ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

ПОДАТОЧНИ СТРУКТУРИ И ПРОГРАМИРАЊЕ 2023/2024

АУДИТОРИСКИ ВЕЖБИ 3

РАЗЛИКИ ПОМЕЃУ C И C++

Вовед

C++ вклучува многу карактеристики на јазикот C дополнети со неговите средства за објектно-ориентирано програмирање (ООП), ветувајќи значително зголемена продуктивност на трудот на програмерите, како и зголемување на квалитетот и можности за повторното искористување на програмите.

Јазикот C++ бил развиен во Bell лабораториите и во почетокот бил наречен „C со класи“. Името C++ вклучува во себе операција за зголемување на јазикот C (++) и укажува на тоа дека ++ е верзија на C со проширени можности. C++ се јавува како проширено множество на C и затоа програмерите можат да користат C++ компајлер за преведување на постоечките програми напишани во C и на таков начин полека да ги надградат во C++ програми.

Влез/Излез текови во C++

C++ поседува друг начин за работа со стандардниот влез и излез (во споредба со функциите `printf` и `scanf`). Така на пример, кодот:

```
printf("Vnesi nova vrednost : ");
scanf("%d", &vrednost);
printf("Novata vrednost e : %d\n", vrednost);
```

во C++ се запишува како:

```
cout << "Vnesi nova vrednost : ";
cin >> vrednost;
cout << "Novata vrednost e : " << vrednost << '\n';
```

Притоа препрорачливо е изолираните `'\n'`да се заменат со `endl`:

```
cout << "Novata vrednost e : " << vrednost << endl;
```

Во првата операција се користи стандардниот излезен тек `cout` и операцијата `<<` (операција за пуштање во текот, означува „прати во“). Операторот се чита како: "Vnesi nova vrednost" се испраќа во излезниот тек `cout`. Треба да се напомене дека операцијата предава во текот така што врши постепено празнење со поместување во лево.

Во втората операција се користи стандардниот влезен тек cin и операцијата `>>` (вадење од текот, означува „извади од“). Операцијата се чита како: "Извади од влезниот тек вредност за променливата vrednost". Треба да се напомене дека операцијата вади од текот така што врши постепено вадење со поместување во десно, доколку она што е во текот одговара на прочитаниот тип. Операциите за праќање и вадење од тек за разлика од функциите `printf` и `scanf` не бараат форматирање со спецификација на типот на влезните и излезните податоци. C++ во многу случаи, автоматски знае кои типови на променливи учествуваат во операцијата и автоматски ја прилагодува операцијата според типовите на операндите.

Треба да се обрне внимание на тоа дека при внесување на вредност од стандарден влез, пред променливата не стои знак за адреса &, како што се бара кај `scanf`.

За организација на влезно/излезниот тек, програмите во C++ задолжително ја вклучуваат датотеката `iostream`, а потребна е и задолжителна употреба на просторот на имиња `std`.

Пример за прост влезен/излезен тек:

```
#include <iostream>
using namespace std;
int main()
{
	cout << "Vnesi gi tvoite godini:";
	int moi;
	cin >> moi;
	cout << "Vnesi gi godinite na prijatel:";
	int prijatel;
	cin >> prijatel;
	if (moi > prijatel)
		cout << "Ti si postar." << endl;
	else
		if (moi < prijatel)
			cout << "Prijatelot e postar.\n";
		else
			cout << "Ti i prijatelot imate ist broj na godini." << endl;
	return 0;
}
```

Декларација во C++

Променливите во C++ може да се декларираат во било кој дел од програмата, се додека нивната декларација е пред употреба на променливата во некоја наредба.

Пример:

```
for (int i = 0; i <= 5; i++)
	cout << i << endl;
```

Областа на делување на локалните променливи почнува од декларацијата и се протега до крајот на блокот во кој припаѓа означен со }. Декларација не може да се врши во делот за услов кај структурите за повторување `while`, `do` /`while`, `for` или кај `if`.

Унарна операција за разрешување на областа на дејство (scope оператор)

Во C и C++ е можно декларирање на локални и глобални променливи со исто име. Во C, секоја локална променлива има област на делување и секоја употреба на името на променливата се однесува на вредноста на локалната променлива. Глобалната променлива не може да се види во областа на делување на локалните променливи.

Во C++ постои унарна операција за разрешување на областа на дејствување **(::)** која дозволува пристап до вредноста на глобалната променлива во областа на дејство на истоимена локална променлива. Унарната операција не може да се искористи за пристап до променливи со исто име во внатрешен блок. Кон глобалните променливи можно е да се обрати непосредно без операција за разрешување на областа на дејствување во областите каде не постои локална променлива со исто име.

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

float vrednost = 1.2345;

int main()
{
	int vrednost = 7;
	cout << "lokalna promenliva = " << vrednost << "\n globalna promenliva = " <<
::vrednost << '\n';
	return 0;
}
```

Inline функции во C++

Реализацијата на програмата преку поголем број на функции е добра од гледна точка на лесна поправка на кодот, но употребата на вакви функции е поврзана со определени додатни временски трошоци при користењето на програмата.

Во C++ постојат функции кои ги намалуваат овие трошоци особено кога станува збор за мали функции. Службениот збор `inline` кој се поставува пред типот на излезот на функцијата му препорачува на компајлерот на местата каде се користи функцијата да генерира копија од кодот сместен во функцијата (доколку тоа е можно), со што се избегнува повикување на функцијата. Компромисот се состои во тоа што наместо една копија на функцијата во која се предава управувањето при секој повик на функцијата, во програмата се поставуваат повеќе копии на кодот на таа функција. Компајлерот може да ја игнорира `inline` препораката и обично и така постапува со исклучок на мали функции.

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

inline float volumen(const float s) { return s*s*s; }
int main()
{
	cout << "Vnesi ja stranata na kockata : ";
	float strana;
	cin >> strana;
	cout << "Volumenot na kocka so strana " << strana << " e " << volumen(strana) << '\n';
	return 0;
}
```

Аргументи по референца

Во C при секое обраќање кон функцијата аргументите се пренесуваат по вредност. Пренесувањето на аргументите по референца во C се имитира со помош на пренесување на покажувач кон објект и пристап до тој објект преку разменување на покажувачот во повикувачката функција. C++ го отстранува овој недостаток на C, дозволувајќи пренесување на аргументи по референца.

Аргумент по референца се јавува како **псевдоним** (синоним) на соодветната променлива. Предавањето на аргумент по пат на референца се врши со поставување на знакот & по типот на аргументот, на истото место како и * за покажувачите. Овој знак, &, се употребува само при наведувањето на аргументот при дефинирање на аргументите на функцијата.

При повикување на функцијата се пренесува променливата која потоа автоматски се пренесува по референца. По ова референцирање на локално име, параметрите во таа функција се однесуваат како излезни променливи.

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

int kvadratPoVrednost(int);
void kvadratSoPokazhuvach(int *);
void kvadratPoReferenca(int &);

int main()
{
	int x = 2, y = 3, z = 4;
	cout << "x = " << x << "pred kvadratPoVrednost" << endl
		<< "Vrednosta vratena od kvadratPoVrednost : " << kvadratPoVrednost
		<< "\nx = " << x << "po povikuvanje na kvadratPoVrednost\n\n ";

	cout << "y = " << y << "pred kvadratSoPokazhuvach" << endl;
	kvadratSoPokazhuvach(&y);
	cout << "y = " << y << "po povikuvanje na kvadratSoPokazhuvach\n ";
```

```
	cout << "z = " << z << "pred kvadratPoReferenca" << endl;
	kvadratPoReferenca(z);
	cout << "z = " << z << "po povikuvanje na kvadratPoReferenca" << endl;
	return 0;
}
int kvadratPoVrednost(int a)
{
	return a *= a;
}
void kvadratSoPokazhuvach(int *bPtr)
{
	*bPtr *= *bPtr;
}
void kvadratPoReferenca(int &cRef)
{
	cRef *= cRef;
}
```

Излез:

```
x=2 pred kvadratPoVrednost
Vrednosta vratena od kvadratPoVrednost: 4
x=2 po povikuvanje na kvadratPoVrednost
y=3 pred kvadratSoPokazhuvach
y=9 po povikuvanje na kvadratSoPokazhuvach
z=3 pred kvadratPoReferenca
z=9 po povikuvanje na kvadratPoReferenca
```

Референците можат да послужат како псевдоними за други променливи во внатрешноста на некои функции. Како пример, земете го кодот:

```
int count = 1;    // deklaracija na celobrojna promenliva count
int &c = count;   // kreira c vo uloga na psevdonim za count
++c;              // se zgolemuva vrednosta na promenlivata count so koristenje na nejziniot
psevdonim
```

Ваквите променливи по референци треба да се иницијализираат при декларацијата и не можат да бидат преназначени за псевдоними на други променливи. Сите операции кои се изведуваат врз променливите по референци се однесуваат исто како да се изведуваат врз самата променлива кон која се референцира.

```
int x = 3, &y = x;
cout << "x = " << x << '\n' << "y = " << y << '\n';
y = 7;
cout << "x = " << x << '\n' << "y = " << y << '\n';
```

Излез:

```
x=3 y=3
x=7 y=7
```

Референците не може да се разменуваат со примена на операции за индиректно адресирање. Исто така не смее да се користат покажувачи на референци, изедначување на референци, адресирање на референци.

Функциите можат да вратат покажувачи или референци, но ова може да претставува опасност. При враќање на покажувачи или референци кон променливи, променливата кон која се враќа референца или покажувач треба да биде декларирана како статичка. Во спротивно, покажувачите или референците ќе се однесуваат на променлива со автоматска класа на меморијата, која се уништува при

завршувањето на функцијата. Таквите променливи се јавуваат како неопределени и однесувањето на програмата може да биде непредвидливо.

Аргументи со подразбирани (предефинирани) вредности

Често при повикување на функција може да се придава особено значење на аргументите. Програмерот може да укаже дека таквиот вид аргументи се јавуваат како аргументи со подразбирани вредности за кои се зададени соодветни вредности. Ако таков аргумент се испушти при повикување на функцијата, во функцијата автоматски се употребува подразбираната вредност.

Аргументите со предефинирани вредности мора да бидат последни аргументи (сосема десно) во списокот на аргументи на функцијата. При повикување на функција со два или повеќе аргументи со предефинирани вредности ако испуштените параметри не се јавуваат во списокот со аргументи крајно десно, тогаш сите останати аргументи треба да бидат испуштени. Дозволено е да се испуштат само последните неколку аргументи со подразбирана вредност при повикот на функцијата.

```
inline int Volumen(int dolzhina = 1, int shirina = 1, int visina = 1)
{
	return dolzhina * shirina * visina;
}
int main()
{
	cout << "Predefiniraniot volumen e : " << Volumen() << endl
		<< "Volumenot na kutija so dolzhina 10 e : " << Volumen(10) << endl
		<< "Volumenot na kutija so dolzhina 10 i shirina 5 e : "
		<< Volumen(10, 5) << endl
		<< "Volumenot na kutija so dolzhina 10, shirina 5 i visina 2 e : "
		<< Volumen(10, 5, 2) << endl;
	return 0;
}
```

Преоптоварување на функции

C++ дозволува дефинирање на различни функции со исто име. Ова повеќекратно користење на имиња се нарекува преоптоварување на функции. Во општ случај ако две функции во една програма имаат исто име, C++ нема да може да направи разлика меѓу нив. Сепак, името на функцијата го вклучува и бројот и типот на нејзините аргументи, но не и типот на излезот. Поради тоа следните функции не се исти:

```
void funkcija(void)
void funkcija(int n)
void funkcija(double d)
void funkcija(int n1, int n2)
```

Од друга страна пак, следните функции за C++ се исти:

```
int funkcija(int n)
double funkcija(int n)
```

Пример:

```
int Kvadrat(int x) { return x * x; }
double Kvadrat(double y) { return y * y; }

int main()
{
	cout << "kvadrat na 7 e " << Kvadrat(7) << endl
		<< "kvadrat na 7.5 e " << Kvadrat(7.5) << endl;
	return 0;
}
```

Typedef и Struct so функции

Користење на typedef е дозволено, но не и неопходно при дефинирање на структури или набројувачки множества (enum) во C++ .

```
struct Chovek
{
	char ime[80], adresa[90];
	double plata;
};

int main()
{
	Chovek vraboteni[50];
	Chovek rakovoditel;
	rakovoditel.ime = "Aleksandar";
	cout << "Imeto na rakovoditelot e" << rakovoditel.ime << endl;
}
```

Функции како членови на структура

Во C++ е дозволено дефинирање на функција како дел од некоја структура (во согласност е и со дефиницијата на објект: пакет кој содржи атрибути (податоци) и однесување (функции асоцирани со објектот)).

```
struct Chovek
{
	char ime[80], adresa[80];
	void print(void);  //декларација на функцијата за печатење на името и адресата на еден човек
};
```

Функцијата `print()` е само декларирана, кодот на функцијата се наоѓа на друго место.

Големината на структурата (`sizeof`) е определена САМО од големината на податочните елементи. Функциите кои се декларирани во неа не влијаат на нејзината големина. Компајлерот го имплементира ова однесување со тоа што функцијата `print()` е позната само во контекст на структурата `Chovek`.

Пристапот до функција дефинирана како дел од структура е идентичен како и пристапот до податочен елемент од структура, т.е. се користи операторот (**.**). Кога се употребува покажувач кон структура се употребува **->**. Оваа синтакса дозволува да се користи исто име на функција во повеќе структури.

```
struct Chovek
{
	char ime[80], adresa[90];
	double plata;
	void print();  //функцијата е само декларирана
};
void Chovek::print() //тука ја дефинираме функцијата и мора да нагласиме на која структура припаѓа
{
	cout << "Imeto na vraboteniot e : " << ime <<
		"a, negovata adresa e : " << adresa << endl; /* функцијата може директно да пристапува до членовите на структурата во која и самата е дефинирана */
}

int main()
{
	Chovek rakovoditel = { "Misho", "Prvomajska 55", 30000 };
	rakovoditel.print();
```

```
	return 0;
}
```

Референци (продолжение)

Референците во C++ се имплементирани преку покажувачи, односно компајлерот ги гледа референците како покажувачи, а програмерот се ослободува од синтаксата – индирекциите кои одат со покажувачите.

```
void by_reference(int &valr)
{
	valr += 5;
}
```

Референците вообичено се користат кога очекуваме промена на аргументот во функцијата. Меѓутоа, тие имаат важна улога и во случаи во кои нема промена на аргументот во функцијата, како во случај кога аргументот на функцијата е структура или кога функцијата враќа структура (се избегнува копирање на целата структура што се случува при пренос на аргументи по вредност). Во случај да имаме аргумент структура и не се очекува нејзина измена во кодот на функцијата, вообичаено е (и треба) да се користи зборот `const`.

```
struct Chovek
{
	char ime[80], adresa[90];
	double plata;
};
Chovek vraboteni[50]; //глобална променлива
void print(Chovek const &p) // пренесуваме константна референца
{
	cout << "Imeto na vraboteniot e : " << p.ime <<
		"a, negovata adresa e : " << p.adresa << endl;
}
Chovek const &zemiVraboten(int index) /*враќаме референца на оној вработен чиј реден број во
низата vraboteni е index */
{
	return (vraboteni[index]); // враќа референца, а не копија од vraboteni[index]
}

int main()
{
	Chovek rakovoditel;
	print(rakovoditel); //променливата rakovoditel нема да се промени во функцијта
	print(zemiVraboten(5)); // се пренесува референца, а не копија
	return 0;
}
```

Употреба на const наместо #define

Со употреба на const се специфицира дека вредноста на променливата или аргументот не смее да се менува. На пример во следниот код наредбата за промена на променливата vrednost = 4 ќе врати грешка

```
int main()
{
	int const ival = 3;
	// int konstanta ival e inicijalizirana na 3
	ival = 4; // vraka error message
	return 0;
}
```

За разлика од C, во C++ променливите кои се дефинирани како константи може да употребуваат при спецификација на големината на низа.

Пример:
```
int const size = 20;
char buffer[size]; // pole od 20 znaci
```

Употреба на const со покажувачи

```
char const *buf;
```

buf е покажувач кон знаци кои се дефинирани како константи, т.е. не смеат да се менуваат, додека покажувачот buf смее да се менува. Следствено, buf ++ е дозволена, а *buf='a'е недозволена операција. За разлика од претходната дефиниција, наредбата:

```
char *const buf;
```

дефинира константен покажувач кој не смее да се менува, но податокот од типот знак кон кој покажува buf смее да се менува. И конечно,

```
char const *const buf;
```

означува дека ниту покажувачот ниту податокот кон кој покажува тој не смеат да се променат. Дефинициите или декларациите во кои се користи зборчето const се читаат почнувајќи од името на променливата (односно името на функцијата) кон типот на променлива/функција. Следствено, последната наредба се чита како "buf е константен покажувач кон константа знак".

Меѓутоа, многу често се греши со користење на обратното (неточно) правило: зборчето const претходи на тоа што не треба да се менува. Според оваа интерпретација, наредбата: char const *buf; би "означувала" дека покажувачот не смее да се менува, бидејќи const претходи на ознаката за покажувач *. Всушност, дадената наредба компајлерот ја интерпретитра како: вредноста (знаците) покажувани од buf не се менуваат.

Следствено, следниот програмски код ке биде одбиен од компајлерот:

```
int main()
{
	char const *buf = "hello";
	buf++; // prifateno od kompajlerot
	*buf = 'u'; // ne se prifaka od kompajlerot
	return 0;
}
```

CAST – Нова синтакса

Во C се користеше следната синтакса за кастирање на променливи:

```
(typename)expression
(float) broitel;
```

Во C++ иако е дозволена и претходната нотација, новиот начин на нотација е:

```
typename(expression)
float(broitel);
```

ЗАДАЧИ:

1. Да се напише програма која ќе ја пресмета разликата во површината на квадрат и површината на круг зададени со дијаметар и страна со еднакви должини. Напишете најефикасно решение!

```
#include <iostream>
using namespace std;
float pi = 3.14; //globalna promenliva

inline float Ploshtina(float dijametar, float pi) { return (dijametar / 2) * (dijametar / 2) *
pi; }
inline float Ploshtina(float strana) { return strana * strana; }

int main()
{
	float dolzina;
	float pi = 3.14215; //lokalna promenliva

	cout << "Vnesi ja dolzinata: " << endl;
	cin >> dolzina;

	float &strana = dolzina;  //drugo ime za ista promenliva
	float &dijametar = dolzina;

	cout << "Razlikata vo povrsinite na dvete figuri za PI="
		<< pi << " e:"
		<< Ploshtina(strana) - Ploshtina(dijametar, pi) << endl;
	cout << "Razlikata vo povrsinite na dvete figuri za PI="
		<< ::pi << " e:" << Ploshtina(strana) - Ploshtina(dijametar, ::pi)
		<< endl;
	return 0;
}
```

2. На една туристичка агенција и е потребна програма за менаџирање на слободните соби во хотелите со кои соработува. За секој хотел се знае неговото име, број на соби, низа од соби и локација. За секоја соба се знае нејзиниот број, типот на собата (*еднокреветна, двокреветна, супериор, делукс, апартман*), податок дали е зафатена и нејзината цена. Агенцијата соработува со 5 хотели.

```
#include <iostream>
#include <string>

using namespace std;

enum tipsoba { ednokrevetna, dvokrevetna, superior, delux, apartman };
char *tipNiza[5] = { "Ednokrevetnata", "Dvokrevetnata", "Superior", "Delux", "Apartman" };

struct Soba
{
	int broj;
	tipsoba tip;
	bool zafatena;
	int cena;
	void init(int br = 1, tipsoba t = ednokrevetna, bool z = 0, int c = 1200)
	{
		broj = br;
		tip = t;
		zafatena = z;
		cena = c;
	}
	void pechati();
};

void Soba::pechati()
{
	cout << tipNiza[tip] << " soba so reden broj " << broj;
```

```
	(zafatena == 1) ? cout << " e" : cout << " ne e";
	cout << " zafatena, a nejzinata cena iznesuva " << cena << " denari " << endl;
}

Soba defaultSobi[2] = { { 1, ednokrevetna, 0, 1200 }, { 2, dvokrevetna, 0, 2400 } };

struct Hotel
{
	string ime, lokacija;
	int brojSobi;
	Soba listaSobi[100];

	void init(string i = "Name", string l = "Address", int br = 2, Soba *s = defaultSobi)
	{
		ime = i;
		lokacija = l;
		brojSobi = br;
		for (int j = 0; j < brojSobi; j++)
			listaSobi[j] = s[j];
	}

	void pechati();
};

void Hotel::pechati()
{
	cout << "Hotelot " << ime << " raspolaga so vkupno " << brojSobi << " sobi " << endl;
	cout << "Vo prodolzhenie e listata na sobi " << endl;
	for (int j = 0; j < brojSobi; j++)
		listaSobi[j].pechati();
}

int main()
{
	Hotel hoteli[3];
	Soba sobi[100];
	int brojSobi;
	int tip;
	bool zafatena;
	int cena;
	string ime, adresa;

	//inicijalizacija na 3 hoteli
	hoteli[0].init(); //prviot go inicijalizirame so default-ni vrednosti

	for (int i = 1; i <= 2; i++)
	{
		cout << "Inicijalizirajte gi sobite za " << i << "-ot hotel" << endl << endl;
		cout << "Kolku sobi ima hotelot?" << endl;

		cin >> brojSobi;
		for (int j = 0; j < brojSobi; j++)
		{
			cout << "Vnesete tip:" << endl;
			cin >> tip;
			cout << "Dali e zafatena (1 za da, 0 za ne)" << endl;
			cin >> zafatena;
			cout << "Vnesete cena:" << endl;
			cin >> cena;
			sobi[j].init(j + 1, tipsoba(tip), zafatena, cena);
		}

		cout << endl << "Inicijalizirajte go " << i << "-ot hotel" << endl;
		cout << "Vnesete ime:" << endl;
		cin >> ime;
		cout << "Vnesete adresa:" << endl;
		cin >> adresa;
		hoteli[i].init(ime, adresa, brojSobi, sobi);
```

```
			cin.clear();
		}

		cout << "Pechatenje na informaciite za hotelite" << endl;
		for (int i = 0; i <= 2; i++)
		{
			hoteli[i].pechati();
		}

		return 0;
}
```