Основы алгоритмизации и программирования. Преобразование данных. Оператор "?". Оператор ","

Лекция 6

Привезенцев Д.Г.

Муромский инстиут Владимирского государственного университета
Очная форма обучения

7 ноября 2021 г.

Понятие преобразования типов

Определение

Преобразование значения переменной одного типа в значение другого типа называется приведением типа и бывает явным и неявным.

- При явном приведении перед выражением следует указать в круглых скобках имя типа, к которому необходимо преобразовать исходное значение.
- При неявном приведении преобразование происходит автоматически, по правилам, заложенным в языке Си.

Примеры приведения типа

Пример явного преобразования типов:

```
1       int x = 5;
2       double y = 15.3;
3       x = (int) y;
4       y = (double) x;
```

Пример неявного преобразования типов

```
1           int x = 5;
2           double y = 15.3;
3           y = x;
4           x = y;
```

Неявное приведение типа при арифметических операциях

типы операндов	тип результата
float / float	float
float / int	float
int / flaot	float
int / int	int

В последнем случае (и только в нем) осуществляется целочисленное деление с отбрасыванием остатка.

Зачем нужно явное приведение типов если есть неявное?

Чему будет равна переменная Z после выполнения следующего кода?

```
1        int x = 12;
2        int y = 7;
3        double z = x/y;
```

В данном выражении разбор начнется с операции наиболее высокого приоритета — с деления, и только потом дело дойдет до присваивания.

Т.е. операция с точки зрения типов будет такой:

```
z = (double)( (int)x / (int)y )
```

Итого z = 1.0

Зачем нужно явное приведение типов если есть неявное?

А в этом случае чему будет равно значение переменной Z?

```
1       int x = 12;
2       int y = 7;
3       double z = (double)x/y;
```

В данном выражении разбор начнется с операции наиболее высокого приоритета — с унарной операции приведения типов, и только потом дело дойдет до деления.

Т.е. операция с точки зрения типов будет такой:

```
z = (double)( (double)x / (int)y )
```

Итого z = 1.714285714285714

Результаты преобразования типов

Целевой тип	Исходный тип	Возможные потери информации
signed char	unsigned char	Возможно изменение знака
char	short int	Старшие 8 бит
char	int (16 бит)	Старшие 8 бит
char	int (32 бита)	Старшие 24 бита
short int	int (16 бит)	Нет
short int	int (32 бита)	Старшие 16 бит
int (16 бит)	long int	Старшие 16 бит
int (32 бита)	long int	Нет
float	double	Точность, результат округляется
double long	double	Точность, результат округляется

Оператор ?

Оператор **?** имеет следующий вид:
выражение1 ? выражение2 : выражение3;

Тернарные операции, как и операции условия, могут быть вложенными.
Для разделения вложенных операций используются круглые скобки.

Оператор ? работает следующим образом:
вычисляется **выражение1**; если оно истинно, то вычисляется **выражение2** и все выражение получает это значение; а если оно ложно, то вычисляется **выражение3** и все выражение получает это значение.

Пример

Например:

```
1         x = 10;
2         y = x > 9 ? 100 : 200;
```

В данном примере y получает значение 100. Если бы x было меньше, чем 9, то y получило бы значение 200.

Ниже приведен фрагмент программы, выполняющий такие же действия, но с использованием операторов if/else:

```
1         x = 10;
2         if (x > 9)
3             y = 100;
4         else
5             y = 200;
```

Пример

Листинг 1: Пример использования оператора ?

```
1  #include <stdio.h>
2  int main()
3  {
4    int key; // объявляем целую переменную key
5    printf("Enter the number, 1 or 2: ");
6    scanf("%d", &key); // вводим значение переменной key
7    key == 1 ? printf("\n You Choose number 1") :
8      (key == 2 ? printf("\n You Choose number 2") :
9        printf("\n Number 1 and 2 was not choosen"));
10   getchar();
11   getchar();
12   return 0;
13 }
```

Пример

Листинг 2: Нахождение минимального из двух чисел

```
1  #include <stdio.h>
2  int main()
3  {
4      int a, b, c;
5      scanf("%d %d", &a, &b);
6      c = (a > b) ? b : a;
7      printf("%d", c);
8      getchar();
9      getchar();
10     return 0;
11 }
```

Листинг 3: Вывод название введеного числа

```
#include <stdio.h>
int main()
{
    int n=0;
    char *strnum  = (char*) malloc(10);
    while((n>=0) && (n<=10))
    {
        printf("Please enter an integer\r\n");
        printf("To exit the program, enter a number greater than 10 ...\n");
        scanf("%d", &n);
        strnum = n == 0 ? "zero":
                 n == 1 ? "one":
                 n == 2 ? "two":
                 n == 3 ? "three":
                 n == 4 ? "four":
                 n == 5 ? "five":
                 n == 6 ? "six":
                 n == 7 ? "seven":
                 n == 8 ? "eight":
                 n == 9 ? "nine":
                 n == 10 ? "ten":
                 "greater than 10 or less than 0";
        printf("The number entered is %s.\n", strnum);
    }
    getchar();getchar();
    return 0;
}
```

Листинг 4: Как НЕ надо делать (вложеные операторы без скобок)

```
1  #include <stdio.h>
2  int main()
3  {
4      int a = 1;
5      int b = 2;
6      int c = 3;
7      int mid;
8
9      mid = (a>b)?(a>c)?(c>b)?c:b:a:(b>c)?(c>a)?c:a:b;
10
11     printf("%d", mid);
12     getchar();
13     getchar();
14     return 0;
15 }
```

Листинг 5: Как надо делать (вложеные операторы со скобками)

```
1  #include <stdio.h>
2  int main()
3  {
4      int a = 1;
5      int b = 2;
6      int c = 3;
7      int mid;
8
9      mid = (a>b)?((a>c)?((c>b)?c:b):a):((b>c)?((c>a)?c:a):b);
10
11     printf("%d", mid);
12     getchar();
13     getchar();
14     return 0;
15 }
```

Что такое Оператор «,» (запятая)

Оператор «запятая» используется для связки нескольких выражений. Левая сторона оператора «запятая» всегда вычисляется как void (то есть не выдающее значения). Это означает, что значение выражения, находящегося с правой стороны, станет значением разделенного запятыми выражения.

Оператор «,» принимает два аргумента, выполняет первый, после этого выполняет второй и возвращает его значение.

Что такое Оператор «,» (запятая)

```
int x = (y = 3, y + 1);
```

Сначала присваивается 3 переменной y, а затем 4. переменной x. Скобки необходимы, поскольку оператор «запятая» имеет более низкий приоритет по сравнению с оператором присваивания.

Оператор «запятая» вызывает выполнение последовательности действий. Когда он используется с правой стороны оператора присваивания, то присваиваться будет значение последнего выражения, стоящего в разделенном запятыми списке. Ниже приведен еще один пример:

```
int y = 10;
int x = (y = y - 5, 25 / y);
```

После выполнения x получит значение 5, поскольку исходным значением y было 10, а затем оно уменьшилось на 5. Затем 25 поделили на полученное 5 и получили результат.

Еще примеры

```
int x = 0;
int y = 2;
int z = (++x, ++y);
```

Почти в каждом случае, выражение, в ко тором есть оператор Запятая, лучше записывать в виде отдельных инструкций. Выше приведенный код корректнее будет записать следующим образом:

```
int x = 0;
int y = 2;
++x;
++y;
int z = y;
```

Приоритет оператора «,» (запятая)

Оператор Запятая имеет самый низкий приоритет из всех операторов (даже ниже, чем в оператора присваивания), поэтому следующие две строки кода делают не одно и то же:

```
1  z = (a, b);
2  z = a, b;
```

В первой строчке сначала вычисляется выражение (a, b), которое равняется значению b, а затем результат присваивается переменной z.
Вторая строчка вычисляется как

```
(z = a), b
```

, поэтому переменной z присваивается значение a, переменная b - игнорируется

Листинг 6: Полный пример оператора

```
1 #include <stdio.h>
2  int main()
3  {
4      int a, b, c;
5      c = (a = 3, b = 4);
6      printf("%d", c);
7      getchar();
8      return 0;
9  }
```

«Коварство» оператора «,»

Особенности оператора запятая часто приводит к следующей ошибке. Вместо десятичной точки

```
a = 1.5;
```

пишут

```
a = 1,5;
```

Таким образом, переменная a будет иметь значение 5.

Еще пример

Для оператора «,» гарантированно выполнение по порядку, слева направо. Оператор запятая поэтому используется в условиях, когда нужно выполнить несколько действий или получить значение.

Листинг 7: Использование оператора в условном выражении цикла

```
1  #include <stdio.h>
2  int foo(int a)
3  {
4      return a % 5 + a*a;
5  }
6  int main()
7  {
8      int i = 0, mod;
9      while (mod = foo(i), mod < 100)
10     {
11         printf("%d ", i++);
12     }
13     getchar();
14     return 0;
15 }
```