Napredne tehnike programiranja

Vježba 1

Napredne tehnike programiranja

- Studenti su dužni prisustvovati na najmanje 80% nastavnih sati iz laboratorijskih vježbi
- Bodovi na vježbama:
  - 13 vježbi (40 bodova)
  - 2 kolokvija (2*30 bodova)

Napredne tehnike programiranja

Kolokviji:

- Sastoje se od rješavanja teorijskih i praktičnih zadataka otvorenog tipa
- Kolokvije mogu polagati samo studenti koji su u tekućoj akademskoj godini prvi put upisali kolegij
- Prepisivanje na kolokvijima je zabranjeno te povlači disciplinsku odgovornost

Napredne tehnike programiranja

- Pismeni dio ispita je položen ukoliko je student na svim kontrolnim zadaćama, kolokvijima i pedavanjima ostvario bar 50% bodova
- Ocjena kontinuiranog praćenja vrijedi (samo jednom) umjesto pismenog dijela ispita na prva dva ispitna roka odmah po završetku semestra
- Studenti koji ne polože ili ne pristupe usmenom dijelu ispita u navedena dva ispitna roka nadalje su obvezni polagati i pismeni dio ispita

Napredne tehnike programiranja

- Sve aktivnosti koje se ocjenjuju (zadaci na vježbama i kolokviji) se pišu u Verifikatoru
- Verifikator je dostupan na e-učenju
- Do slijedećeg tjedna svi studenti trebaju instalirati i isprobati Verifikator

Programski jezik C++

asm	auto	bool	break
case	catch	char	class
const	const_cast	continue	default
delete	do	double	dynamic_cast
else	enum	explicit	export
extern	false	float	for
friend	goto	if	inline
int	long	mutable	namespace
new	operator	private	protected
public	register	reinterpret_cast	return
short	signed	sizeof	static
static_cast	struct	switch	template
this	throw	true	try
typedef	typeid	typename	union
unsigned	using	virtual	void
volatile	wchar_t	while	

Naredba #include

- #include je pretprocesorska naredba pomoću koje u naš program uključujemo dodatne biblioteke
- Uz sam jezik dolazi mnoštvo gotovih funkcija, klasa i predložaka koje možemo koristiti u našim programima
- Primjer:

```
#include "nesto.h" // iz tekućeg foldera
#include <iostream> // iz standardnog imenika
```

Standardna biblioteka C++ jezika

<algorithm>
<bitset>
<chrono>
<codecvt>
<complex>
<exception>
<functional>
<initializer_list>
<iterator>
<limits>
<locale>
<memory>
<new>
<numeric>
<random>
<ratio>
<regex>

<string>
<system_error>
<tuple>
<typeindex>
<typeinfo>
<type_traits>
<utility>
<valarray>
<fstream>
<iomanip>
<ios>
<iosfwd>
<iostream>
<istream>
<ostream>
<sstream>
<streambuf>

<atomic>
<condition_variable>
<future>
<mutex>
<thread>

STL Spremnici (Container class templates)

Sequence containers:

- **array** Array class (class template)
- ***vector*** *Vector* (class template)
- **deque** Double ended queue (class template)
- **forward_list** Forward list (class template)
- **list** List (class template)

Container adaptors:

- **stack** LIFO stack (class template)
- **queue** FIFO queue (class template)
- **priority_queue** Priority queue (class template)

Associative containers:

- **set** Set (class template)
- **multiset** Multiple-key set (class template)
- **map** Map (class template)
- **multimap** Multiple-key map (class template)

Unordered associative containers:

- **unordered_set** Unordered Set (class template)
- **unordered_multiset** Unordered Multiset (class template
- **unordered_map** Unordered Map (class template)
- **unordered_multimap** Unordered Multimap (class template)

Funkcije

- Program napisan u C++ programskom jeziku sastoji se od funkcija
- Obavezna je funkcija `main`, a ostale pišemo po potrebi
- Funkcija je izdvojeni dio koda koji prima parametre, obrađuje ih i vraća rezultat
- Funkcija ne mora imati prarametre i ne mora vratiti rezultat (`void`)
- Funkciju u programu treba deklarirati (opisati) i definirati (napisati)

Povratni tip funkcije

- Povratni tip funkcije je tip rezultata koji funkcija vraća (npr. ako funkcija kao rezultat vraća cijeli broj njen povratni tip je int)
- Povratni tip funkcije može biti:
  - ugrađeni tip (npr. int ili double)
  - složeni tip poput int& ili double*
  - korisnički definirani tip (klasa, struktura)
  - void (ako funkcija ne vraća rezultat)
- Primjeri:

```
list<char> foo_bar();
int *foo_bar();
int[10] foo_bar(); //greska: polje ne moze biti povratni tip
```

Proslijeđivanje argumenata

- Zadatak: Napišite funkciju zamjena()koja zamjenjuje vrijednosti dva dobivena cjelobrojna argumenta.
- Rješenje1:

```
void zamjena(int v1, int v2) {
        int tmp = v2;
        v2 = v1;
        v1 = tmp;
}
```

- Rješenje 2:Implementacija preko pokazivača

```
void zamjena(int *v1, int *v2) {
        int tmp = *v2;
        *v2 = *v1;
        *v1 = tmp;
}
```

- Rješenje 3:Implementacija preko referenci

```
void zamjena(int& v1, int& v2) {
        int tmp = v2;
        v2 = v1;
        v1 = tmp;
}
```

Proslijeđivanje argumenata po vrijednosti

- Funkcija:

```
void zamjena(int v1, int v2) {
            int tmp = v2;
            v2 = v1;
            v1 = tmp;
      }
```

- Glavni program:

```
int main() {
      int a=3, b=5;
      zamjena(a, b);
      cout <<a<<" "<<b<<endl;//ispisuje "3 5"
}
```

Proslijeđivanje argumenata pokazivačem

- Funkcija:

```
void zamjena(int *v1, int *v2) {
            int tmp = *v2;
            *v2 = *v1;
            *v1 = tmp;
      }
```

- Glavni program:

```
int main() {
      int a=3, b=5;
      zamjena(&a, &b);
      cout <<a<<" "<<b<<endl;//ispisuje "5 3"
}
```

Proslijeđivanje argumenata preko referenci

- Funkcija:

```
void zamjeni(int& v1, int& v2) {
            int tmp = v2;
            v2 = v1;
            v1 = tmp;
      }
```

- Glavni program:

```
int main() {
      int a=3, b=5;
      zamjena(a, b);
      cout <<a<<" "<<b<<endl;//ispisuje "5 3"
}
```

Proslijeđivanje argumenata preko referenci

- Situacije u kojima je prikladno proslijeđivati parametre po referenci:
  - kada u funkciji želimo promijeniti vrijednost dobivenih argumenata
  - kada imamo dodatne rezultate koje želimo vratiti pored same povratne vrijednosti
  - kada u funkciju prosljeđujemo "velike" objekte

Proslijeđivanje argumenata preko referenci

- Kad samo želimo izbjeći kopiranje stvarnog argumenta, formalni argument treba deklarirati kao konstantnu referencu.
- Kako je moguće konstantnu referencu inicijalizirati nekonstantnim objektom (const je samo obećanje da referenca neće služiti za mijenjanje objekta), funkcija koja deklarira konstantnu referencu može uvijek uzeti nekonstantan argument (koji, naravno, ne može u svom tijelu promijeniti).

```
void foo(const tip& x) {
    // x ovdje ne moze biti mijenjan
}
```

Vraćanje povratne vrijednosti

- Ako je funkcija deklarirana kao void, koristimo naredbu `return;` bez argumenta
- Funkcija s povratnom vrijednošću nužno mora vratiti vrijednost naredbom oblika `return expr;`
- Primjer:

```
void d_copy(double *src, double *dst, int sz) {
      if ( !src || !dst )
            return;
      if ( src == dst )
            return;
      if ( sz == 0 )
            return;
      for ( int i = 0; i < sz; ++i )
      dst[i]=src[i];//nije potreban eksplicitan return
}
```

inline funkcije

- Ako ispred imena funkcije stavimo ključnu riječ inline prevoditelj će na svako mjesto u kodu gdje se poziva funkcija umetnuti njen kod
- Prevoditelj može ignorirati ovu oznaku
- Primjer:

```
inline double zbroj(float x, float y) {
    return x + y;
}
```

inline funkcije

- Ako definiciju funkcije stavimo unutar strukture ili klase ona je po defaultu inline
- Kompajler će pokušati cijelo tijelo funkcije zalijepiti na mjesto poziva

```
struct x{
      ...

      double zbroj(float x, float y) {
            return x + y;
      }
};
```

Strukture podataka

- Omogućavaju nam da napravimo tip podatka koji odgovara našem problemu
- Sastoji se od niza podataka ne nužno istog tipa, koji su smješteni u memoriji na uzastopnim memorijskim lokacijama
- Možemo ih koristiti na isti način kao i standardne tipove podataka

Primjer

```
struct student {
    string ime;
    int godina;
    bool status;
};

student s={"Pero",2,true};
```

Pristup elementima strukture

```
struct student {
      string ime;
      int godina;
      bool status;
};

student s={"Pero",2,true};
cout<<s.ime<<endl;
s.status=false;
```

Elementima strukture pristupam preko njihovog imena

Pokazivač na strukturu

```
struct student {
      string ime;
      int godina;
      bool status;
};

student *s;
s=new student;
cout<<(*s).ime<<endl; //ISPRAVNO
*s.status=false;      // GREŠKA
delete s;
```

Operator ->

```
struct student {
      string ime;
      int godina;
      bool status;
};

student *s;
s=new student;
cout<<s->ime<<endl; // -> koristimo umjesto .
s->status=false;    // -> koristimo umjesto .
delete s;
```

Primjer – korištenje strukture

```
struct tocka {
    int x, y;
};
void ispisiTocku( tocka P );
tocka simetricnaTocka( tocka P );
float udaljenost( tocka P,tocka Q);
```

Primjer – korištenje strukture

- **nastavak**

```
void ispisiTocku( tocka P )
{
cout << "(" << P.x << ", ";
cout << P.y << ")";
}
tocka simetricnaTocka( tocka P ) { ...kod... }
float udaljenost( tocka P, tocka Q ) { ..kod.. }
```

Primjer – korištenje strukture

- **main.cpp** koristimo tip tocka i funkcije

```
#include <iostream>
#include "tocka.h"
using namespace std;
int main( void )
{
	tocka P, Q;
	cin >> P.x >> P.y >> Q.x >> Q.y;
	cout << "d(P,Q)=" << udaljenost( P, Q );
	tocka simP = simetricnaTocka( P );
	ispisiTocku( simP );
	return 0;
}
```

Primjer

- napišite datoteku stack.h, da slijedeći kod radi:

```
#include "stack.h"
int main( void ){
  stack S;
  makeNull( S );
  push( S, 3 );
  push(S, 5 );
  int a = top( S );
  cout<<a<<endl;
  pop( S );
  a = top( S );
  cout<<a<<endl;
  return 0;
}
```

Stack.h

```
struct stack {
        int vrh, podaci[100];
};
void makeNull( stack &s  );
void push( stack &s,int  x);
int pop( stack &s );
int top( stack  s);
void makeNull( stack &s )
        { s.vrh = 0; }
void push( stack &s,int x )
        {s.podaci[s.vrh]=x; ++ s.vrh; }
int pop( stack &s )
        {--s.vrh; return s.podaci[s.vrh]; }
int top( stack s )
        { return s.podaci[s.vrh-1]; }
```

Zadatak

- Napišite datoteku pravokutnik.h strukture pravokutnik
- Kod mora biti takav da se donji klijentski program uspješno kompajlira:

```
#include "pravokutnik.h"
int main()
{
        pravokutnik P={2,4}; //kreira pravokutnik sa stranicama 2 i 4
        cout<<povrsina(P)<<endl; //vraca povrsinu pravokutnika
        povecaj_za(P,3); //povecava stranice pravokutnika za 3
        smanji_za(P,2); //smanjuje stranice pravokutnika za 2
        ispisi(P);  //ispisuje sve podatke o pravokutniku
        return 0;
}
```