Въведение в езика C++

Любомир Чорбаджиев
Технологическо училище "Електронни системи"
Технически университет, София
lchorbadjiev@elsys-bg.org
Revision : 1.8

27 септември 2004 г.

Учебници

- Bjarne Stroustrup, The C++ Programming Language, 3rd Edition, Addison Wesley Longman, Inc., Reading, MA (1997)
- Stanley B. Lippman, Josee Lajoie, C++ Primer, Third Edition, Addison Wesley (1997)
- Bruce Eckel, Thinking In C++, Second Edition, Prentice Hall Inc., 2000, http://www.MindView.net
- Scott Meyers, Effective C++, 2nd Edition, Addison Wesley Longman, Inc., Reading, MA (1998)
- Herb Sutter, Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions, Addison-Wesley, Reading, MA, 2000

История

- Езикът C++ е разработен от Bjarne Stroustrup. Работата по езика започва през 1979. Първият вариант на езика се появява през 1980 – "C with Classes".
- Името C++ се използва за първи път през 1983, а през 1984 се появява следваща версия на езика. В следващите години езикът продължава да се развива и разпространява.
- През 1998 г. е одобрен стандартът за езика C++ – ISO/IEC 14882 "Standard for the C++ Programming Language".

Връзка между C и C++

- Като базов език за C++ е избран езикът C.
- Основна цел при разработването на C++ е той да бъде съвместим със C. Всяка конструкция, която е допустима в C и C++, има еднакъв смисъл в двата езика.
- Пълна съвместимост между C и C++ няма. Налагането на пълна съвместимост между двата езика би довело до жертването на различни предимства, които C++ има.

Обзор на езика C++

- C++ е език за програмиране с общо предназначение.
- C++ предоставя механизми за поддръжка на обектно-ориентиран стил на програмиране.
- C++ е създаден с цел да се добави поддръжка на обектно-ориентираното и обобщено програмиране към традиционния C.

Пример: HelloWorld.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5   cout<<"Hello"<<' '<<"World!"<<endl;
6   return 0;
7 }
```

- `iostream` е стандартен заглавен файл, част от стандартната библиотека на C++. В него са дефинирани основните операции за вход и изход. Там се съдържа информацията за `cout`, която е необходима за нашата програма.

Пример: HelloWorld.cpp

- Имената, които са дефинирани в стандартната библиотека на C++, не могат да бъдат използвани в нашата програма, освен ако не се използва израза: **using namespace** `std;` (вж. ред 2).
- За входно/изходни операции в C++ се използват **потоци**. `cout` е стандартният поток за изход. Потокът за изход може да обработва последователност от различни по тип аргументи чрез оператора << (вж. ред 5).
- В `iostream` е дефинирана специална функция `endl`, която обозначава край на реда — извежда информацията на изходното устройство, като предава и символ за край на реда.

Потоци за вход/изход

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    int number;
6    cout<< "Enter␣an␣integer:␣";
7    cin >> number;
8    cout<< "hex␣(0x" << hex << number << ")" <<endl;
9    return 0;
10 }
```

- Потокът `cin` е стандартният поток за вход. Потокът за вход може да обработва последователност от различни по тип аргументи чрез оператора >>.
- В стандартната входно/изходна библиотека на C++ са дефинирани набор от специални "функции", които се наричат **манипулатори**. Пример за манипулатор е `hex` – използван на ред 8. При взаимодействие с потока манипулаторите не отпечатват нищо, а само променят състоянието на потока.

Стрингове

- В `C`, когато се говори за стрингове, се има предвид масив от **char**. Масивите от символи са трудни за употреба. Такива стандартни операции, като копиране и конкатенация, изискват от потребителя да се грижи сам за всички детайли по тях.
- В `C++` когато се говори за стрингове, обикновено се има предвид типа `string`, който е дефиниран в стандартната библиотека. Този тип е разработен така, че да облекчи обичайните задачи, които се изпълняват със стрингове.

Стрингове

```
1  #include <string>
2  #include <iostream>
3  using namespace std;
4
5  int main() {
6    string str1;
7    string str2("Hello␣World!");
8    string str3="Hello␣World␣Again!";
9
10   str1="Hi!";
11   string r1=str1+"␣"+str2;
12   r1+="␣";
13   r1+=str3;
14
15   cout << r1 << endl;
16   return 0;
17 }
```

Работа с файлове

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5
6  int main() {
7    string line;
8    ifstream in("temp.txt");
9    ofstream out("copy-of-temp.txt");
10
11   while(getline(in,line)) {
12     out << line << endl;
13   }
14   return 0;
15 }
```

Типове

- Всеки идентификатор в една C++ програма, трябва да има асоцииран с него **тип**.
- **Типът** определя какви операции са приложими към дадения идентификатор и как трябва да се интерпретират тези операции.
- В езика C++ са дефинирани набор от базови (фундаментални) типове и са предоставени средства за дефиниране на нови типове от потребителя.

Фундаментални типове

- Логически (булев) тип – **bool**
- Символни типове – **char**, **wchar_t**
- Целочислени типове – **int**
- Типове с плаваща запетая – **float**, **double**
- Изброими типове – дефинирани от потребителя с използването на **enum**
- Типът **void**

Освен тези типове, могат да се конструират и други:

- Указатели – например **int***
- Масиви – например **char**[]
- Псевдоними – например **double&**
- Структури от данни и класове

Фундаментални типове: примери

```
1  bool a=false;
2  bool b=true;
3  bool bb=a||b;
4
5  char ch='a';
6
7  int count=1;
8  unsigned int i=0;
9  long int li;
10
11 double x=0.0;
12 double y, z;
13 const double pi=3.14159265358;
```

Типът void

- По принцип типът **void** е фундаментален (базов) тип, но неговото използване е ограничено.
- Типът **void** може да се използва само като част от по-сложен тип. Обекти от тип **void** не съществуват.
- Допустимото използване на този тип е:
  - ◇ Типът **void** може да се използва за да укаже, че дадена функция не връща резултат – **void** `fun();`.
  - ◇ Типът **void** може да се използва като базов за указател към обект от неизвестен тип – **void*** `pv;`.

Изброими типове

- Изброимите типове, дефинирани с **enum**, задават набор от значения, определен от потребителя. Като елементи на изброимия тип могат да се дефинират именувани цели константи. Например:

```
enum {FIRST, SECOND, THIRD};
```

- На изброимия тип може да му се присвои име:

```
enum ENUMERATION_TYPE {FIRST, SECOND, THIRD};
```

  След това името да се използва за дефиниране на променливи от този тип:

```
ENUMERATION_TYPE enumerator;
```

Декларации

- Преди даден идентификатор да може да се използва в една програма на C++, той трябва да бъде деклариран.
- **Декларация** е термин, който се използва за всичко, което казва на компилатора какъв е смисълът на даден идентификатор.
- За да се използва даден идентификатор, компилаторът трябва да знае какво представлява този идентификатор — дали е име на променлива, на функция, на тип или на нещо друго. С други думи трябва да бъде указан **типът** на идентификатора.
- Поради това във всеки файл с код трябва да се съдържа декларация на всички имена, които се използват.

Декларации: примери

```
1  char ch;
2  int count=1;
3  const double pi=3.14159265358979;
4  extern int error_number;
5  char* name="Bjarne␣Stroustrup";
6  char* season[]={"spring","summer","autumn","winter"};
7
8  struct Date {int d,m,y;};
9  int day(Date* p) {return p->d;}
10 double sqrt(double);
11
12 struct User;
13 enum Beer {Carlsberg, Tuborg, Beiks, Amstel};
```

Декларации и дефиниции

- Повечето от представените **декларации** са всъщност **дефиниции** – те определят някаква същност, която съответства на даденият идентификатор:
  - ◇ за променливата `ch` дефинираната същност представлява подходящото количество памет, необходима на тази променлива;
  - ◇ за структурата `Date`, дефинираната същност представлява нов тип;
  - ◇ за функцията **int** `day(Date* p);` дефинираната същност е алгоритъмът по който се изпълнява функцията;

Декларации и дефиниции

- От представените примерни **декларации**, само следните **не са дефиниции**:

```
extern int error_number;
double sqrt(double);
struct User;
```

- Променливата `error_number`, функцията `sqrt` и структурата `User` трябва да бъдат **дефинирани** някъде другаде в програмата.

Декларации и дефиниции

```
1 int count;
2 int count;// error
3
4 extern int error_number;
5 extern int error_number; // OK!
6
7 extern short error_number; // error!
```

- В дадена програма на C++ за всеки идентификатор може да има **само една дефиниция**.
- В дадена програма на C++ може да **има няколко декларации** за даден идентификатор.