Краткая памятка по языку С

Оглавление

Занятие 1

Типы данных языка С

Переменные

Программа оперирует информацией, представленной в виде различных объектов и величин. Переменная – это символическое обозначение величины в программе. Как ясно из названия, значение переменной (или величина, которую она обозначает) во время выполнения программы может изменяться.

С точки зрения архитектуры компьютера, переменная – это символическое обозначение ячейки оперативной памяти программы, в которой хранятся данные. Содержимое этой ячейки – это текущее значение переменной.

В языке С прежде чем использовать переменную, ее необходимо объявить. Объявить переменную с именем x можно так:

```
int x;
```

В объявлении первым стоит название типа переменной int (целое число), а затем идентификатор x – имя переменной. У переменной x есть тип – в данном случае целое число. Тип переменной определяет, какие возможные значения эта переменная может принимать и какие операции можно выполнять над данной переменной. Тип переменной изменить нельзя, т.е. пока переменная x существует, она всегда будет целого типа.

Язык С – это строго типизированный язык. Любая величина, используемая в программе, принадлежит к какому-либо типу. При любом использовании переменных в программе проверяется, применимо ли выражение или операция к типу переменной. Довольно часто смысл выражения зависит от типа участвующих в нем переменных.

Например, если мы запишем x+y, где x – объявленная выше переменная, то переменная y должна быть одного из числовых типов.

Соответствие типов проверяется во время компиляции программы. Если компилятор обнаруживает несоответствие типа переменной и ее использования, он выдаст ошибку (или предупреждение). Однако во время выполнения программы типы не проверяются. Такой подход, с одной стороны, позволяет обнаружить и исправить большое количество ошибок на стадии компиляции, а, с другой стороны, не замедляет выполнения программы.

Переменной можно присвоить какое-либо значение с помощью операции присваивания. Присвоить – это значит установить текущее значение переменной. По-другому можно объяснить, что операция присваивания запоминает новое значение в ячейке памяти, которая обозначена переменной.

```
int x;          // объявить целую переменную x
int y;          // объявить целую переменную y
x = 0;          // присвоить x значение 0
y = x + 1;      // присвоить y значение x + 1,
                // т.е. 1
x = 1;          // присвоить x значение 1
y = x + 1;      // присвоить y значение x + 1,
                // теперь уже 2
```

Типы переменных

Название	**Обозначение**	**Диапазон значений**
Байт	char	от -128 до +127
без знака	unsigned char	от 0 до 255
Короткое целое число	short	от -32768 до +32767
Короткое целое число без знака	unsigned short	от 0 до 65535
Целое число	int	от – 2147483648 до + 2147483647
Целое число без знака	unsigned int	от 0 до 4294967295
Длинное целое число	long	от – 2147483648 до + 2147483647
Длинное целое число без знака	unsigned long	от 0 до 4294967295
Вещественное	float	от ±3.4e-38 до ±3.4e+38
Вещественное двойной точности	double	от ±1.7e-308 до ±1.7e+308
Логическое значение	bool	true(истина) или false (ложь)

Для целых чисел определены стандартные арифметические операции сложения (+), вычитания (-), умножения (*), деления (/); нахождение остатка от деления (%), изменение знака (-). Результатом этих операций также является целое число. При делении остаток отбрасывается.

Последний вопрос, который мы рассмотрим в отношении целых чисел, – это преобразование типов. В языке С допустимо смешивать в выражении различные целые типы. Например, вполне допустимо записать x + y, где x типа short , а y – типа long . При выполнении операции сложения величина переменной x преобразуется к типу long . Такое преобразование можно произвести всегда, и оно безопасно, т.е. мы не теряем никаких значащих цифр. Общее правило преобразования целых типов состоит в том, что более короткий тип при вычислениях преобразуется в более длинный. Только при выполнении присваивания длинный тип может преобразовываться в более короткий. Например:

```
short x;
long y = 15;
x = y;   // преобразование длинного типа в более короткий
```

Такое преобразование не всегда безопасно, поскольку могут потеряться значащие цифры. Обычно компиляторы, встречая такое преобразование, выдают предупреждение или сообщение об ошибке.

Вещественные числа записываются либо в виде десятичных дробей, например 1.3, 3.1415, 0.0005, либо в виде мантиссы и экспоненты: 1.2E0, 0.12e1. Отметим, что обе предыдущие записи изображают одно и то же число 1.2.

Для вещественных чисел определены все стандартные арифметические операции сложения (+), вычитания (-), умножения (*), деления (/) и изменения знака (-). В отличие от целых чисел, операция нахождения остатка от деления для вещественных чисел не определена.

Константы

В программе можно явно записать величину – число, символ и т.п. Например, мы можем записать выражение x + 4 – сложить текущее значение переменной x и число 4. В зависимости от того, при каких условиях мы будем выполнять программу, значение переменной x может быть различным. Однако целое число четыре всегда останется прежним. Это неизменяемая величина или константа.

Таким образом, явная запись значения в программе – это константа.

Далеко не всегда удобно записывать константы в тексте программы явно. Гораздо чаще используются символические константы. Например, если мы запишем

```
const int BITS_IN_WORD = 32;
```

то затем имя BITS_IN_WORD можно будет использовать вместо целого числа 32.

Преимущества такого подхода очевидны. Во-первых, имя BITS_IN_WORD (битов в машинном слове) дает хорошую подсказку, для чего используется данное число. Без комментариев понятно, что выражение

```
b / BITS_IN_WORD
```

(значение b разделить на число 32) вычисляет количество машинных слов, необходимых для хранения b битов информации. Во-вторых, если по каким-либо причинам нам надо изменить эту константу, потребуется изменить только одно место в программе – определение константы, оставив все случаи ее использования как есть. (Например, мы переносим программу на компьютер с другой длиной машинного слова.)

Массивы

Массивы - это группа элементов одинакового типа (double, float, int и т.п.). Из объявления массива компилятор должен получить информацию о типе элементов массива и их количестве. Объявление массива имеет формат:

```
тип имя[размер];
```

Например:

```
int my_array[10];       // Массив из 10 элементов
```

Первый элемент массива имеет индекс 0.

В языке С формально определены только одномерные массивы, но поскольку элементом массива может быть массив, можно определить и многомерные массивы.

Пример:

```
int array_2d[2][3];      // Массив, представляющий собой матрицу 2x3
```

При обработке массивов, как правило, к их элементам обращаются в цикле, используя в качестве номера элемента переменную цикла.

Пример обработки одномерного массива:

```
for(i = 0; i < 10; i++)
{
        my_array[i]++;
}
```

Пример обработки двумерного массива:

```
for(i = 0; i < 2; i++)
{
        for(j = 0; j < 3; j++)
        {
                array_2d[i][j]++;
        }
}
```

Операции языка С

Арифметические операции

+	сложение
-	вычитание
*	умножение
/	деление

Операции сложения, вычитания, умножения и деления целых и вещественных чисел. Результат операции – число, по типу соответствующее большему по разрядности операнду. Например, сложение чисел типа short и long в результате дает число типа long.

%	остаток

Операция нахождения остатка от деления одного целого числа на другое. Тип результата – целое число.

++	увеличить на единицу
--	уменьшить на единицу

Эти операции иногда называют "автоувеличением" (инкремент) и "автоуменьшением" (декремент). Они увеличивают (или, соответственно, уменьшают) операнд на единицу. Разница между постфиксной (знак операции записывается после операнда, например x++) и префиксной (знак операции записывается перед операндом, например --y) операциями заключается в том, что в первом случае результатом является значение операнда до изменения на единицу, а во втором случае – после изменения на единицу.

Операции сравнения

==	равно
!=	не равно
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно

Сравнивать можно операнды любого типа, но либо они должны быть оба одного и того же встроенного типа (сравнение на равенство и неравенство работает для двух величин любого типа), либо между ними должна быть определена соответствующая операция сравнения. Результат – логическое значение true или false.

Логические операции

&&	логическое И
||	логическое ИЛИ
!	логическое НЕ

Логические операции конъюнкции, дизъюнкции и отрицания. В качестве операндов

выступают логические значения, результат – тоже логическое значение true или false.

Операции присваивания

= присваивание

Присвоить значение правого операнда левому. Результат операции присваивания – это значение правого операнда.

+=, -=, *=, /=, %=, |=, &=, ^=, <<=, >>=

выполнить операцию и присвоить

Выполнить соответствующую операцию с левым операндом и правым операндом и присвоить результат левому операнду. Типы операндов должны быть такими, что, во-первых, для них должна быть определена соответствующая арифметическая операция, а во-вторых, результат может быть присвоен левому операнду.

Порядок вычисления выражений

У каждой операции имеется приоритет. Если в выражении несколько операций, то первой будет выполнена операция с более высоким приоритетом. Если же операции одного и того же приоритета, они выполняются слева направо.

Приоритет операций достаточно интуитивен. В порядке убывания:

* (умножение), / (деление), % (остаток)

+ (сложение), – (вычитание)

< <= > >= (сравнения на больше или меньше)

== != (равно, неравно)

&& (логическое И)

|| (логическое ИЛИ)

Для того чтобы изменить последовательность вычисления выражений, можно воспользоваться круглыми скобками. Скобки могут быть вложенными, соответственно, самые внутренние выполняются первыми.

Условные операторы

Условные операторы позволяют выбрать один из вариантов выполнения действий в зависимости от каких-либо условий. Условие – это логическое выражение, т.е. выражение, результатом которого является логическое значение true (истина) или false (ложь).

Оператор if выбирает один из двух вариантов последовательности вычислений.

```
if   (условие)
{
        оператор1 ;
}
else
{
        оператор2 ;
}
```

Если условие истинно, выполняется оператор1, если ложно, то выполняется оператор2.

Конструкция else необязательна. Можно записать:

```
if   (x < 0)
{
        x = -x;
}
abs = x;
```

В данном примере оператор x = -x; выполняется только в том случае, если значение переменной x было отрицательным. Присваивание переменной abs выполняется в любом случае. Таким образом, приведенный фрагмент программы изменит значение переменной x на его абсолютное значение и присвоит переменной abs новое значение x.

Условный оператор можно расширить для проверки нескольких условий:

```
if (x  < 0)
{
        printf("Отрицательная величина\n");
}
else if   (x > 0)
{
        printf("Положительная величина\n");
}
else
{
        printf("Ноль\n");
}
```

Конструкций else if может быть несколько.

Хотя любые комбинации условий можно выразить с помощью оператора if, довольно часто запись становится неудобной и запутанной. Оператор выбора switch используется, когда для каждого из нескольких возможных значений выражения нужно выполнить определенные действия. Например, предположим, что в переменной code хранится целое число от 0 до 2, и нам нужно выполнить различные действия в зависимости от ее значения:

```
switch (code)
{
```

```
case 0:
        printf("код ноль\n");
        break;
case 1 :
        printf("код один\n");
        break;
case 2:
        printf("код два\n");
        break;
default:
        printf("Необрабатываемое значение\n");
}
```

В зависимости от значения code управление передается на одну из меток case. Выполнение оператора заканчивается по достижении либо оператора break, либо конца оператора switch. Таким образом, если code равно 1, выводится "код один". Если бы после этого не стоял оператор break, то управление "провалилось" бы дальше, была бы выведена фраза "код два" и т.д.

Если значение переключателя не совпадает ни с одним из значений меток case, то выполняются операторы, записанные после метки default. Метка default может быть опущена.

Очевидно, что приведенный пример можно переписать с помощью оператора if.

Пожалуй, запись с помощью оператора переключения switch более наглядна. Особенно часто переключатель используется, когда значение выражения имеет тип набора.

Операторы цикла

Предположим, нам нужно вычислить сумму всех целых чисел от 0 до 100. Для этого воспользуемся оператором цикла for:

```
int sum = 0;
int i;
for (i = 1; i <= 100; i = i + 1)          // Заголовок цикла
{
        sum = sum + i;                    // Тело цикла
}
```

Оператор цикла состоит из заголовка цикла и тела цикла. Тело цикла – это оператор, который будет повторно выполняться (в данном случае – увеличение значения переменной sum на величину переменной i). Заголовок – это ключевое слово for, после которого в круглых скобках записаны три выражения, разделенные точкой с запятой. Первое выражение вычисляется один раз до начала выполнения цикла. Второе – это условие цикла. Тело цикла будет повторяться до тех пор, пока условие цикла истинно. Третье выражение вычисляется после каждого повторения тела цикла.

Оператор for реализует фундаментальный принцип вычислений в программировании – итерацию. Тело цикла повторяется для разных, в данном случае последовательных, значений переменной i. Повторение иногда называется итерацией.

Еще одним вспомогательным оператором при выполнении циклов служит оператор продолжения continue. Оператор continue заставляет пропустить остаток тела цикла и перейти к следующей итерации (повторению). Например, если мы хотим найти сумму всех целых чисел от 0 до 100, которые не делятся на 7, можно записать это так:

```
int sum = 0;
for (int i = 1; i <= 100; i = i+1)
{
        if ( i % 7 == 0)
        {
                continue;
        }
        sum = sum + i;
}
```

Другой формой оператора цикла является оператор while. Его форма следующая:

```
while (условие)
{
        оператор;
}
```

Условие – как и в условном операторе if – это выражение, которое принимает логическое значение "истина" или "ложь". Выполнение оператора повторяется до тех пор, пока значением условия является true (истина). Условие вычисляется заново перед каждой итерацией.

Подсчитать, сколько десятичных цифр нужно для записи целого положительного числа N, можно с помощью следующего фрагмента:

```
int digits = 0;
while (N > 1)
{
        digits = digits + 1;
        N = N / 10;
}
```

Третьей формой оператора цикла является цикл do while. Он имеет форму:

```
do {
        операторы;
} while ( условие);
```

Отличие от предыдущей формы цикла while заключается в том, что условие проверяется после выполнения тела цикла. Предположим, требуется прочитать символы с терминала до тех пор, пока не будет введен символ "звездочка".

```
char ch;
do {
        ch = getch();   // функция getch возвращает
                        // символ, введёный с
                        // клавиатуры
} while (ch != '*');
```

В операторах while и do также можно использовать операторы break и continue.

Как легко заметить, операторы цикла взаимозаменяемы. Разные формы нужны для удобства и наглядности записи.

Занятие 2

Функции

Объявление и вызов функций

Функция вызывается при вычислении выражений. При вызове ей передаются определенные аргументы, функция выполняет необходимые действия и возвращает результат.

Программа на языке С состоит, по крайней мере, из одной функции – функции main. С нее всегда начинается выполнение программы. Встретив имя функции в выражении, программа вызовет эту функцию, т.е. передаст управление на ее начало и начнет выполнять операторы. Достигнув конца функции или оператора return – выхода из функции, управление вернется в ту точку, откуда функция была вызвана, подставив вместо нее вычисленный результат.

Прежде всего, функцию необходимо объявить. Объявление функции, аналогично объявлению переменной, определяет имя функции и ее тип – типы и количество ее аргументов и тип возвращаемого значения.

Функция sqrt с одним аргументом – вещественным числом двойной точности, возвращает результат типа double:

```
double sqrt(double x);
```

Функция sum от трех целых аргументов , возвращает целое число:

```
int sum(int a, int b, int c);
```

Объявление функции называют иногда прототипом функции. После того, как функция объявлена, ее можно использовать в выражениях:

```
double x = sqrt(3) + 1;

sum(k, l, m) / 15;
```

Определение функции описывает, как она работает, т.е. какие действия надо выполнить, чтобы получить искомый результат. Для функции sum, объявленной выше, определение может выглядеть следующим образом:

```
int sum(int a, int b, int c)
{
        int result;
        result = a + b + c;
        return result;
}
```

Первая строка – это заголовок функции, он совпадает с объявлением функции, за исключением того, что объявление заканчивается точкой с запятой, а заголовок - нет. Далее в фигурных скобках заключено тело функции – действия, которые данная функция выполняет.

Аргументы a, b и c называются формальными параметрами. Это переменные, которые определены в теле функции (т.е. к ним можно обращаться только внутри фигурных скобок). При написании определения функции программа не знает их значения. При вызове функции вместо них подставляются фактические параметры – значения, с которыми функция вызывается. Выше, в примере вызова функции sum, фактическими параметрами (или фактическими аргументами) являлись значения переменных k, l и m.

Формальные параметры принимают значения фактических аргументов, заданных при вызове, и функция выполняется.

Первое, что мы делаем в теле функции — объявляем внутреннюю переменную result типа целое. Переменные, объявленные в теле функции, также называют локальными. Это связано с тем, что переменная result существует только во время выполнения тела функции sum. После завершения выполнения функции она уничтожается – ее имя становится неизвестным, и память, занимаемая этой переменной, освобождается.

Вторая строка определения тела функции – вычисление результата. Сумма всех аргументов присваивается переменной result. Отметим, что до присваивания значение result было неопределенным (то есть значение переменной было неким произвольным числом, которое нельзя определить заранее).

Последняя строчка функции возвращает в качестве результата вычисленное значение. Оператор return завершает выполнение функции и возвращает выражение, записанное после ключевого слова return, в качестве выходного значения.

В следующем примере программы в результате работы будет распечатано значение 10.

```
#include <stdio.h>

int sum(int a, int b, int c);      // Объявление функции

int main ()                        // Точка входа в программу

{

        int k = 2;
        int l = 3;
        int m = 5;
        int s = sum(k, l, m);    // Вызов функции
        printf("Результат: %d\n", s);
return 0;
}

int sum(int a, int b, int c)       // Определение функции
{
        int result;
        result = a + b + c;
        return result;
}
```

Рекурсия

Определения функций не могут быть вложенными, т.е. нельзя внутри тела одной функции определить тело другой. Разумеется, можно вызвать одну функцию из другой. В том числе функция может вызвать сама себя.

Рассмотрим функцию вычисления факториала целого числа. Ее можно реализовать двумя способами. Первый способ использует итерацию:

```
int fact(int n)
{
	int result = 1;
	for (int i = 1; i <= n; i++)
	{
		result = result * i;
	}
	return result;
}
```

Второй способ:

```
int fact(int n)
{
	if (n == 1)	// факториал 1 равен 1
	{
		return 1;
	}
	else	// факториал числа n равен факториалу n-1, умноженному на n
	{
		return n * fact(n -1);
	}
}
```

Функция fact вызывает сама себя с модифицированными аргументами. Такой способ вычислений называется рекурсией. Рекурсия – это очень мощный метод вычислений. Значительная часть математических функций определяется в рекурсивных терминах. В программировании алгоритмы обработки сложных структур данных также часто бывают рекурсивными.

Довольно часто рекурсия и итерация взаимозаменяемы (как в примере с факториалом). Выбор между ними может быть обусловлен разными факторами. Чаще рекурсия более наглядна и легче реализуется. Однако, в большинстве случаев итерация более эффективна.