imaju funkcije članove koje vraćaju iteratore, koji pokazuju na elemente kontejnera
- Npr. funkcije `std::begin()` i `std::end()` vraćaju iteratore koji pokazuju na *prvi* odnosno *poslednji* element kontejnera - string objekta ili polja koje se prenese kao argument
- Funkcije koje vraćaju vrednost *reverznih* iteratora, koji omogućavaju oblizak sekvenci *unazad* su `std::rbegin()` i `std::rend()`



Upotreba pokazivača

- Osnovni način upotrebe iteratora za obilazak svih elemenata kontejnera je:

```
kontejner<T>::iterator p;  
for (p = kontejner.begin(); p != kontejner.end(); p++)  
    // Obrada elementa kontejnera na poziciji p
```

- Određivanje tipa **T** promenljive na osnovu tipa dodeljene vrednosti može skratiti kod koji koristi šablone i iteratore
- Umesto pune deklaracije

```
vector<int>::iterator p = v.begin();
```

može se napisati kraće

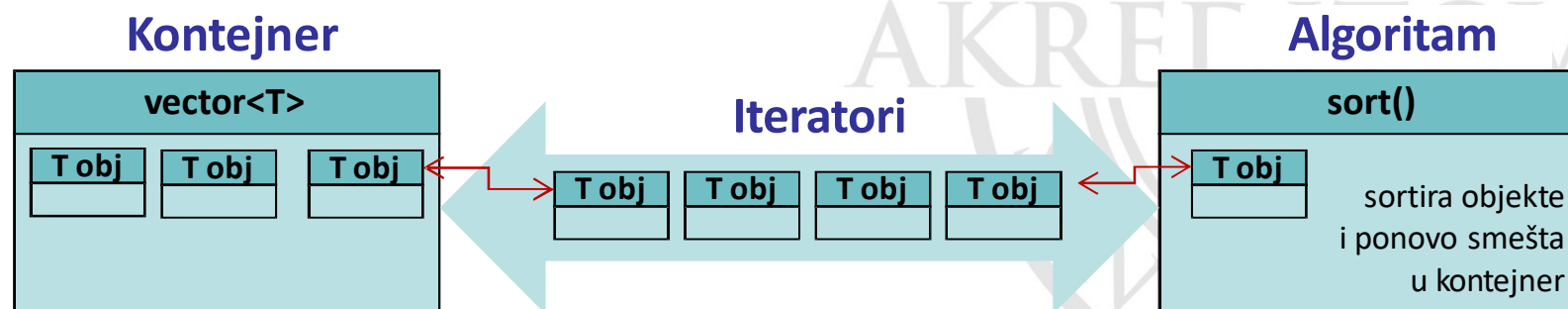
```
auto p = v.begin();
```

Pametni pokazivači

- Pametni pokazivači (*smart pointers*) su tipovi šablona slični pokazivačima, koji "pametno" rukuju dinamički alociranim objektima i vrše njihovo automatsko uklanjanje
 - objekt ovog tipa nikad nije potrebno brisati, jer se to vrši automatski, tako da se ne javljaju gubici memorije
- Ovo je moguće jer se upravljanje dinamičkom memorijom vrši na osnovu brojača referenci (*reference counting*). Objekti se brišu kad na njih više ne pokazuje nijedan pokazivač
- Postoje tri tipa šablona pametnih pokazivača:
 - `unique_ptr<T>` definiše *jedinstveni* objekt određenog tipa
 - `shared_ptr<T>` definiše objekt na koji može pokazivati *više pokazivača*
 - `weak_ptr<T>` sadrži pokazivač povezan s deljivim pokazivačem

4.4 Algoritmi

- **Algoritmi** su šabloni STL funkcija koje vrše operacije nad objektima koje im na raspolaganje stavlja iterator
- Zbog toga algoritmi nemaju informaciju o poreklu objekata, koji mogu biti iz nekog kontejnera ili toka
 - pošto su iteratori poput pokazivača, STL funkcije koje imaju iteratore kao argumente koriste ih kao obične pokazivače
 - algoritmi koriste iteratore za pristup elementima i njihovo smeštanje u niz. Npr. funkcija **sort()** koristi iteratore za pristup elementima radi poređenja i za upis objekata u kontejner u određenom redosledu



Primer: Sortiranje dvostruko povezane liste (1/2)

```
#include <iostream>
#include <deque>
#include <algorithm>
#include <numeric>
#include <functional>
using namespace std;

// šablon sort<T>()
// šablon accumulate<T>()
// funktori

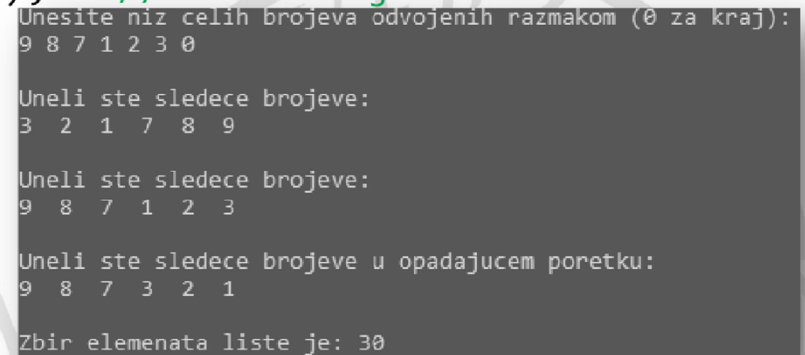
int main() {
    deque<int> podaci;
    // Unos podataka
    cout << "Unesite niz celih brojeva odvojenih razmakom (0 za kraj):" << endl;
    int vrednost = 0;
    while (cin >> vrednost, vrednost != 0)
        podaci.push_front(vrednost);
    // Ispis unesenih podataka
    cout << endl << "Uneli ste brojeve:" << endl;
    for (const auto& n : podaci)
        cout << n << " ";
    cout << endl;
```

Primer: Sortiranje dvostruko povezane liste (2/2)

```
// Ispis podataka pomoću reverznog iteratora
cout << endl << "Uneli ste, obrnutim redosledom, sledeće brojeve:" << endl;
for (auto riter = crbegin(podaci); riter != crend(podaci); ++riter)
    cout << *riter << " ";
cout << endl;

// Sortiranje liste u opadajućem poretku
cout << endl << "Uneli ste sledeće brojeve u opadajućem poretku:" << endl;
sort(begin(podaci), end(podaci), greater<>()); // sortiranje liste
for (const auto& n : podaci)
    cout << n << " ";
cout << endl;

// Računanje zbira unesenih brojeva
cout << endl << "Zbir elemenata liste je: "
    << accumulate(cbegin(podaci), cend(podaci), 0) << endl;
return 0;
}
```



```
Unesite niz celih brojeva odvojenih razmakom (0 za kraj):
9 8 7 1 2 3 0

Uneli ste sledece brojeve:
3 2 1 7 8 9

Uneli ste sledece brojeve:
9 8 7 1 2 3

Uneli ste sledece brojeve u opadajućem poretku:
9 8 7 3 2 1

Zbir elemenata liste je: 30
```

4.5 Funkcijski objekti

- Funkcijski objekti ili *funktori* su objekti tipa klase koji preklapaju operator poziva funkcije, odnosno imaju funkciju-člana `operator>()()`
 - implementacija ovog operatora može da vrati rezultat bilo kog tipa
- Npr. korisnička klasa `Linija` s preklopljenim operatorom poziva funkcije može se koristiti kao *funkcija*

```
class Linija {  
    private:  
        double nagib;  
        double y0;  
    public:  
        Linija(double nl_= 1, double y_= 0) : nagib(nl_), y0(y_) {}  
        double operator()(double x) { return y0 + nagib*x; }  
};
```

Primer: Upotreba korisničke klase s preklopljenim operatorom () kao funkcije

```
#include <iostream>
using namespace std;

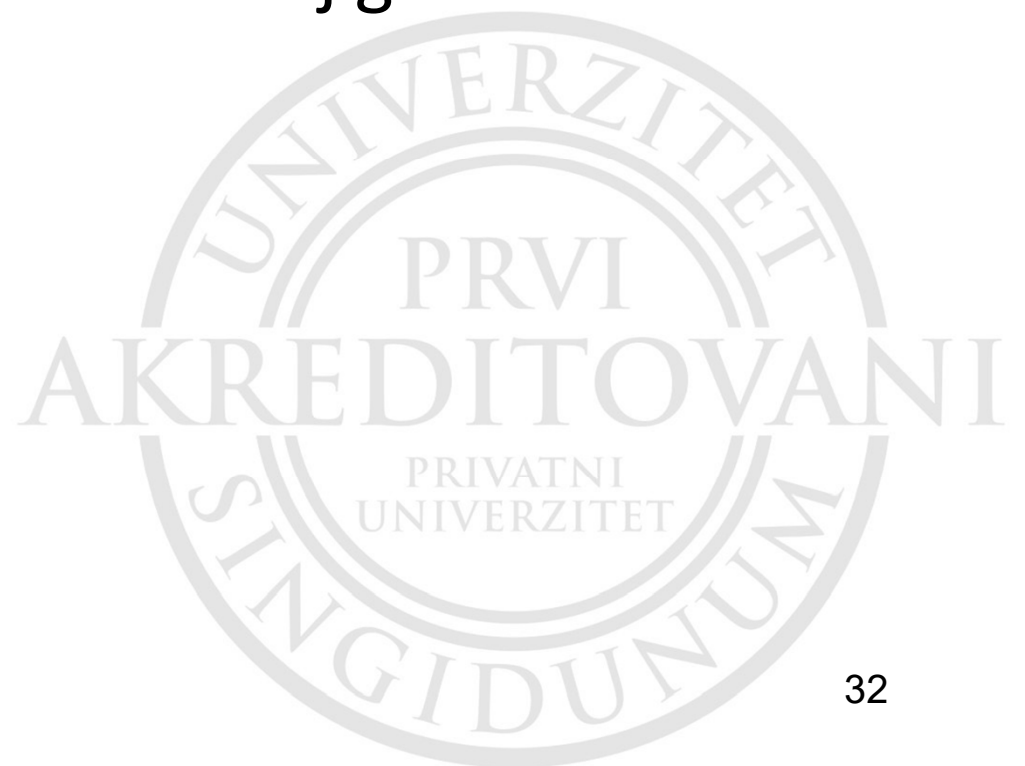
class Linija {
private:
    double nagib;
    double y0;
public:
    Linija(double nl_ = 1, double y_ = 0) : nagib(nl_), y0(y_) {}
    double operator()(double x) { return y0 + nagib*x; }
};

int main() {
    Linija f1;
    Linija f2(2.5, 10.0);
    double y1 = f1(12.5); // poziv f1.operator()(12.5) = 0+1*12.5
    double y2 = f2(0.4); // 10.0+2.5*0.4
    cout << "y1=" << y1 << " y2=" << y2 << endl;
}
```

y1=12.5 y2=11

5. Vektorska grafika u jeziku C++

1. Računarska grafika i jezik C++
2. Razvoj grafičkih aplikacija u operativnom sistemu Windows
3. Multiplatformska biblioteka za razvoj grafičkih aplikacija Qt



5.1 Računarska grafika i jezik C++

- Standardna biblioteka jezika C++ i biblioteka šablona *nemaju podršku za računarsku grafiku i multimedijske strukture*
 - računarska složenost i hardverska zavisnost
 - razvijeni softver ne bi bio prenosiv
- Biblioteke za vektorsku grafiku, obradu slika/zvuka i razvoj korisničkih interfejsa nisu deo standarda
- Neke poznatije biblioteke za razvoj 2D grafičkih aplikacija i grafičkih korisničkih interfejsa su:
 - Qt multiplatformska, obimna, popularna
 - GTK+ multiplatformska (od GIMP Toolkit)
 - Blend2D biblioteka za vektorsku grafiku razvijena u C++
 - OpenGL *nije objektno orijentisana* (jezik C)

5.2 Razvoj grafičkih aplikacija u operativnom sistemu Windows

- Windows API i biblioteka klasa MFC
- Aplikacije operativnog sistema Windows
- Aplikacije koje koriste biblioteku MFC
- Osnovna grafika u grafičkom prozoru
- Prikaz geometrijskih oblika
- Grafika u boji
- Bojenje površina



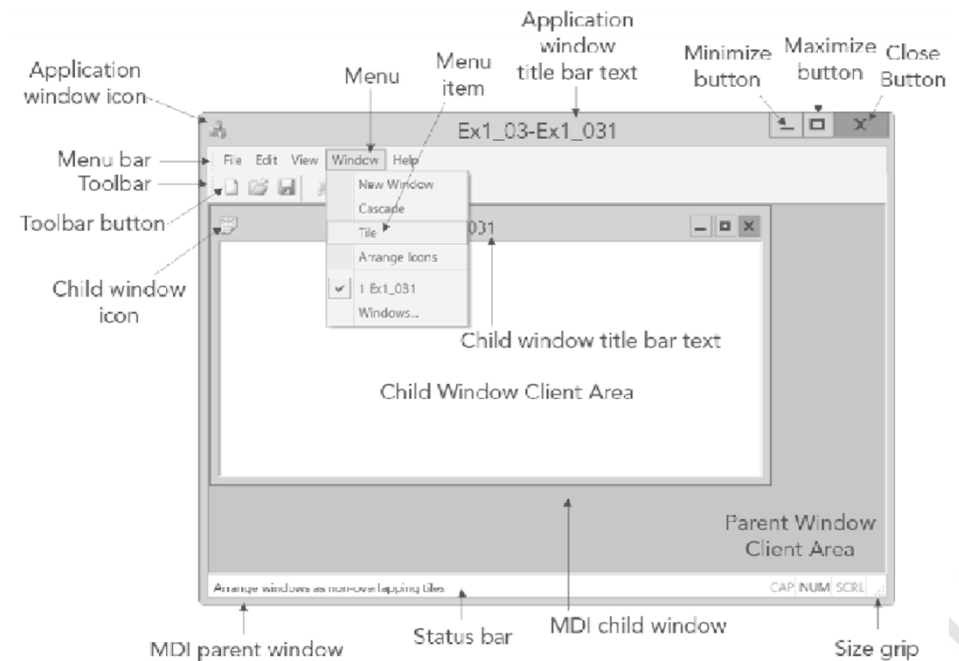
Windows API i biblioteka klasa MFC

- Razvoj grafički orijentisanog softvera interfejsa u jeziku C++ za operativni sistem *Windows* može se realizovati na različitim nivoima apstrakcije
 - direktnim pozivima funkcija operativnog sistema iz aplikativnog programa preko aplikativnog programskog interfejsa OS Windows (*Windows API*, Win API, Win 32/64).

To je najteži je i nasporniji način razvoja aplikacija, koji podrazumeva programsko kreiranje svih elemenata interfejsa pozivima funkcija operativnog sistema
 - upotrebom biblioteke klasa *Microsoft Foundation Classes (MFC)*, koja obuhvata sve neophodne funkcije Win API. Predstavlja viši nivo programiranja i znatno lakši način realizacije aplikacija

Elementi grafičkog prozora

- Osnovni element grafičkog korisničkog interfejsa je objekt *window* čije su najvažnije komponente:
 - veza s nadređenim i podređenim objektima-prozorima (*parent/child window*)
 - naslov (*title bar*)
 - meni linija
 - paleta alatki (*toolbar*)
 - radna površina (*client area*)
 - statusna linija (*status bar*)



Aplikacije operativnog sistema Windows

- Veliki deo koda *Windows* aplikacija bavi se obradom *događaja* koji su posledica akcija korisnika ili sistemskih događaja
- Operativni sistem beleži događaje u porukama koje stavlja u red čekanja odgovarajućeg programa za obradu događaja
 - program mora imati funkciju namenjenu rukovanju događajima, koja se obično naziva `WndProc()` ili `WindowProc()`
- Minimalni *Windows* program koji koristi Win API sastoji se od dve funkcije:
 - `WinMain()` - funkcija u kojoj počinje izvršavanje osnovnog programa i njegova inicijalizacija
 - `WindowProc()` - funkcija koju Windows poziva radi prenošenja poruka aplikacije

Aplikacije operativnog sistema Windows

- Funkcija *WinMain()* ekvivalent je funkcije *main()* konzolnih programa i ima prototip:

```
int WINAPI WinMain(HINSTANCE hInstance,  
HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow);
```

- Osnovna namena funkcije *WinMain()* je:
 - saopštava operativnom sistemu vrstu prozora potrebnu programu
 - kreira i inicijalizuje prozor programa
 - prihvata poruke sistema namenjene programu

- Prototip funkcije *WindowProc()* je:

```
LRESULT CALLBACK WindowProc(HWND hWnd, UINT message,  
WPARAM wParam, LPARAM lParam);
```

Aplikacije koje koriste biblioteku MFC

- Biblioteka *Microsoft Foundation Classes* (MFC) je skup klasa koje olakšavaju razvoj *Windows* desktop aplikacija u jeziku Visual C++
- Nazivi svih klasa u biblioteci MFC počinju slovom C, npr. **CDocument** ili **CView**
- Minimalna MFC aplikacija može se kreirati u dva koraka
 1. **File/New/Project**, izbor tipa projekta *Win32 Project*, a u sledećem dijalogu opcije *Windows Application* i *Empty project*
 2. Iz glavnog menija izbor **Project/Properties**, zatim panela *General*, u kome treba postaviti svojstvo projekta *Use of MFC* na vrednost *Use MFC in a Shared DLL*
- Zatim se kreira novi **cpp** fajl i unese izvorni kod aplikacije

Primer: Minimalna Windows aplikacija koja koristi MFC (1/2)

```
// Elementarni MFC program koji prikazuje prazan prozor
#include <afxwin.h>          // biblioteka klasa

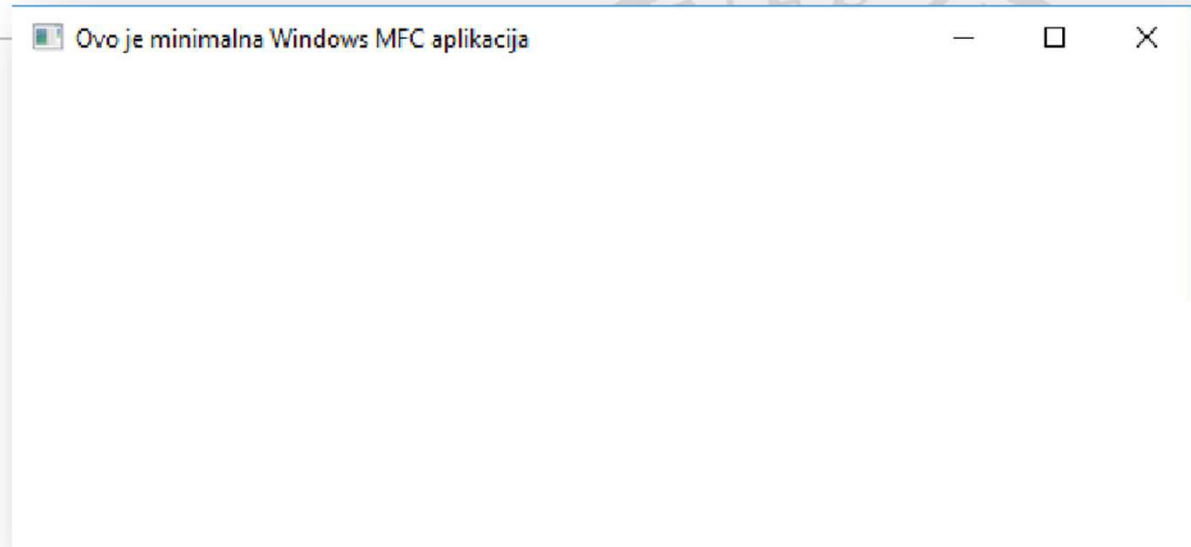
// Definisanje klase aplikacije
class CMojaApp:public CWinApp {
public:
    virtual BOOL InitInstance() override;
};

// Definicija klase Window
class CMojWnd:public CFrameWnd {
public:
    // Konstruktor klase Window
    CMojWnd() {
        Create(nullptr, _T("Ovo je minimalna Windows MFC aplikacija"));
    }
};
```

...

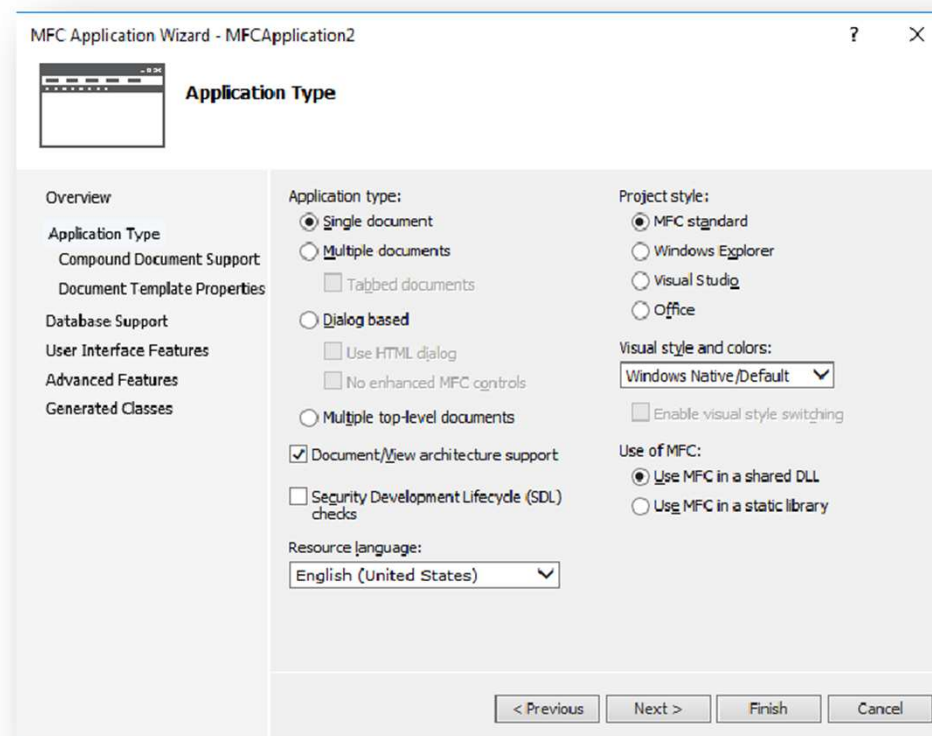
Primer: Minimalna Windows aplikacija koja koristi MFC (2/2)

```
// Funkcija za kreiranje instance glavnog prozora aplikacije
BOOL CMojaApp::InitInstance(void) {
    m_pMainWnd = new CMojWnd;           // Konstruiše objekt window...
    m_pMainWnd -> ShowWindow(m_nCmdShow); // ...i prikazuje prozor
    return TRUE;
}
// Definicija globalnog objekta aplikacije
CMojaApp MojaAplikacija;
```



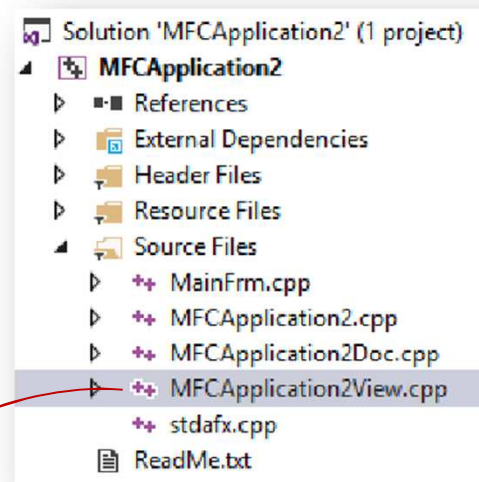
Kreiranje MFC aplikacije pomoću *Application Wizzarda*

1. Kreira se novi projekt tipa *MFC Application* koristeći *Application Wizzard*
2. Postave se opcije kao na slici i prihvate podrazumevajuće vrednosti do kraja dijaloga (*Finish*)



Kreiranje MFC aplikacije pomoću *Application Wizzarda*

- Visual Studio kreira folder MFC aplikacije s programskim kodom raspoređenim u više foldera
- Folder **Source Files** sadrži inicijalni kod MFC aplikacije
- Za prikaz grafike prilagođava se funkcija **OnDraw()** klase **CMFCApplication2View**



```
void CMFCApplication2View::OnDraw(CDC* /*pDC*/)
{
    CMFCApplication2Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: add draw code for native data here
}
```

```
void CMFCApplication2View::OnDraw(CDC* pDC)
{
    CMFCApplication2Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: add draw code for native data here
    pDC->MoveTo(50, 50); // postavljanje tekuće pozicije
    pDC->LineTo(50, 200); // vertikalna linija dole 150 jed.
    pDC->LineTo(150, 200); // horizontalna linija desno 100
    pDC->LineTo(150, 50); // vertikalna linija gore 150 jed.
    pDC->LineTo(50, 50); // horizontalna linija levo 100 jed.
}
```

Osnovna grafika u grafičkom prozoru

- Crtanje po radnoj površini vrši se pomoću generisane funkcije `OnDraw()` klase `CMFCGrafikaView`, npr.

```
void CMFCGrafikaView::OnDraw(CDC* /*pDC*/) {  
    CCMFCGrafikaDoc* pDoc = GetDocument();  
    ASSERT_VALID(pDoc);  
    if (!pDoc)  
        return;  
    // Dodavanje koda za crtanje, npr.  
    pDC->MoveTo(50, 50); // postavljanje tekuće pozicije  
}
```

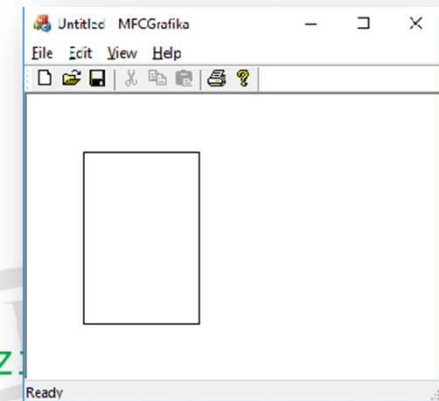
brisati oznake komentara

- Kad god aplikacija dobije poruku `WM_PAINT`, potrebno je ponovo izvršiti crtanje dela ili cele radne površine prozora, npr. zbog toga što je korisnik promenio veličinu prozora

Prikaz geometrijskih oblika

- Linija se može crtati pozivom metoda `MoveTo()` i `LineTo()` , koja na radnoj površini crta liniju do zadane pozicije, npr.

```
void CMFCGrafikaView::OnDraw(CDC* pDC) {  
    CMFCGrafikaDoc* pDoc = GetDocument();  
    ASSERT_VALID(pDoc);  
    if (!pDoc)  
        return;  
    pDC->MoveTo(50,50); // postavljanje tekuće poz.  
    pDC->LineTo(50,200); // vertikalna linija dole 150 jed.  
    pDC->LineTo(150,200); // horizontalna linija desno 100  
    pDC->LineTo(150,50); // vertikalna linija gore 150 jed.  
    pDC->LineTo(50,50); // horizontalna linija levo 100 jed.  
}
```



Grafika u boji

- Grafika koristi objekt **pero** (*pen*) za definisanje linije kojom se crta: boje, debljine i tipa (puna, tačkasta, isprekidana i sl.)
- Objekt se najjednostavnije definiše kreiranjem objekta klase **CPen** pomoću podrazumevajućeg konstruktora klase:

```
CPen pero;
```

- Objekt se inicijalizuje zadanim vrednostima svojstava pomoću funkcije **CreatePen()** klase **Cpen**:

```
BOOL CreatePen(int stil, int sirina, COLORREF boja);
```

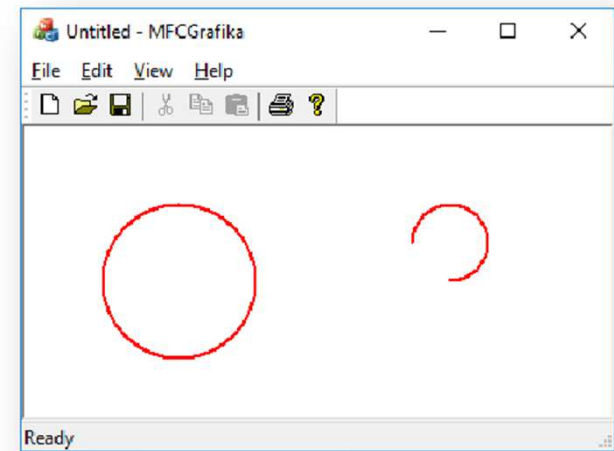
- Npr. **pero** za crtanje pune crvene linije kreira se kao objekt:

```
pero.CreatePen(PS_SOLID, 2, RGB(255,0,0));
```

- elipsa i kružnica crtaju se funkcijom **Ellipse()** kojoj se zadaju koordinate dva temena opisanog pravougaonika

Primer: MFC program za crtanje nekoliko geometrijskih figura

```
void CMFCApplication2View::OnDraw(CDC* pDC) {  
    CMFCApplication2Doc* pDoc = GetDocument();  
    ASSERT_VALID(pDoc);  
    if (!pDoc) return;  
    // Definisanje pera: crvena puna lin. sir. 2px  
    CPen pero;  
    pero.CreatePen(PS_SOLID, 2, RGB(255, 0, 0));  
    CPen* pStaroPero = pDC->SelectObject(&pero); // Promena pera  
    pDC->Ellipse(50,50,150,150); // Crtanje velike kruznice  
    // Definisanje okvira za manju kružnicu (pravougaonika)  
    CRect rect {250,50,300,100};  
    CPoint start {275,100}; // Početna tačka luka  
    CPoint end {250,75};    // Krajnja tačka luka  
    pDC->Arc(&rect,start, end); // Crtanje luka druge kružnice  
    pDC->SelectObject(pStaroPero); // Vraćanje starog pera  
}
```

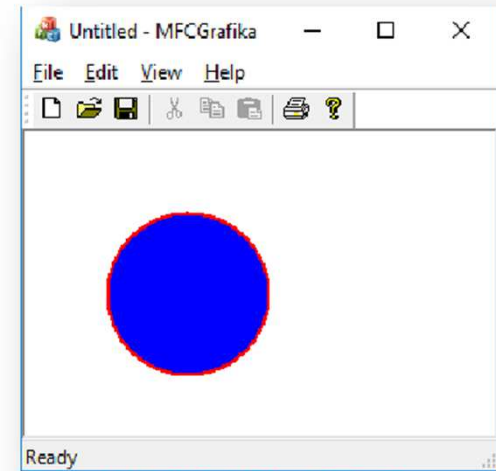


Bojenje površina

- Površine se boje definisanjem objekta `CBrush` koja sadrži *Windows* četkicu za bojenje površina (puna boja, raster ili neki uzorak)
 - definiše blok dimenzija 8x8 piksela, koji se ponavlja po površini koju treba popuniti
- Za bojenje punom bojom dovoljno je prilikom kreiranja četkice definisati boju, npr.
`CBrush` cetkica {RGB(0,0,255)}; // Plava boja popune
- Četkica se bira CDC funkcijom `SelectObject()`
`CBrush*` pStara {pDC->SelectObject(&cetkica)};
 - stara vrednost četkice se pamti radi mogućnosti restauriranja prethodnog stanja nakon upotrebe četkice

Primer: Obojen krug

```
void CMFCGrafikaView::OnDraw(CDC* pDC) {  
    CMFCGrafikaDoc* pDoc = GetDocument();  
    ASSERT_VALID(pDoc);  
    if (!pDoc) return;  
    // Definisanje pera: crvena puna linija 2px  
    CPen pero;  
    pero.CreatePen(PS_SOLID, 2, RGB(255, 0, 0));  
    CPen* pStaroPero = pDC->SelectObject(&pero);  
    // Definisanje boje popune: plava  
    CBrush cetkica{ RGB(0,0,255) };  
    CBrush* pStara{ pDC->SelectObject(&cetkica) }; // promena četkice  
  
    pDC->Ellipse(50, 50, 150, 150); // crtanje popunjene kružnice  
  
    pDC->SelectObject(pStara); // vraćanje prethodne četkice  
    pDC->SelectObject(pStaroPero); // vraćanje prethodnog pera  
}
```



5.3 Multiplatformska biblioteka za razvoj grafičkih aplikacija Qt

- Biblioteka **Qt** je u stvari aplikativni okvir (*framework*) za razvoj multiplatformskih aplikacija s grafičkim interfejsom (GUI)
- Prenosivost multiplatformskih aplikacija koje koriste okvir Qt omogućava se zasebnim prevođenjem za svaku platformu
 - razvija se samo *jedna verzija* aplikacije i prevodi za svaku platformu
- Qt je popularn, ali je njena upotreba znatno složenija nego upotreba većine C++ biblioteka
 - ima niz pomoćih alata, npr. *Qt Creator*, integrisano razvojno okruženje koje pojednostavljuje razvoj aplikacija s grafičkim interfejsima (GUI)
- Nije besplatna, osim za razvoj softvera otvorenog koda
 - download folder <https://www.qt.io/download>

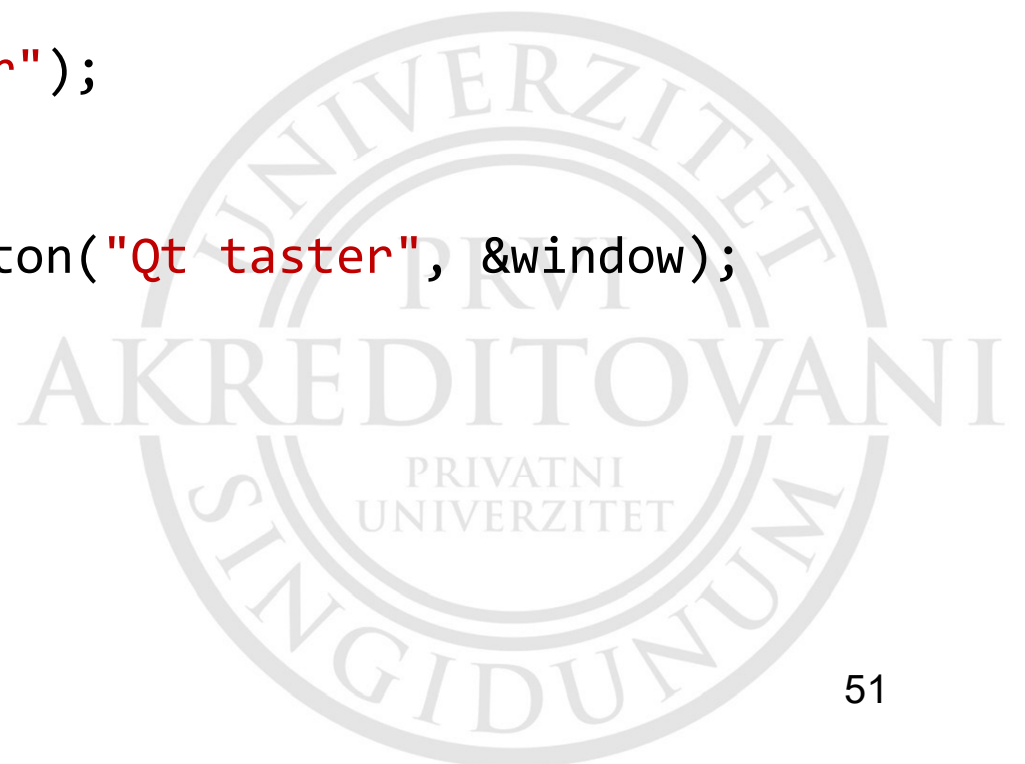
Ilustracija: Jednostavna Qt aplikacija s grafičkim interfejsom (GUI) [8]

```
#include <QtWidgets>
// Primer upotrebe biblioteke Qt
// -- kreiranje grafičkog prozora s jednim tasterom

int main(int argc, char** argv) {
    QWidget window;
    window.resize(120, 100);
    window.setWindowTitle("Qt prozor");
    window.show();

    QPushButton* btn = new QPushButton("Qt taster", &window);

    return app.exec();
}
```



Literatura

1. Branović I., *Osnove objektno orijentisanog programiranja: C++*, Univerzitet Singidunum, 2013
2. Stroustrup B., *The C++ Programming Language*, 4th Ed, Addison Wesley, 2013
3. Horton I., Van Weert P., *Beginning C++ 20*, 6th Edition, Apress, 2020
4. Horton I., *Beginning C++*, Apress, 2014
5. Horton I., *Beginning Visual C++ 2013*, Wox/John Wiley&Sons, 2014
6. Horton I., *Using the C++ Standard Template Libraries*, Apress, 2015
7. Galowitz J., *C++ 17 STL Cookbook*, Packt, 2017
8. Grigoryan V., Wu S., *Expert C++: Become a proficient programmer by learning coding best practices with C++17 and C++20*, Packt Publishing, 2020
9. Veb izvori
 - <http://www.cplusplus.com/doc/tutorial/>
 - <http://www.learncpp.com/>
 - <http://www.stroustrup.com/>
10. Knjige i priručnici za *Visual Studio* 2010/2012/2013/2015/2017/2019