1.12. Дополнительные типы данных

Tuesday, July 28, 2020 21:08

В данном уроке мы разберем перечисляемый тип данных (*enumerated data type*), оператор `typedef`, а также конверсию одних типов данных в другие.

Перечисляемые типы данных (*enumerated data types*)

Перечисляемые типы позволяют создавать собственные типы данных с фиксированным набором их возможных значений. Такие типы создаются через ключевое слово `enum`:

```
enum primaryColor { red, yellow, blue };
```

Сначала идет сам `enum`, затем название типа и затем в скобках названия допустимых значений этого типа. Таким образом, переменным типа `enum primaryColor` можно присвоить только одно из трех значений `red`, `yellow` или `blue`. Другие значения **недопустимы**. Если попытаться присвоить типу дополнительное значение с недопустимым названием (т.е. отсутствующим в списке), компилятор выдаст ошибку.

Создание переменных типа enum `primaryColor` делается по аналогии со структурами:

```
enum primaryColor myColor, gregsColor;
```

Эта инструкция создает две переменные типа `primaryColor`. Этим переменным могут быть присвоены только значения переменных `red`, `yellow` и `blue`. То есть, допустимы инструкции вроде

```
myColor = red;
```

и

```
if ( gregsColor == yellow )
   ...
```

У перечисляемых типов может отсутствовать имя и по аналогии со структурами можно задавать переменные прямо при объявлении типа:

```
enum { east, west, south, north } direction;
```

Также помните, что названия идентификаторов не должны совпадать с названиями других идентификаторов и переменных.

Приведем еще один пример использования `enum` – переменная для хранения текущего месяца:

```
enum month { January, February, March, April, May, June,
             July, August, September, October, November, December };
```

Компилятор C работает со типами `enum` как с **беззнаковыми целочисленными константами** (иногда как с `int` или `char`). Он присваивает идентификаторам списка порядковые номера, начиная с нуля: `January` = 0, `February` = 1 и т.д. То есть, программа

```
enum month thisMonth;
      ...
thisMonth = February;
```

присвоит 1 переменной `thisMonth`, а не само имя "February".

Если нужно чтобы определенному идентификатору присваивался не порядковый номер, а произвольное число, его нужно указать при создании типа:

```
enum direction { up, down, left = 10, right };
```

при этом счет последующих слов продолжится с присвоенного числа. То есть, если мы присвоили идентификатору `left` значение 10, то `right` = 11.

Напишем программу, которая будет использовать перечисляемый тип для вывода количества дней в месяцах.

```
1   //  Program to print the number of days in a month
2
3   #include <stdio.h>
4
5   int main (void)
6   {
7       enum  month  { January = 1, February, March, April, May, June,
8                      July, August, September, October, November, December };
9       enum  month  aMonth;
10      int          days;
11
12      printf ("Enter month number: ");
13      scanf ("%u", &aMonth);
14
15      switch (aMonth ) {
16         case January:
17         case March:
18         case May:
19         case July:
20         case August:
21         case October:
22         case December:
23                    days = 31;
24                    break;
25         case April:
26         case June:
27         case September:
28         case November:
29                    days = 30;
30                    break;
31         case February:
32                    days = 28;
33                    break;
34         default:
35                    printf ("bad month number\n");
36                    days = 0;
37                    break;
38      }
39
40      if ( days != 0 )
41         printf ("Number of days is %i\n", days);
42
43      if ( aMonth  == February )
44         printf ("...or 29 if it's a leap year\n");
45
46      return 0;
47  }
```

Идентификаторы могут иметь одинаковые значения:

```
enum  switch  { no=0, off=0, yes=1, on=1 };
```

Присвоить переменной значение идентификатора можно также через оператор приведения типа. То есть, если `monthValue` = 5 – идентификатор типа `month`, то выражение

```
thisMonth = (enum month) (monthValue - 1);
```

присвоит `thisMonth` значение 4.

Директива **typedef**

Язык C позволяет присвоить типу данных альтернативное название. Для этого используется директива `typedef`. Инструкция

```
typedef  int  Counter;
```

задает альтернативное название для типа `int`, – `Counter`. После этого целочисленные переменные можно задавать как тип `Counter`:

```
Counter  j, n;
```

Компилятор C будет работать с этими переменными как с обычными `int`-ами.

В большинстве случаев, `typedef` можно заменить директивой `#define`:

```
#define  Counter  int
```

но так как `typedef` обрабатывается именно компилятором, а не препроцессором, как `#define`, у `typedef` больше гибкости в работе с производными типами данных. Например, инструкцию

```
typedef  char  Linebuf [81];
```

невозможно реализовать через `#define`.

Также через `typedef` можно создавать указатели на тип:

```
typedef  char *StringPtr;
```

А также давать название структурам:

```
1    typedef  struct
2             {
3                  int   month;
4                  int   day;
5                  int   year;
6             } Date;
7            ...
8    Date  birthdays[100];
```

При работе с большими программами желательно помещать часто используемые `typedef`-ы в отдельный файл и потом добавлять его через `#include`.

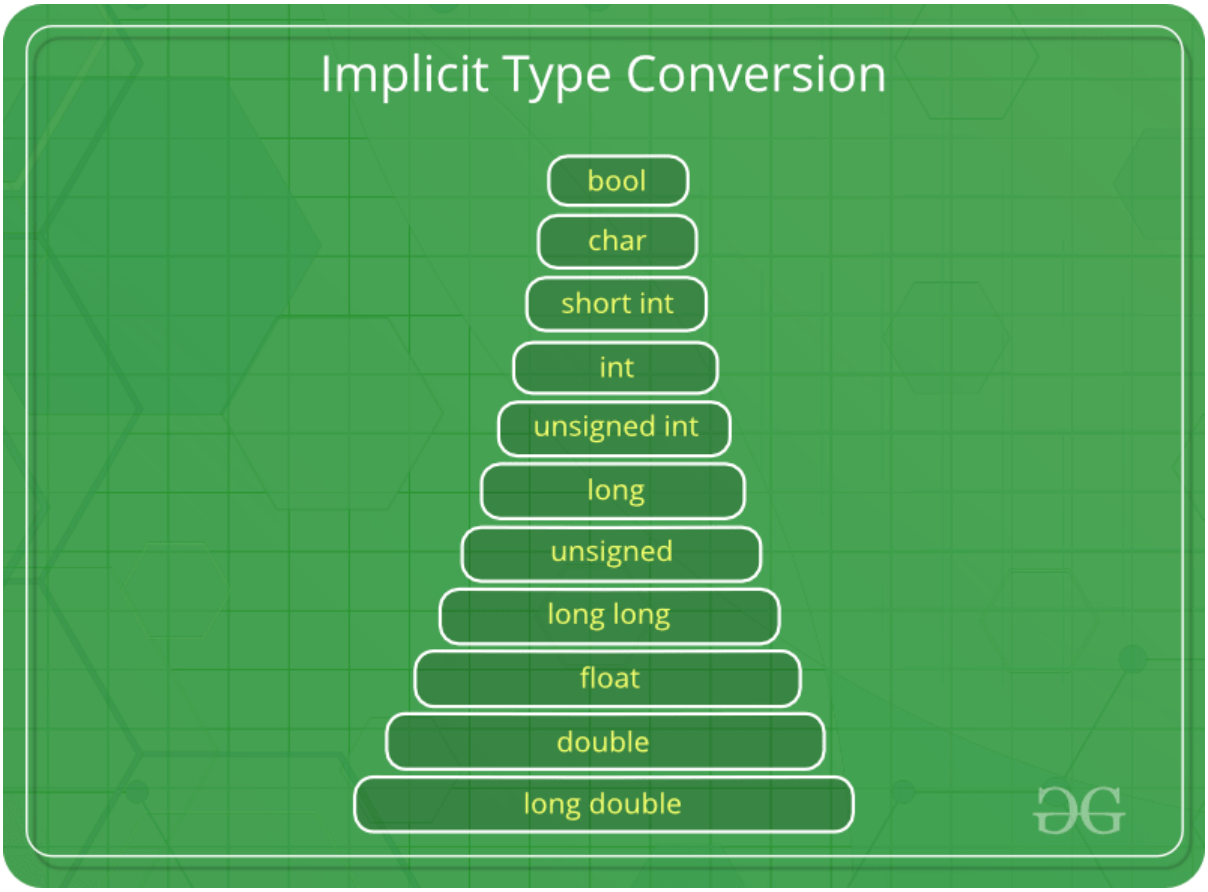
Также важно понимать, что `typedef` не определяет новый тип, а лишь новое имя для существующего типа.

Конверсия типов данных

В уроке 1.2 (*Типы переменных*) упоминалось о неявной конверсии типов данных при вычислении выражений. Разбирались случаи, когда `float` превращался в `int` и наоборот. Также было показано, как использовать оператор приведения типа для явной конверсии типа данных. Так, в выражении

```
average = (float) total / n;
```

значение переменной `total` конвертируется во `float` **до того**, как выражение будет вычислено, что дает гарантию, что будет произведено деление типов `float`, а не `int`. Компилятор C следует строгим правилам, когда производится работа с выражениями, состоящими из разных типов данных. Приведем порядок проведения конверсий типов в выражении из двух переменных разных типов.



То есть, нужно следить за возможной **потерей знака** при неявной конверсии в тип `unsigned` и за **переполнением** при конверсии `long long` во `float`.

Конверсия операторов довольно четко определена и почти не имеет подводных камней, но некоторые моменты нужно учитывать:

1. В некоторых системах корректная конверсия конверсия `char` в `int` может требовать дополнительную операцию расширению знака (*sign extension*), при условии что `char` не был объявлен как `unsigned`.
2. Конверсия знакового `int` в бóльшую разрядность приводит к продлению знака слева, а конверсия беззнакового `int` приводит к заполнению нулями свободного пространства слева.
3. Конверсия любого значения в _Bool результирует единицей, если это значение не 0, в таком случае он останется нулем.
4. Конверсия `int` в меньшую разрядность приводит к "обрезанию" левых бит числа.
5. Конверсия `float` в `int` происходит с обрезанием дробной части числа. Если разрядности `int` недостаточно, чтобы вместить всё число, результат будет неопределенным, как и результат конверсии отрицательного `float` в беззнаковый `int`.
6. При конверсии `float` в меньшую разрядность некооторые системы округляют, а некоторые не округляют числа перед обрезкой бит.

Разберем выражение, в котором `f – float`, `i – int`, `l – long int`, а `s – short int`:

```
f * i + l / s
```

Сначала идет операция `f * i`, `– float` на `int`. Согласно **п.3**, `i` будет сконвертирован во `float` и результат умножения тоже будет типа `float`.

Затем идет `l / s`, т.е. деление `long int` на `short int`. Согласно **п.4**, `short int` будет преобразован в `long int` и результат деления тоже будет типа `long int`.

Наконец, идет сложение результатов предыдущих вычислений: `float` и `long int`. Согласно **п.3**, `long int` будет преобразован во `float` и результат тоже будет типа `float`.

Помните, что если нужно явно указать конверсию типа, можно использовать оператор приведения типа. Например, если мы хотим сохранить дробную часть при делении `l` на `s`, можно конвертировать один из операндов, например, во `float`:

```
f * i + (float) l / s
```

Здесь `l` будет конвертирован в тип `float`, потому что у оператора приведения типа более высокий приоритет, чем у оператора деления. Так как `l` теперь `float`, согласно **п.3** `s` тоже будет конвертирован во `float`.

Расширение знака (*sign extension*)

Когда `signed int` или `signed short int` конвертируется в бóльшуюо разрядность, его знак продлевается влево.

Например, 8-битное число 10 в двоичном виде будет `0000 1010` и если оно конвертируется в 16-битное число, его знак (т.е. 0, означающий положительное значение) продлевается влево:

```
0000 0000 0000 1010
```

Число –15 в двоичном виде будет `1111 0001` и если оно конвертируется в 16-битное число, его знак также продлится влево, но в этот раз вместо нулей будут единицы, так как число отрицательное:

```
1111 1111 1111 0001
```

Таким образом, как число 10, так и число –15 после конверсии сохраняют и свой знак, и свое значение. В некоторых системах символы обрабатываются как знаковые числа. Это значит, что когда такой символ конвертируется в `int`, происходит расширение знака. Если в переменной символа хранится символ из стандартного набора ASCII, эффект расширения знака не будет проблемой. Но если символ из нестандартного набора, тут уже надо быть осторожным. Например, в Mac OS, символ `'\377'` конвертируется в –1, потому что его значение будет отрицательным, если его обработать как знаковое 8-битное значение.

Вспомните, что в C символ можно объявить как `unsigned char`, что поможет избежать подобных проблем, потому что его значение всегда будет положительным.

Если же нужно явно указать, что переменная должна быть знаковая, можно использовать ключевое слово `signed`, например `signed char`. Это даст гарантию, что при конверсии в `int` произойдет расширение знака даже на системах, которые по умолчанию это не делают.

Источники:

1. Stephen Kochan – Programming in C (4th Edition); chapter 13
2. Дополнительный код – https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BF%D0%BE%D0%BB%D0%BD%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4#%D0%A0%D0%B0%D1%81%D1%88%D0%B8%D1%80%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B7%D0%BD%D0%B0%D0%BA%D0%B0
3. C in a Nutshell. Type Conversions – https://www.oreilly.com/library/view/c-in-a/0596006977/ch04.html
4. Typedef for Function Pointers – https://riptutorial.com/c/example/31818/typedef-for-function-pointers
5. Type Conversion in C – https://www.geeksforgeeks.org/type-conversion-c/

Упражнения

1. *Define a type `FunctionPtr()` (using `typedef`) that represents a pointer to a function that returns an `int` and that takes no arguments.*

```
typedef int  (*FunctionPtr) (void);
```

2. *Write a function called `monthName()` that takes as its argument a value of type `enum month` and returns a pointer to a character string containing the name of the month.*

```
#include <stdio.h>

enum  month  { January = 1, February, March, April, May, June,
               July, August, September, October, November, December };


char *monthName(enum month aMonth)
{
     char *months[] = {
          "January",
          "February",
          "March",
          "April",
          "May",
          "June",
          "July",
          "August",
          "September",
          "October",
          "November",
          "December",
          "Wrong month number"
     };

     if (aMonth < 1 && aMonth > 12)
          return(months[12]);
     else return(months[aMonth - 1]);
}

int main (void)
{
     enum  month  aMonth;

     printf ("Enter month number: ");
     scanf ("%u", &aMonth);
     printf(monthName(aMonth));

     return 0;
}
```

3. *Given the following variable declarations:*

```
float     f = 1.00;
short int i = 100;
long int  l = 500L;
double    d = 15.00;
```

and the seven steps outlined in this chapter for conversion of operands in expressions, determine the type and value of the following expressions:

`f + i = 101.00, float`

```
l / d = 33.333, double
i / l + f = 0 + 1.00 = 1.00, long int -> float
l * i = 50000, long int
f / 2 = 0.5, float
i / (d + f)   = 6.25, double
l / (i * 2.0) = 2.5, double
l + i / (double) l = 500.2, double
```