Универзитет „Св. Кирил и Методиј“ - Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Аудиториски вежби 2

Вовед во C++

Напреден развој на софтвер

Содржина

Вовед

- C++ вклучува многу карактеристики на C јазикот дополнети со дополнителни механизми за објектно-ориентирано програмирање (ООП)
- Јазикот C++ е развиен во лабораториите на Bell и во почетокот бил наречен „C со класи“
- Името C++ вклучува во себе операција за зголемување на јазикот C (++) и укажува на тоа дека C++ е верзија на C со проширени можности
- C++ компајлерот може да се користи и за преведување на C програми

Влез/Излез текови во C++ <iostream>

Во C++ за работа со влезно/излезните текови наместо функциите `printf` и `scanf` ги користиме наредбите `cout «` и `cin »`.

Пример

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

во C++ се запишува како:

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

При тоа препрорачливо е изолираните '$\backslash n$' да се заменат со '`endl`':

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

Влез/Излез текови во C++ <iostream>

За работа со влезно/излезните текови во C++ треба да се вклучи датотеката со заглавја <iostream>

Пример

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

int main() {
    cout << "Vnesi gi tvoite godini: ";
    int myAge;
    cin >> myAge;
    cout << "Vnesi gi godinite na prijatel: ";
    int friendsAge;
    cin >> friendsAge;
    if (myAge > friendsAge)
        cout << "Ti se postar.\n";
    else if (myAge < friendsAge)
        cout << "Prijatelot e postar.\n";
    else
        cout << "Ti i prijatelot imate godini.\n";
    return 0;
}
```

Декларација на променливи во C++

Променливите во C++ може да се декларираат во било кој дел од програмата сè 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`.

inline функции

При секој стандарден повик на функција се троши дополнително време во процесот на повикување на функцијата. Затоа во C++ при дефинирањето на мали и едноставни функции може да се употреби клучното зборче `inline` кое што наместо повик кон функцијата на местото каде што треба да се повика го вметнува самиот код на функцијата со што се избегнува дополнителното време при повикување на функцијата.

Пример функција за пресметување на волумен на коцка

```
#include <iostream>
using namespace std;
inline float cube(const float s) {
    return s * s * s;
}
int main() {
    cout << "Stranata na kockata: ";
    float strana;
    cin >> strana;
    cout << "Volumenot na kocka so strana " << strana << " e: " << cube(strana)
         << endl;
    return 0;
}
```

Дополнителни клучни зборови во C++

asm	explicit	operator
catch	friend	private
class	inline	protected
const_cast	mutable	public
delete	new	reinterpret_cast
dynamic_cast	namespace	static_cast
template	throw	using
this	try	virtual

typedef

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

Пример

```
#include <iostream>
using namespace std;
struct Person {
    char ime[80], adresa[90];
    double plata;
};
int main() {
    Person Vraboten[50]; // pole so elementi od tip struct Person
    Person Rakovoditel;
    Rakovoditel.ime = "Aleksandar";
    cout << "Imeto na rakovoditelot e" << Rakovoditel.ime << endl;
    return 0;
}
```

Содржина

Функции како членови на структура
1/3

Во C++ е дозволено дефинирање на функција како дел од некоја структура.

Пример

```
struct Person {
    char ime[80], adresa[80];
    // deklaracija na funkcijata za pecatenje na imeto i adresata
    void print(void);
};
```

Функции како членови на структура
2/3

1. Функцијата `print()` е само декларирана; кодот на функцијата се наоѓа на друго место.
2. Големината на структурата (sizeof) е определена **само** од големината на податочните елементи. Функциите кои се декларираирани во неа не влијаат на нејзината големина. Компајлерот го имплементира ова однесување со тоа што функцијата `print()` е позната само во контекст на структурата `Person`.
3. 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.

Функции како членови на структура
3/3

Користење на функција како член на структура

Пример

```
#include <iostream>
using namespace std;
struct Person {
    char ime[80], adresa[90];
    double plata;
    void printperson();
};
void Person::printperson() {
    cout << "Imeto na vraboteniot e:" << ime << "a, negovata adresa e:"
            << adresa << endl;
}
int main() {
    Person Rakovoditel;
    //...fali inicijalizacija
    Person.printperson();
    return (0);
}
```

Содржина

Референци

Дефиниција и начин на декларација

Референца е нов податочен тип во C++ кој е сличен на покажувач во C но многу по безбеден за употреба.

Начин на декларација

```
<Type> & <Name>
```

Пример

```
int A = 5;
int& rA = &A;
```

Референци

Разлики и сличности со покажувачи

Референците во C++ се разликуваат од покажувачите во следниве разлики:

1. Не може да се пристапи директно на референцата откако ќе се дефинира, секое пристапување до неа е всушност пристапување до објектот/променливата кон која референцира.
2. Отакако еднаш ќе се инцијализира не може да се промени да референцира кон друг објект/променлива.
3. Не може да бидат NULL (да не референцираат кон ништо).
4. Отакако ќе се декларираат мораат веднаш да се иницијализираат

Референци

Употреба

Една од најважните употреби на референците е при пренос на аргументи во функции.

Пример

```
int swap(int &a, int &b) {
    a += b;
    b = a - b;
    a -= b;
}
int main() {
    int x = 10;
    int y = 15;
    swap(x, y);
    // x = 15, y = 10
    return 0;
}
```

Референци

Пример аргументи референца кон структура

```
// deklaracija na struktura
struct Person {
    char ime[80], adresa[90];
    double plata;
};
Person Vraboten[50]; // poli elementi od tip struktura Person
// printperson ocekuva referenca do struktura
void printperson(Person const &p) {
    cout << "Imeto na vraboteniot e:" << p.ime << "a, negovata adresa e:"
            << p.adresa << endl;
}
// zemi gi podatocite za vraboteniot preku negoviot reden broj
Person const &ZemiVraboten(int index) {
... return (person[index]); // vrakja referenca
}
int main() {
    Person Rakovoditel;
    printperson(Rakovoditel); // nema referenca
    // promenlivata nema da se promeni vo funkcijata
    printperson(getperson(5)); // se prenesuva referenca
    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 ival = 4 ќе врати грешка

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

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

```
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.

Употреба на const со покажувачи

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".

```
#include <iostream>
using namespace std;
int main() {
    char const *buf = "hello";
    buf++; // prifateno od kompajlerot
    *buf = 'u'; // ne se prifaka od kompajlerot
    return (0);
}
```

Содржина

Простор на имиња - namespace
1/3

namespace се делови во кодот во кои може да се дефинираат и употребуваат (без конфликти) ентитети (променливи, функции, структури, и сл.) со исти имиња како постојните стандардно дефинирани или оние кои се јавуваат во друг простор на имиња.

Дефиниција на namespace

```
namespace ime_na_namespace {
    // (region za deklariranje entiteti)
    // deklarirani promenlivi, entiteti i funkcii
    // strukturi, klasi, vgnezdeni prostori na iminja
}
```

Простор на имиња - namespace
2/3

Дефиниција на простор не може да биде во блок (пр: функција), но може да се користат декларации на простор со исто име на повеќе места (multiple namespace). Тогаш се вели, просторот е отворен (open). На пример, ако имаме декларација на namespace Moj_prostor во prog1.cpp и prog2.cpp, тие ќе се обединат во еден простор кој ги содржи сите ентитети дефинирани во двата простора.

```
// vo prog1.cpp
namespace Moj_prostor {
double cos(double argInDegrees) {
...kod na funkcijata cos
}
}
// vo prog2.cpp
namespace Moj_prostor {
double sin(double argInDegrees) {
...kod na funkcijata sin
}
}
```

Простор на имиња - namespace
3/3

Просторот `Moj_prostor` ги содржи дефинициите и на двете функции, `sin()` и `cos()`. Наместо да се дефинираат, ентитетите може само да се декларираат како во следниот пример:

```
namespace Moj_prostor {
double cos(double degrees);
double sin(double degrees);
}
```

Затворени простори на имиња

Простор може да се дефинира без име; тогаш тој е анонимен (затворен) односно ентитетите на просторот може да се користат само во програмата во која е дефиниран просторот.
Пристап до ентитети од простор на имиња

```
// namespace Moj_prostor e definiran vo sledniot header file:
#include <Moj_prostor>
int main() {
    cout << "Cosinus od 60 stepeni e: " <<
    Moj_prostor::cos(60) << endl;
    return (0);
}
```

Затворени простори на имиња

Дефиниција на ентитет (пр. функција) со исто име како стандардна функција имплицира дека стандардната функција не може автоматски да се користи, туку треба да се користи scope операторот како во следниот пример:
Пристап до функции со исто име кои припаѓаат на стандардниот и новодефинираното простор на имиња `#include <iostream>`

```
#include <cmath>
namespace Moj_prostor {
double cos(double argInDegrees) {
...kod na funkcijata cos
}
using namespace std;
int main()
{
using Moj_prostor::cos;
...
cout << cos(60) // ja koristi funkcijata Moj_prostor::cos()
<< ::cos(1.5) // ja koristi standardnata cos() funkcija
<< endl;
return (0);
}
```

Употреба на using

Декларацијата using може да се користи во блок, меѓутоа треба да се внимава да нсе дефинираат променливи со исти имиња како имињата на ентитетите од просторот кој се декларира со using.

```
#include <iostream>
using namespace std;
namespace Moj_prostor {
int vrednost = 1;
}
int main() {
    using Moj_prostor::vrednost;
    ...
    cout << value << endl; // ja koristi Moj_prostor::vrednost
    int vrednost; // error: value already defined.
    return (0);
}
```

Употреба на using

За разлика од декларативно користење на using каде се наведува ентитетот или листата на ентитети кои се преземаат од просторот, може да се употреби наредбата using за импортирање на сите ентитети од соодветниот простор.

```
using namespace Moj_prostor;
```

Напомена:
Ако cos() е дефинирана во Moj_prostor, тогаш Moj_prostor::cos() ќе се повика ако употребиме cos(), односно стандардната cos() ке се повика ако не постои дефиниција на cos() во Moj_prostor.

Вгнездени простори на имиња (nested namespaces)

```
namespace Moj_prostor {
namespace Virtual {
void *pointer;
}
}
```

Пристап до вгнезден ентитет може да се напише на следните начини:

1. Со користење на scope операторот за целосна референца

```
int main() {
    Moj_prostor::Virtual::pointer = 0;
    return (0);
}
```

2. Користење на `using`-декларација за `Moj_prostor::Virtual`

```
. . .
using Moj_prostor::Virtual;
int main() {
    Virtual::pointer = 0;
    return (0);
}
```

Вгнездени простори на имиња (nested namespaces)

3. Користење на using-декларација за Moj_prostor::Virtual::pointer

```
...
using Moj_prostor::Virtual::pointer;
int main() {
    pointer = 0;
    return (0);
}
```

4. Користење на using – наредбата

```
using namespace Moj_prostor::Virtual;
int main() {
    pointer = 0;
    return (0);
}
```

Вгнездени простори на имиња (nested namespaces)

Две одвоени using-наредби не се дозволени

```
using namespace Moj_prostor;
using namespace Virtual;
int main() {
    pointer = 0;
    return (0);
}
```

5. Комбинација на using-наредба и using-деклрација

```
using namespace Moj_prostor;
using Virtual::pointer;
int main() {
    pointer = 0;
    return (0);
}
```

Нова синтакса за кастирање

Во C се употребуваше следната синтакса за кастирање

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

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

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

Во C++ се воведуваат 4 нови видови кастирање:

1. `static_cast<type>(expression)` - стандардно кастирање за конверзија на еден тип во друг
2. `const_cast<type>(expression)` - се користи за модифицирање на типот на константи
3. `reintrepret_cast<type>(expression)` - се користи за промена на интерпретација на информацијата
4. `dynamic_cast<type>(expression)` - поврзан со polymorphism

Материјали и прашања

Предавања, аудиториски вежби, соопштенија
courses.finki.ukim.mk

Изворен код на сите примери и задачи
bitbucket.org/tdelev/finki-nrs

Прашања и одговори
qa.finki.ukim.mk