Увод в програмирането

Лекция 2:

Основни елементи на една програма на C++

(продължение)

Преговор (1)

- Идентификатори – x, main, SIZE
- Литерали (явни константи) – 5, -3.14, "Hi"
- Запазени думи – int, return, …
- Аритметични оператори - +, -, *, /, %
- Коментари - /*текст*/, // текст
- Променливи – int a = 5, b; b = 10;
- Типове – int, double, ...
- Изрази – 5, a, 5+a, (5+a)*4385/(4832-a)

Преговор (2)

- Вход и изход: cin >> a; cout << 2 * a;
- Константи – const double PI = 3.1415; ~~PI = 5;~~
- Приоритет на операциите – a+b*c и (a+b)*c
- Готови математически функции в cmath – sqrt, fabs, sin, ...

Израз – формална дефиниция

- До момента имаме следната представа:

  <израз> ::= <константа> |

  <променлива> |

  <унарна_опрация> <израз> |

  <израз> <бинарна_операция> <израз> |

  (<израз>)

- Операция (operator)
- Оператор (statement)
  - Завършва с ”;”
  - Нашите програми са последователности от оператори; изпълняват се в същия ред

Оператор за присвояване

- След като една променлива е декларирана, на нея многократно може да ѝ се задава нова стойност

  <променлива> = <израз>;

- Напр.: **`int a, b; a = 10; b = 15 * a;`**
- Възможно е десният израз да се присвои на няколко променливи наведнъж:
  - **`a = b = 5;`** е еквивалентно на **`b = 5; a = 5;`**
  - Операторът за присвояване ”връща” стойността, която е зададена като десен аргумент, затова цялата конструкция може да участва като израз

Функция main

- Входна точка на програмата
- В една програма трябва да има точно една main функция
  - Ако създадем нов .cpp файл в същия проект с нова main функция, ще получим грешка при компилиране
- Може да се дефинира по различни начини, напр.:
  - `int main() { … }`
  - `int main(int argc, char *argv[]) { … return 0; }`
  - `void main() { … }`
  - Ще бъдат разгледани в лекцията за функции

Специални символи в низовете

- Нека искаме да отпечатаме следното съобщение:

  `Cannot find file "C:\UP-ocenki.xls"`

- Ако напишем

  `cout << "Cannot find file "C:\UP-ocenki.xls"";`

  ще получим съобщение за грешка – според синтаксиса на C++ низът свършва при първото срещане на кавичка и следващите символи вече нарушават синтаксиса

Специални символи (2)

- Символът ”\” се използва, за да се задават комбинации от символи, които се интерпретират като един символ
  - \n → нов ред
  - \” → ”
  - \\ → \
  - и други
- Така получаваме:

```
cout <<
"Cannot find file \"C:\\UP-ocenki.xls\"";
```

Отпечатване на нов ред

- ```
  cout << "First line\nSecond line" <<
       "\n";
  ```
- ```
  cout << "First line" << endl <<
       "Second line" << endl;
  ```

#include

- Това не е hashtag :)
- Когато използваме наготово вече реализирана функционалност, трябва да посочим съответните header файлове
  - iostream – за вход и изход със cin и cout
  - cmath – за математически функции

#include (2)

- Допълнителен материал:
  - Можем да дефинираме и свои header файлове
    - Вместо #include <file.h> пишем #include "file.h"
  - Редовете, започващи с ”#”, са директиви на препроцесора
    - Обработват се преди самото компилиране
    - Редът с #include се заменя със съдържанието на указания файл
  - Namespaces – за да няма проблеми с функции/променливи с еднакви имена, дефинирани в различни header файлове
    - using namespace std; - за да не се налага да пишем std:: пред cout и т.н.

Преобразуване на типовете (1)

- C++ е строго типизиран език
- Всеки израз има тип (int, double и т.н.)
- Примери:

```
int a = 5;
int b = a;
double d = 3.14159 * 2.0;
cout << (a + b) * 34245543 – a;
```

- От какви типове са горните изрази?

Преобразуване на типовете (2)

- При задаване на стойност на променлива (или константа) можем да използваме израз и от друг тип

```
double x = 5; // int → double

int i = 4; short s = i; // int → short

i = 3.14; // double → int, i става 3

x = "Nice"; // не за всяка комбинация от // типове е дефинирано такова присвояване
```

Неявно преобразуване (1)

- В предишните примери се извършва неявно (имплицитно) преобразуване на типовете
- Когато се извършва преобразуване към ”по-голям” тип, няма загуба на данни (стойността на израза остава същата)
  - Напр. short → int, int → double

Неявно преобразуване (2)

- Ако преобразуването не е към ”по-голям” тип, може да има загуба
  - Напр. **`int i = 3.14; unsigned int u = -1;`**
- Компилаторът може да направи предупреждение (Warning)
  - Warning означава, че синтаксисът на езика е спазен, но е много вероятно да е допусната грешка
  - Друг пример: декларирали сме променлива, която не се използва – вероятно сме използвали друга вместо нея или сме забравили да допишем нещо

Явно преобразуване

- Явно преобразуване – пред даден израз се указва нов тип (ограден със скоби):
  - `double x = 3.14; int i = (int)x * 2;`
  - Няма warning – изрично указваме, че искаме това
  - Получаваме нов израз, който има нов тип, а самата променлива остава от стария тип

Преобразуване на типовете (3)

- Изрази от еднакъв тип могат да се комбинират с операциите, дефинирани за този тип
- Новият израз е от същия тип
- Интересен пример е прилагането на операцията деление върху целочислени изрази:

```
int i = 7; double x = i / 2;
cout << x;
```

- Какво ще отпечата горният код?

Преобразуване на типовете (4)

- Горният пример ще отпечата 3, понеже:
  - Първо се изчислява дясната част – компилаторът не се интересува, че частното ще се запише в double променлива
  - Операторът ”/”, приложен върху целочислени операнди, извършва целочислено деление
  - Т.е. намира цялата част на частното, без остатъка

Преобразуване на типовете (5)

- При смесване на изрази от различен тип (съвместими), новият израз е от ”по-големия” тип
  - Пример: 2 * 3.14 е от тип double
- Ако в примера с целочислено деление искаме да получим 3.5:
  - `double x = i / 2.0;`
  - `double x = (double)i / 2;`

Красиво оформление

- Ако синтаксисът на езика е спазен, компилаторът не се интересува дали кодът е написан ”красиво”
  - Напр. интервали, нови редове, табулации (всичко това е white space) се игнорират, когато не са задължителни
- Кодът трябва да е четлив и еднакво оформен
  - След нас ще работят по него и други
  - В реален проект кодът ни няма да бъде приет, ако не отговаря на общоприети правила

Някои *конвенции* за писане на код

- Имената на променливите трябва да указват предназначението им, напр. вместо a, b, c – price, count, lastDigit, numberOfStudentsInFirstGrade
  - За дългите имена има auto complete
  - Този стил се нарича camelCase; има и други
  - Константи обикновено с главни букви: PI, MAX_SIZE
  - Ще се има предвид при оценяването в курса по УП
- Около операторите като +, -, << и т.н. се поставят интервали (вместо a+b → a + b)
- След ”{” редовете трябва да са отместени малко навътре (колко навътре – има различни мнения)
- Редовете не трябва да бъдат много дълги
- Коментари, които подпомагат разбирането на кода

Напомняне за ИС

- Бакалаври: записахте ли практикума по УП като избираем курс в СУСИ? Има краен срок
- Магистри: записахте ли курса по УП като избираем курс в СУСИ? Има краен срок

Въпроси