Увод в програмирането

4: Основни елементи на езика (част 2)

доц. Атанас Семерджиев

Съдържание

- Типове дефинирани от потребителя
  - Изброим тип (enumeration)
  - typedef
- Оператори
- Изрази
- Преобразуване на типовете

2

Типове дефинирани от потребителя

Изброим тип (enum) и typedef

3

Изброим тип (enum)

- Това е един от т.нар. „типове дефинирани от потребителя“.
- Състои се от една или повече дефиниции на константи.

```
enum DayOfWeek
{
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
};
```

Точка и запетая!!!

4

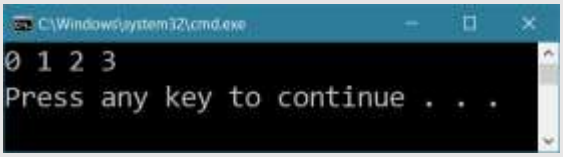
```
#include <iostream>

using namespace std;

enum SeasonType
{
   SPRING, SUMMER, AUTUMN, WINTER
};

int main()
{
  cout << SPRING << " " << SUMMER << " "
       << AUTUMN << " " << WINTER << " "
       << endl;

  return 0;
}
```



5

```
enum SeasonType
{
   SPRING, SUMMER, AUTUMN, WINTER
};

int main()
{
   enum SeasonType season;   // Валидно в C/C++
   SeasonType anotherSeason; // Валидно в C++
   season = SPRING;
   season = SeasonType::SPRING;
   anotherSeason = season;
   season = 1; // Грешка!
   return 0;
}
```

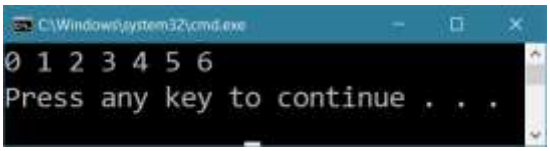
6

Стойности

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

enum WeekdayType
{
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
};
```

```
int main()
{
    cout << MONDAY    << " "
         << TUESDAY   << " "
         << WEDNESDAY << " "
         << THURSDAY  << " "
         << FRIDAY    << " "
         << SATURDAY  << " "
         << SUNDAY    << "\n";
}
```



7

Стойности дефинирани от потребителя

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

enum WeekdayType {
  MONDAY = 4,
  TUESDAY = 1000,
  WEDNESDAY = 900,
  THURSDAY = 45,
  FRIDAY = 11,
  SATURDAY = 123,
  SUNDAY = 5
};
```

```
int main()
{
  cout << MONDAY    << "\n"
       << TUESDAY   << "\n"
       << WEDNESDAY << "\n"
       << THURSDAY  << "\n"
       << FRIDAY    << "\n"
       << SATURDAY  << "\n"
       << SUNDAY    << "\n";
}
```

Изход

```
4
1000
900
45
11
123
5
```

8

Стойности дефинирани от потребителя

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

enum WeekdayType {
  MONDAY,
  TUESDAY,
  WEDNESDAY = 1000,
  THURSDAY,
  FRIDAY,
  SATURDAY = -20000,
  SUNDAY
};
```

```
int main()
{
  cout << MONDAY    << "\n"
       << TUESDAY   << "\n"
       << WEDNESDAY << "\n"
       << THURSDAY  << "\n"
       << FRIDAY    << "\n"
       << SATURDAY  << "\n"
       << SUNDAY    << "\n";
}
```

Изход

```
0
1
1000
1001
1002
-20000
-19999
```

9

Повтарящи се стойности

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

enum WeekdayType {
  MONDAY = 1000,
  TUESDAY,
  WEDNESDAY,
  THURSDAY = 1000,
  FRIDAY,
  SATURDAY,
  SUNDAY
};
```

```
int main()
{
  cout << MONDAY    << "\n"
       << TUESDAY   << "\n"
       << WEDNESDAY << "\n"
       << THURSDAY  << "\n"
       << FRIDAY    << "\n"
       << SATURDAY  << "\n"
       << SUNDAY    << "\n";
}
```

Изход

```
1000
1001
1002
1000
1001
1002
1003
```

10

Повтарящи се стойности

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

enum WeekdayType {
  MONDAY = 1000,
  TUESDAY,
  WEDNESDAY,
  THURSDAY = 1000,
  FRIDAY,
  SATURDAY,
  SUNDAY
};
```

```
int main()
{
   WeekdayType day1 = TUESDAY;
   WeekdayType day2 = FRIDAY;

   if(day1 == day2)
     cout << "Equal!\n";
}
```

Изход

```
Equal
```

11

Добра практика: Добавяне на „празна“ стойност

```
enum SeasonType {
   UNDEFINED, SPRING, SUMMER, AUTUMN, WINTER
};

int main()
{
   SeasonType season = UNDEFINED;
   // ... (някакъв код) ...
   if(season == UNDEFINED)
      cout << "Season is undefined!\n";
}
```

12

typedef

typedef може да се използва за създаване на алтернативно име (alias) на някой вече съществуващ тип.

```
typedef unsigned char BYTE;

int main()
{
   BYTE x = 3;
   return 0;
}
```

13

Оператори

14

Оператори: Приоритет

Операторите имат приоритет (precedence), който определя реда на тяхното прилагане.

```
bool IsItTrue = 20 > 10 - 1;

if(IsItTrue)
    cout << "True";
else
    cout << "False";
```

15

Оператори: Асоциативност

Различаваме ляво и дясно асоциативни оператори. Например:

```
// Ляво асоциативен, т.е.
// Division = (8 / 4) / 2
// Така имаме Divison == 1
double Division = 8 / 4 / 2;
```

16

Оператор	Име	Асоциативност
++ --	Postfix Increment Postfix Decrement	Left to Right Left to Right
++ --	Prefix Increment Prefix Decrement	Right to Left Right to Left
* /	Multiplication Division	Left to Right Left to Right
+ - %	Addition Subtraction Modulus	Left to Right Left to Right Left to Right
< >	Less Than Greater Than	Left to Right Left to Right
&&	Logical And	Left to Right
||	Logical Or	Left to Right
?:	Conditional	Right to Left
=	Assignment	Right to Left
,	Comma	Left to Right

17

Префиксни, инфиксни и постфиксни оператори

- Префиксни:
  - ++index
  - -value
- Инфиксни:
  - x + y
  - Age > 10
- Постфиксни:
  - index++;
  - CalculateArea(500)

18

Обекти, rvalue, lvalue

Обект: Регион в паметта, който можем да манипулираме.

lvalue: израз реферираш обект.

rvalue: израз, който не е lvalue.

```
int a; // Дефинира обект от тип int

a = a + 50; // a реферира обект (lvalue)
            // a + 50 е израз (rvalue)
```

19

Обекти, rvalue, lvalue

Внимание!!!

Не е задължително lvalue да бъде променлива.

Не е задължително да можем да присвоим стойност на lvalue.

```
const int MAX_AGE = 200; // a е lvalue

MAX_AGE = 50; // Грешка!!!
```

20

Оператор за присвояване (=)

21

Присвояване (=)

- Лявата страна задължително е lvalue.
- Дясната страна е rvalue.

```
int a, b, c;
a = 58; // Константа
b = a; // Променлива
c = (a + b) / 2; // Израз
```

22

Присвояване (=)

Присвояването е дясно асоциативно:

```
int a, b;

// Първо се изпълнява b = 10
// и едва после – a = b.
a = b = 10;

// Еквивалентен запис:
b = 10;
a = b;
```

23

Присвояване (=)

Даденият по-долу код е допустим, но не е препоръчително да пишете по този начин:

```
int a, b;

b = 500;

a = 123 + (b = 10);
```

24

Compound Assignment

```
int a = 5;

// Добавя 5 към a
a = a + 5; // a: 10

// Този израз е еквивалентен на по-горния
a += 5; // a: 15
```

25

Compound Assignment

```
int a = 5;

a += 5;   // a: 10

a -= 9;   // a: 1

a *= 100; // a: 100

a /= 10;  // a: 10
```

26

Аритметични оператори

27

Аритметични оператори

```
int a = 1 + 2;
int b = 2 - 1;
int c = 2 * 2;

int d = 234 / 10  // Целочислено деление
                  // Тук d == 23

int e = 234 % 10; // Остатък при целочислено деление
                  // В случая e == 4
```

28

Типът има значение!

```
int A = 234 / 10;      // Целочислено деление
                       // Тук A == 23

float B = 234 / 10;    // Отново имаме
                       // целочислено деление!!!
                       // B == 23

float C = 234.0 / 10; // OK, тук вече
                       // C == 23.4
```

29

Инкрементиране / декрементиране

```
int a = 5;

a++; // a: 6
++a; // a: 7

a--; // a: 6
--a; // a: 5
```

30

Инкрементиране / декрементиране

```
int a = 5;

cout << a++; // извежда 5
cout << ++a; // извежда 7

cout << a--; // извежда 7
cout << --a; // извежда 5
```

31

Инкрементиране / декрементиране

- **Важно!!!**
  - И двата оператора се прилагат върху lvalue
  - И двата израза за rvalue изрази.
- По тази причина, даденият по-долу код е невалиден!

```
int a = 5;
a++++; // Грешка!!!
```

32

Еквивалентност и релации

33

```
int a = 10;

a > 10;  // false

a < 10;  // false

a <= 10; // true

a >= 10; // true

a == 10; // true

a != 10; // false
```

34

Yoda Conditionals

Внимание!!!

Разпространена грешка е да се объркат операторите = и ==, например:

```
int count = 6;

if(count = 5)
{
  // Some stuff...
}
```

```
if(5 = count)
```

```
if (5 == count)
```

Това би предизвикало грешка при компилация и ще открием проблема навреме.

35

Логически оператори

36

Логически оператори: Конюнкция

- **Асоциативност:** лява
- Истина само ако и двете страни са истина.

A && B	A(true)	A(false)
B(true)	True	False
B(false)	False	False

```
bool A = true;
bool B = true;
bool C = false;

A && B; // true;
A && C; // false;
C && C; // false;
```

37

Логически оператори: Дизюнкция

- **Асоциативност:** лява
- Истина ако поне една от двете страни е истина.

A || B	A(true)	A(false)
B(true)	True	True
B(false)	True	False

```
bool A = true;
bool B = true;
bool C = false;

A || B; // true;
A || C; // true;
C || C; // false;
```

38

Логически оператори

- **Важно!!!**
  - Операндите на && и || се оценяват отляво-надясно.
  - Операндите се оценяват само докато се получи стойността на израза!
  - Това е от съществено значение, когато някой операнд има страничен ефект.

39

```
int a;
cout << "Enter an integer: ";
cin >> a;

if(a >= 10 && (a+=10) <= 20)
{
    cout << "True: a is " << a << endl;
}
else
{
    cout << "False: a is " << a << endl;
}
```

40

Логически оператори: Отрицание

- Отрицанието приложено върху истина дава лъжа.
- Приложено върху лъжа, то дава истина

```
bool a = !(1 < 2); // false

bool b = !(2 == 5); // true
```

41

Условен оператор (?:)

42

Условен оператор (?:)

- Общ вид:
  - <условие> ? <израз 1> : <израз 2>
- Ако <условие> е истина (има стойност true), тогава се оценява <израз 1> и тази стойност се използва за стойност на целия израз.
- В противен случай се използва <израз2>

```
int a = (1 < 2) ? 100 : 200;
```

43

Условен оператор (?:)

```
int a = 20;

// Какво ли прави това?
a != 0 ? 100 / a : 0;

// Еквивалентен, но по-ясен запис.
(a != 0) ? (100 / a) : 0;
```

44

Условен оператор (?:)

```
int a = -20;

// Внимание!!!
int d = 5 + a > 0 ? a : -a; // d: 20

int e = 5 + (a > 0 ? a : -a); // e: 25
```

45

Оператор	Име	Асоциативност
++ --	Postfix Increment Postfix Decrement	Left to Right Left to Right
++ --	Prefix Increment Prefix Decrement	Right to Left Right to Left
* /	Multiplication Division	Left to Right Left to Right
+ - %	Addition Subtraction Modulus	Left to Right Left to Right Left to Right
< >	Less Than Greater Than	Left to Right Left to Right
&&	Logical And	Left to Right
||	Logical Or	Left to Right
?:	Conditional	Right to Left
=	Assignment	Right to Left
,	Comma	Left to Right

46

Оператор запетая (,)

47

Оператор запетая (,)

- Ляво асоциативен
- Изразите се оценяват гарантирано един след друг отляво-надясно.
- Стойността на оператора се определя от най-десния израз.

```
int a = 0;

1+2, a, a+10;  // Стойност на израза: 10

a++, a++, a++; // Стойност на израза: 2
               // Стойност на a: 3
```

48

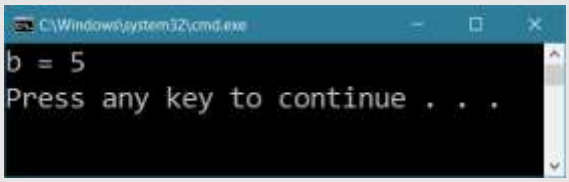
```
#include <iostream>
using namespace std;

int main()
{
    int a = 100;
    int b;

    b = 5, ++a, a;

    cout << "b = " << b << endl;

    return 0;
}
```



49

Оператор	Име	Асоциативност
++ --	Postfix Increment Postfix Decrement	Left to Right Left to Right
++ --	Prefix Increment Prefix Decrement	Right to Left Right to Left
* /	Multiplication Division	Left to Right Left to Right
+ - %	Addition Subtraction Modulus	Left to Right Left to Right Left to Right
< >	Less Than Greater Than	Left to Right Left to Right
&&	Logical And	Left to Right
||	Logical Or	Left to Right
?:	Conditional	Right to Left
=	Assignment	Right to Left
,	Comma	Left to Right

50

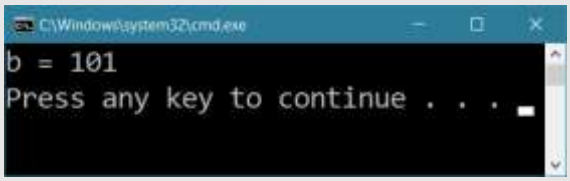
```
#include <iostream>
using namespace std;

int main()
{
   int a = 100;
   int b;

   b = (5, ++a, a);

   cout << "b = " << b << endl;

   return 0;
}
```



51

Оператор sizeof()

Може да се използва за намиране на размера в байтове на даден тип или променлива от даден тип.

```
int a;

int b = sizeof(double);

int c = sizeof(a)
```

52

Преобразуване на типове (type casting)

53

Преобразуване на типове

- Често, когато в даден израз участват операнди от различни типове, те трябва да се преобразуват до някакъв общ тип.
- **Неявно (имплицитно) преобразуване**
  - Извършва се автоматично от компилатора и не изисква оператор.
- **Явно (експлицитно) преобразуване**
  - Ръчно, с помощта на операторите за преобразуване.

54

Преобразуване на типове

- Можем да преобразуваме всеки от разгледаните досега основни типове до останалите:
  - bool
  - char (signed или unsigned)
  - short (signed или unsigned)
  - int (signed или unsigned)
  - long (signed или unsigned)
  - double
  - float
- ВНИМАНИЕ!!!
  - При някои от тези преобразувания можем да загубим информация!

55

Булев тип

- Когато конвертираме число към булев тип:
  - `0` → false
  - `<цяло число различно от 0>` → true
- От булев тип към число:
  - false → `0`
  - true → `1`

56

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

int main()
{
   int StoredItemsCount = 5;

   if(StoredItemsCount)
   {
      cout << "Not empty!";
   }

   return 0;
}
```

57

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

int main()
{
   int Input = 0; // Поредното въведено число
   int EnteredFives = 0; // Брой въведени петици

   while(Input >= 0)
   {
      cout << "Enter a non-negative number or -1 to end:";
      cin >> Input;
      EnteredFives += (Input == 5);
   }

   cout << "you entered " << EnteredFives << " fives\n";
}
```

58

Преобразувания между числа

- По принцип преобразуване от „по-малък“ към „по-голям“ тип не предизвиква проблеми:

```
short x = 10;
int y = x;
double z = y;
```

59

Преобразувания между числа

- Обратните преобразувания могат да доведат до загуба на информация и компилаторът генерира или предупреждение или грешка:

```
long x = 0x7FFFFFFF;
unsigned long y = 0xFFFFFFFF;

short y = x; // y: 0xFFFF
long b = y;  // b: 0xFFFFFFFF
```

60

Неявни преобразувания

- При аритметични изрази (+, -, * или /) „по-простият“ тип се конвертира до по-сложния:

```
int main()
{
   double A = 123.45;

   int B = 12345;

   double result = A + B * 10;
}
```

61

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

int main()
{
   int a = 5;

   double b = a / 7;

   cout << b << endl; // Извежда 0

   return 0;
}
```

62

Явни преобразувания

За целта се използва операторът за преобразуване на тип (type casting operator):

(<тип>)<израз>

```
bool BooleanValue = true;

int IntValue = (int) BooleanValue;

double DoubleValue = (double) (IntValue * 5);
```

63

```
double PI = 3.14;

int Value = 3.14; // Предупреждение

Value = PI; // Предупреждение

Value = PI * 10 * 10; // Предупреждение

Value = (int) PI * 10 * 10; // Value=300

Value = (int)(PI * 10 * 10); // Value=314
```

64

```
int a = 5;

double b = a / 7;   // b: 0

double c = a / 7.0; // c: 0.714285...

double d = a / (double)7; // d: 0.71428...

double e = a / double(7); // e: 0.71428...

double f = (double)a / 7; // f: 0.71428...
```

65

Преобразувания от и до enum

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

enum SeasonType {
   UNDEFINED, // UNDEFINED == 0
   SPRING,    // SPRING == 1
   SUMMER,    // SUMMER == 2
   AUTUMN,    // AUTUMN == 3
   WINTER     // WINTER == 4
};
```

66

```
int Value = SPRING; // OK

SeasonType Season = SPRING; // OK

Season = 0; // Грешка!

Season = Value; // Грешка!

Season = (SeasonType) 0; // OK

// По-долният ред се компилира, но след него
// стойността съхранена в Season е невалидна!
Season = (SeasonType) 1000;
```

67

```
// Стойност от enum тип може да участва в аритметичен
// израз, но има важна особеност:
// По-долу foo конвертира до int, а не обратното!
SeasonType Season = SPRING;
int Value = Season + 1;

// Върху стойността не може да се прилагат compound
// assignment операторите или ++ и --
Season++;  // Грешка!
Season+=1; // Грешка!
```

68

```
// Като следствие: стойностите в един изброим тип не
// могат да се итерират чрез инкрементиране.
// Даденият по-долу код не е валиден:
for(SeasonType i = UNDEFINED; i < WINTER; i++)
    cout << i << "\n" ;

// Възможно е такъв ефект да се постигне чрез
// преобразуване, но това е лоша практика, тъй като
// лесно можем да генерираме невалидна стойност.
SeasonType Season = AUTUMN;
SeasonType NextSeason = (SeasonType)(Season + 1);
```

69