5

Скаларни типове символен и изброен

5.1 Тип символен

Досега използвахме главно числови данни. В редица приложения се налага да се описва и обработва и символна информация. Да разгледаме следната задача.

Задача 44. Да се напише програма, която намира стойността на числов израз без скоби и без приоритет на операторите (+, -, * и /). Например, стойността на израза 12-3*2/4 е 4.5, а не 10.5. Пресмятането завършва след въвеждане на знака =.

Тази задача изисква програмата да получава за вход редица от числа, знаците за аритметични операции +, -, * и /, да завършва със знака = и да намира стойността на въведения аритметичен израз.

За реализирането й ще използваме данни от тип символен. Ще се върнем към задачата след разглеждането на този тип.

Типът се нарича още **знаков** и се причислява към интегралните типове на езика. За означаването му се използва запазената дума char.

Множество от стойности

Състои се от крайно и наредено множество от символи, обикновено вариант на ISO-646, например ASCII, и обхваща символите от клавиатурата. Стандартизирана е само част от символното множество – 128 символа (главните и малки буква на латинската азбука, цифрите, символите за пунктуация и някои управляващи символи). Всяка реализация включва стандартизираното подмножество и има свободата на избор за останалите символи, което създава редица проблеми.

Реализацията Visual C++ 6.0 включва в множеството от стойностите си два вида символи – **графични** и **управляващи**.

Графичните символи имат видимо представяне. Означават се чрез ограждане на символа в апострофи.

Пример: ‘a’ ‘d’ ‘K’ ‘!’ ‘1’ ‘ ‘ ‘+’
са означения на символите a, d, K, !, 1, интервал и +.
Графичните символи \ и ‘ се представят чрез ‘\\’ и ‘\’’ съответно. Символът ? се представя и като ‘?’ ,и като ‘\?’.

Управляващите символи нямат видимо представяне. Означават се по следния начин:

‘\символ’
Фиг. 5.1 представя някои от тези символи, а също и предназначението им.

Символ	Предназначение
\a	издава звуков сигнал
\b	връща курсора един символ назад
\n	предизвиква преминаване на нов ред
\r	връща курсора в началото на реда
\t	хоризонтална табулация
\v	вертикална табулация
\0	нулев символ, край на низ

Фиг. 5.1 Някои управляващи символи

Елементите от множеството от стойности на типа символен са константите на този тип. Наричат се още **символни литерали**.

Променлива величина, множеството от допустимите стойности, на която съвпада с множеството от стойности на типа символен, се нарича **символна променлива** или **променлива от тип символен**. Дефинира се по общоприетия начин.

Примери:

```
char c1;
char c2 = '+';
```

Дефиницията свързва променливите с множеството от стойности на типа char или с конкретна стойност от това множество като отделя по 1 байт оперативна памет за всяка от тях. Стойността на тази памет е неопределена или е константата, свързана с дефинираната променлива, в случай, че дефиницията е с инициализация.

След дефиницията от примера по-горе, имаме:

```
   ОП
   c1      c2
   _       +
   1в      1в
```

Стойността на клетките от паметта, именувани със c1 е неопределена, а това на именуваните със c2 е '+'. В същност, вместо '+' в паметта е записан кодът (вътрешното представяне) на символът '+' – цялото число 43.

Елементите от множеството от стойности на типа char е наредено. Всеки символ е свързан с код, съгласно кодирането ASCII. При това кодиране, за управляващите символи се използват целите числа от 0 до 31 включително. Останалите символи се кодират с числата от 32 до 255. Фиг. 5.2 дава частична представа за наредбата на някои символи.

ASCII-кодове на някои символи

0	31	32	48 49 … 57	…	65 66 … 90	…	97 98 … 122
управляващи символи	интервал	0 1 … 9	…	A B … Z	…	a b … z	

Фиг. 5.2 ASCII-кодове на някои символи

Забележка: Цифрите, а също главните и малки букви на латинската азбука имат последователни кодове.

Операции над символни данни

Намиране на кода на символ

Извършва се чрез израза (int)c1, където c1 е символна константа или променлива.

Пример: Операторът

```
cout << (int)'F';
```

извежда кода на главната латинска буква F.

Намиране на символ по даден код

Осъществява се чрез израза (char)<цял_аритметичен_израз>. Стойността на <цял_аритметичен_израз> трябва да е неотрицателно цяло число. Ако не е от интервала [0, 255], намира се остатъкът от делението по модул 256.

Пример: Операторът

```
cout << (char)65 << '\t' << (char)(3*256+65);
```

извежда два пъти символа A главно, латинско, разделени с табулация.

Аритметични операции

Всички аритметични операции, допустими над целочислени данни са допустими и за данни от тип от тип char. Извършват се над кодовете им. Резултатът е цяло число. Така се разширява синтаксисът на целите изрази.

Примери:

1. c1 + 15 - 'A' е цял аритметичен израз, стойността на който се получава като се събере кодът на символа, свързан със c1, с 15 и от полученото цяло число се извади 65 (кодът на 'A').
2. c1%2 е цял аритметичен израз.

Присвояване на стойност

На променлива от символен тип може да се присвояват символни константи и променливи, а също стойностите на цели аритметични изрази. В последния случай, добрият стил на програмиране изисква да се конвертира явно типът на целия аритметичен израз до символен.

Примери:

```
c1 = 'A'
c2 = c1;
c1 = c1 + c2 - 15;
```

В третия пример компилаторът автоматично ще извърши преобразуването от тип цял в тип char. По-добре е явно това да се укаже, т.е. c1 = (char)(c1 + c2 – 15); е по-добър вариант.

Намиране на символа пред и след указан символ

Символът, пред символа с е (char)(c-1), а този след с е (char) (c+1), където с е константа или променлива от символен тип.

Забележка: Ако с е променлива от символен тип, това може да се постигне и чрез c++ и c--, но се променя стойността на с.

Логически операции

Всички логически операции са допустими и над данни от символен тип. Изпълняват се над кодовете им. Резултатът е от булев тип. Освен това, символна константа или променлива, поставена на място на условие, изпълнява ролята на булев израз.

Операции за сравнение

Над данни от тип символен могат да се прилагат стандартните инфиксни оператори за сравнение:

Оператор	Операция
==	сравнение за равно
!=	сравнение за различно

>	сравнение за по-голямо
>=	сравнение за по-голямо или равно
<	сравнение за по-малко
<=	сравнение за по-малко или равно

Сравняват се кодовете. Резултатът е булев.

Примери:

‘a’ <= ‘z’ е true ‘a’ <= ‘0’ е false

‘0’ < ‘9’ е true ‘A’ > ‘a’ е false

‘x’ != ‘X’ е true

Данни от символен тип могат да се сравняват чрез горните оператори и с цели аритметични изрази. Резултатът е булев.

Примери:

‘a’ == 0 c1 != 65 c2 >= 122

Така се разширява синтаксисът на булевите изрази.

Въвеждане

Осъществява се чрез оператора >>. В този случай, операторът >> не е чувствителен към символите: интервал, хоризонтална и вертикална табулации и преминаване на нов ред.

Примери:

1. Операторът

```
cin >> c1 >> c2;
```

се изпълнява по следния начин: Ако в буфера на клавиатурата няма символи, различни от интервали, табулации и знаци за преминаване на нов ред, настъпва пауза до въвеждане на два символа, различни от интервали, табулации и знаци за преминаване на нов ред и разделени със същите знаци. Те обслужват последователно c1 и c2. В противен случай, не настъпва пауза, а символите от буфера се свързват последователно със c1 и c2.

1. Програмният фрагмент

```
cout << “c1= “;
char c1;
cin >> c1;
cout << “c2= “;
```

```
char c2;
cin >> c2;
```
се изпълнява по следния начин: Върху екрана се появява подсещането c1=. Операторът cin >> c1; очаква въвеждане на един символ, различен от интервал, табулация и знак за преминаване на нов ред. Въвеждането трябва да завършва със знака за преминаване на нов ред. Водещите интервали, табулации и знаци за преминаване на нов ред се пренебрегват. След това се повтарят същите действия за променливата c2 освен ако след подсещането за стойност на c1 не са въведени два символа (може да са без разделител по между си), различни от интервали, табулации и знаци за преминаване на нов ред. Тогава първият символ се свързва с променливата c1, а вторият – с променливата c2.

Извеждане

Осъществява се по стандартния начин.

Пример: Операторът

```
cout << c1;
```
ще изведе символът, свързан със c1, а операторът
```
cout << c1+c2;
```
ще изведе цялото число, равно на сумата от кодовете на c1 и c2, тъй като c1+c2 е аритметичен израз. Ако се налага да се изведе символът, чийто код е равен на сумата от кодовете на c1 и c1, трябва да се използва операторът
```
cout << char(c1+c2);
```

Допълнения:

1. Типът char е допустим за тип на switch-израз, т.е. допустим е фрагментът:
```
char c;
cin >> c;
switch (c)
{case ‘a’: …
 case ‘e’: …
 …
```

}

2. Символните константи могат да съдържат и повече от един символ, например ‘ah’, ‘НАНА’ са символни константи. Тази възможност няма да бъде застъпена в нашия курс.

3. Чрез модификаторите signed и unsigned се получават разновидности на типа char. Те също няма да бъдат застъпени, тъй като реализацията Visual C++ 6.0 не е чувствителна към тях.

Задачи върху тип символен

Нека се върнем към задача 44. Програма Zad44.cpp дава едно нейно решение.

```
// Program Zad44.cpp
#include <iostream.h>
#include <iomanip.h>
int main()
{char op = '+';
 double result = 0.0;
 do
 {double arg;
  cin >> arg;
  switch (op)
  {case '+': result = result + arg; break;
   case '-': result = result - arg; break;
   case '*': result = result * arg; break;
   case '/': result = result / arg;
  }
  cin >> op;
 } while (op != '=');
 cout << setprecision(5) << setiosflags(ios::fixed);
 cout << setw(15) << result << "\n";
 return 0;
}
```

Допълнение: Операторът

```
   switch (op)
     {case '+': result = result + arg; break;
      case '-': result = result - arg; break;
      case '*': result = result * arg; break;
      case '/': result = result / arg;k;
     }
```
може да се запише и в следната съкратена форма:
```
   switch (op)
     {case '+': result += arg; break;
      case '-': result -= arg; break;
      case '*': result *= arg; break;
      case '/': result /= arg;
     }
```

Задача 45. Да се напише програма, която въвежда малка буква от латинската азбука и извежда съответната й главна буква.

Програма Zad45.cpp решава задачата.
```
// Program Zad45.cpp
#include <iostream.h>
int main()
{char c;
 cin >> c;
 if (c < 'a' || c > 'z')
 {cout << "Incorrect Input! \n";
  return 1;
 }
 cout << (char)(c - 'a' + 'A') << '\n';
 return 0;
}
```

Задача 46. Да се напише програма, която извежда цифрите, главните и малките букви на латинската азбука и ASCII кодовете им. Всеки символ и кодът му да са на един и същ ред. За разделител между символ и код да служи знакът за хоризонтална табулация.

Програма Zad46.cpp решава задачата.

```
// Program Zad46.cpp
#include <iostream.h>
int main()
{for(char ch = '0'; ch <= '9'; ch++)
 cout << ch << '\t' << (int)ch << '\n';
 for(ch = 'A'; ch <= 'Z'; ch++)
 cout << ch << '\t' << (int)ch << '\n';
 for(ch = 'a'; ch <= 'z'; ch++)
 cout << ch << '\t' << (int)ch << '\n';
 return 0;
}
```

5.2 Тип изброен

Типът е стандартен. За разлика от другите, досега разгледани типиве, той се дефинира от програмиста като се изброяват константите му. Затова се нарича още потребителски дефиниран тип.

Дефиниция на тип изброен

Фиг. 5.3 илюстрира синтаксиса на дефиницията му.

Дефиниране на тип изброен

<дефиниция_на_тип_изброен> ::=
enum [<име_на_тип>]$_{опц}${<идентификатор$_1$> [= <константен_израз$_1$>]$_{опц}$,
<идентификатор$_2$> [= <константен_израз$_2$>]$_{опц}$,
…
<идентификатор$_n$> [= <константен_израз$_n$>]$_{опц}$};
<име_на_тип> ::= <идентификатор>

където

- enum е запазена дума (съкращение от enumerate – изброявам);
- <константен_израз$_i$> (i = 1, 2,…, n) е константен израз от интегрален тип.

Фиг. 5.3 Дефиниране на тип изброен

Името на типа, а също <константен_израз$_i$> (i = 1, 2, …, n) могат да се пропуснат. Пропускането на <константен_израз$_i$> става заедно със знака =. Изброените идентификатори трябва да са различни не само в рамките на една дефиниция, а и в тези на всички дефиниции на изброени типове, в рамките на даден модул.

Примери:

```
enum Weekday{SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
             THURSDAY, FRIDAY, SATURDAY};
enum Name{IVAN=5, PETER=3, MERY=8, SONIA=6, VERA=10};
enum Id{A1, A2, A3, A4=8, A5, A6=10, A7, A8};
enum {FALSE, TRUE};
```

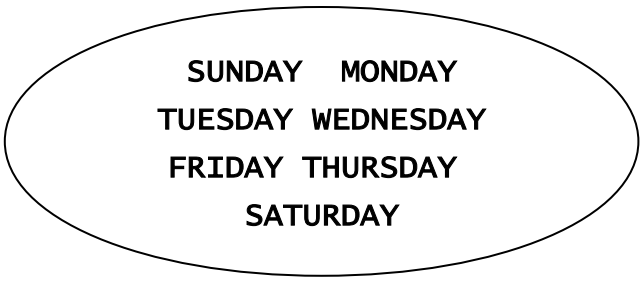
Забелязваме, че в четвъртата дефиниция на примера е пропуснато името на типа. Такива типове се наричат анонимни изброени типове.

Типът изброен дава един начин за дефиниране на константи.

Множество от стойности

Състои се от всички изброени в дефиницията на типа идентификатори.

Пример:



множество от стойности на тип Weekday

Елементите от множеството от стойности на даден изброен тип са константите на този тип. Затова ще ги означаваме с главни букви.

Примери: SATURDAY е константа от тип Weekday, VERA е константа от тип Name. TRUE и FALSE също са константи, но от безименен тип изброен.

Променлива величина, приемаща за стойности константи от множеството от стойности на някакъв изброен тип, се нарича **променлива от този тип изброен**. Дефинира се по общоприетия начин.

Примери:

```
Weekday d1, d2 = SUNDAY;
Name a;
Name b = VERA;
Id x = A2, y = A7;
```

Тук d1 и d2 са променливи от тип Weekday, а и b – от тип Name, а x и y – от тип Id.

Дефиницията свързва променливите от даден тип изброен с множеството от стойности на типа или с конкретна стойност от това множество като отделя по 1 или 4 байта ОП за всяка от тях (за реализацията Visual C++ 6.0 - 4 байта ОП). Стойността на тази памет е неопределена или е константата, свързана с дефинираната променлива, в случай, че тя е инициализирана.

След дефиницията от примера по-горе, имаме:

ОП

d1	d2	a	b	x	y
-	SUNDAY	-	VERA	A2	A7
4 байта	4 байта	4 байта	4 байта	4 байта	4 байта

Стойността на паметта, именувана с d1 и a, е неопределена, това наименуваната с d2 е SUNDAY, на b – VERA, на x и y – A2 и A7 съответно.

В същност, вместо SUNDAY, VERA, A2 и A7, в паметта са записани кодовете им (вътрешните им представяния).

Вътрешните представяния на константите от изброени типове се определят по следния начин:

- Ако не са указани стойности, по подразбиране първият идентификатор получава стойност 0, а всеки следващ – стойност с единица по-голяма от стойността на предходния.

Пример: Вътрешните представяния на константите от тип Weekday са:

SUNDAY - 0, MONDAY - 1, TUESDAY – 2 и т.н. SATURDAY – 6.

• Ако всички идентификатори са свързани със стойности, вътрешното представяне на всяка константа от такъв изброен тип е указаната стойност.

Пример: Вътрешните представяния на константите от тип Name са:

IVAN - 5, PETER - 3, MARY - 8, SONIA - 6, VERA - 10.

• Ако някои идентификатори са свързани със стойности, а други – не, вътрешното представяне на идентификатор, за който е указана стойност е указаната стойност, а на всеки идентификатор, за който не е указана стойност – е вътрешното представяне на идентификатора пред него, увеличено с 1. Вътрешното представяне на първият идентификатор, ако не е указана стойност за него, е 0.

Пример: Вътрешните представяния на константите от тип Id са:
A1 - 0, A2 - 1, A3 - 2, A4 - 8, A5 – 9, A6 – 10, A7 = 11 и A8 – 12.

Възможни са и дефиниции на променливи от тип изброен от вида:

enum {RED, BLUE = 4, WHITE, BLACK = 7} color1, color2;

Тази дефиниция показва, че вместо име на тип, може да се използва *дефиниция* на анонимен изброен тип.

Операции върху данни от тип иьброен

Намиране на кода на константа от тип изброен

Извършва се чрез израза (int)x, където x е константа или променлива от изброен тип.

Пример: Операторът

cout << (int)FRIDAY;

извежда 5 - кода на FRIDAY.

Намиране на константа от тип изброен по даден код

Осъществява се чрез израза (<име_на_тип_изброен>)<код>.

Пример: Изразът d1 = (Weekday)4 намира THURSDAY.

Аритметични операции

Всички аритметични операции, допустими над целочислени данни са допустими и за данни от тип изброен. Извършват се над кодовете на данните. Резултатът е цяло число.

Примери:

1. d1+d2-4 е цял аритметичен израз, стойността на който се получава като се съберат кодовете на d1 и d2 и от полученото се извади 4.

2. d1 % 5 е цял аритметичен израз, изразяващ остатъка от делението на кода на d1 на 5.

Присвояване на стойност

На променлива от тип изброен може да се присвои която и да е константа от множеството от стойности на типа, от който е променливата, а също и стойността на променлива от същия изброен тип.

Пример:

```
d1 = WEDNESDAY;
d2 = d1;
```

Присвояването

```
d1 = d2 - 2;
```

не е допустимо, но

```
d1 = (Weekday)(d2 - 2);
```

вече е допустимо.

Логически операции

Всички логически операции, са допустими и за данни от тип изброен. Извършват се на кодовете на данните. Резултатът е булев.

Операции за сравнение

Данни от един и същ тип изброен могат да се сравняват чрез стандартните инфиксни оператори за сравнение:

Оператор	Операция
==	сравнение за равно

!=	сравнение за различно
>	сравнение за по-голямо
>=	сравнение за по-голямо или равно
<	сравнение за по-малко
<=	сравнение за по-малко или равно

Сравняват се кодовете. Резултатът е булев.

Примери:

SUNDAY <= FRIDAY е true

VERA < PETER е false

A3 != A7 е true

Въвеждане

Не е възможно въвеждане на стойност на променлива от някакъв изброен тип чрез оператора >>, т.е. операторът

cin >> d1;

е недопустим.

Извеждане

Операторите

cout << <константа_от_тип_изброен>;

или

cout << <променлива_от_тип_изброен>;

извеждат кода на константата или променливата.

Следният програмен фрагмент дефинира изброен тип с име Weekday и извежда стойността на променлива от този тип.

```
…
enum Weekday{SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
             THURSDAY, FRIDAY, SATURDAY};
Weekday d;
d = FRIDAY;
switch(d)
{case SUNDAY: cout << "SUNDAY \n"; break;
 case MONDAY: cout << "MONDAY \n"; break;
```

```
 case TUESDAY: cout << "TUESDAY \n"; break;
 case WEDNESDAY: cout << "WEDNESDAY \n"; break;
 case THURSDAY: cout << "THURSDAY \n"; break;
 case FRIDAY : cout << "FRIDAY \n"; break;
 case SATURDAY: cout << "SATURDAY \n";
}
```

Типът изброен е потребителски дефиниран тип. Заради това, всеки потребител може да предефинира операторите ++, --, << и др. за работа с данни от дефиниран от него изброен тип.

Задача 47. Да се напише програма, която намира средната седмична температура.

Програма Zad47.cpp решава задачата.

```
     // Program Zad47.cpp
#include <iostream.h>
#include <iomanip.h>
int main()
{enum Weekday{SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
              THURSDAY, FRIDAY, SATURDAY};
 Weekday d;
 double s = 0.0;
 for(d = SUNDAY; d <= SATURDAY; d = (Weekday)(d+1))
 {double t;
  cout << "t= ";
  cin >> t;
  s = s + t;
 }
 cout << setprecision(2) << setiosflags(ios :: fixed);
 cout << setw(10) << s/7 << "\n";
 return 0;
}
```

Забележете частта <корекция> на оператора for. Операторът за просвояване d++ не е допустим за коректор. Тъй като вътрешното представяне на d е цяло число, възможно е да се увеличи с 1, т.е. d+1

е допустимо, но присвояването му на променливата d от тип Weekday не е възможно. Налага се преобразуване на цялото число d+1 в тип Weekday.

Изброеният тип задава крайно множество от константи. Единствената разлика между константите от тип изброен и еквивалентните им const – дефиниции е, че с тях не е свързан адрес от паметта.

Основното предимство от използването на тип изброен е подобряването читаемостта на програмата. Това се счита за добър стил за програмиране.

Задачи

Задача 1. Да се напише програма, която извежда върху екрана следната таблица:

а)
```
A B C D E
B C D E F
C D E F G
D E F G H
E F G H I
```

б)
```
A
B C
C D E
D E F G
E F G H I
```

Задача 2. Да се напише програма, която въвежда стойности на естествените числа n, m, p и q и намира всички редици от операции (op1, op2, op3) сред +, -, * и /, така че като се зместят в израза (((n op1 m) op2 p) op3 q), получената стойност да е равна на a (а дадено цяло число).

Задача 3. Да се напише програма, която въвежда стойности на естествените числа n, m, p и q и установява дали съществува редица от операции (op1, op2, op3) сред +, -, * и /, така че като се зместят в израза (((n op1 m) op2 p) op3 q), получената стойност да е равна на a (а дадено цяло число).

Допълнителна литература

1. B. Stroustrup, C++ Programming Language. Third Edition, Addison – Wesley, 1997.
2. М. Тодорова, Програмиране на Паскал, Полипринт, София, 1993.