Лекция 1
Структура программы на Си

Состав языка

В тексте на любом естественном языке можно выделить четыре основных элемента: символы, слова, словосочетания и предложения. Подобные элементы содержит и алгоритмический язык:

- *Алфавит* языка, или его *символы* — это основные неделимые знаки, с помощью которых пишутся все тексты на языке.
- *Лексема*— минимальная единица языка, имеющая самостоятельный смысл.
- *Выражение* задает правило вычисления некоторого значения.
- *Оператор* задает законченное описание некоторого действия.

Для описания сложного действия требуется последовательность операторов. Операторы могут быть объединены в ***составной оператор, или блок*** (блоком в языке C++ считается последовательность операторов, заключенная в фигурные скобки **{ }**). В этом случае они рассматриваются как один оператор.

Каждый элемент языка определяется ***синтаксисом* и *семантикой***. Синтаксические определения устанавливают правила построения элементов языка, а семантика определяет их смысл и правила использования.

Процесс выполнения программы

Для того чтобы выполнить программу, требуется перевести ее на язык, понятный процессору — в машинные коды. Этот процесс состоит из нескольких этапов:

- Сначала программа передается ***препроцессору***, который подключает текстовые файлы, содержащие описание используемых в программе элементов.
- Получившийся полный текст программы поступает на вход ***компилятора***, который выделяет лексемы, а затем на основе грамматики языка распознает выражения и операторы, построенные из этих лексем. При этом компилятор выявляет синтаксические ошибки и в случае их отсутствия строит объектный модуль.
- ***Компоновщик***, или **редактор связей**, формирует исполняемый модуль программы, подключая к объектному модулю другие объектные модули. Если программа состоит из нескольких исходных файлов, они компилируются по отдельности и объединяются на этапе компоновки. Исполняемый модуль имеет расширение .exe и может быть запущен на выполнение.

Алфавит языка C++

В алфавит языка Си входят:

- прописные и строчные буквы латинского алфавита (A,B,...,Z, a, b,..., z) и и знак подчеркивания;
- цифры: 0,1,2,3,4,5,6,7,8,9
- специальные знаки: " , {} | [ ] () * + - / % \ ; ' . :?< = >_ ! & # -
- неотображаемые символы ("обобщенные пробельные символы"), используемые для отделения лексем друг от друга (например, пробел, табуляция, переход на новую строку).

Из символов алфавита формируются *лексемы* языка:

- идентификаторы;
- ключевые (зарезервированные) слова;
- знаки операций;
- константы;
- разделители (скобки, точка, запятая, пробельные символы).

Границы лексем определяются другими лексемами, такими, как

разделители или знаки операций.

Идентификаторы

Идентификатор — это имя программного объекта. В идентификаторе могут использоваться латинские буквы, цифры и знак подчеркивания. **Прописные и строчные буквы различаются.** Первым символом идентификатора может быть буква или знак подчеркивания. При этом:

- идентификатор не должен совпадать с ключевыми словами и именами используемых стандартных объектов языка;
- не рекомендуется начинать идентификаторы с символа подчеркивания;
- Идентификаторы могут иметь любую длину, но компилятор учитывает не более 31-го символа от начала идентификатора;

Примеры идентификаторов:

КОМ_16, size88, _MIN, TIME, time

Ключевые слова

Ключевые слова — это зарезервированные идентификаторы, которые имеют специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. Список ключевых слов C++ приведен в таблице:

INT	EXTERN	ELSE
CHAR	REGISTER	FOR
FLOAT	TYPEDEF	DO
DOUBLE	STATIC	WHILE
STRUCT	GOTO	SWITCH
UNION	RETURN	CASE
LONG	SIZEOF	DEFAULT
SHORT	BREAK	ENTRY
UNSIGNED	CONTINUE	AND
AUTO	IF	STRUCT
CONST	VOID	TYPEDEF
TRUE	NEW	BOOL

Знаки операций

Знак операции — это один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются. Операции делятся на унарные, бинарные и тернарную по количеству участвующих в них операндов.

Простые типы данных

***Тип данных* определяет:**

- *внутреннее представление* данных в памяти компьютера;
- *множество значений*, которые могут принимать величины этого типа;
- *операции и функции*, которые можно применять к величинам этого типа.

В языке C++ определено шесть стандартных простых типов данных для представления целых, вещественных, символьных и логических величин.

Типы данных

К *спецификаторам типов* относятся:

char - символьный;

double - вещественный двойной точности с плавающей точкой;

float - вещественный с плавающей точкой;

int - целый;

long - целый увеличенной длины (длинное целое);

short - целый уменьшенной длины (короткое целое);

signed - знаковый, т.е. целое со знаком (старший бит считается знаковым);

unsigned - беззнаковый, т.е. целое без знака (старший бит не считается знаковым);

void - отсутствие значения;

bool - логический (могут принимать только значения **true** и **false**. Внутренняя форма представления значения **false** — 0 (нуль). Любое другое значение интерпретируется как **true**.)

Для описания констант используют служебное слово const перед описанием типа. Например, **const float g=9.8;**

Структура Си-программы

Программа на языке С++ представляет собой одну или несколько функций. Одна из функций должна называться **main () .** Именно с этой функции начинает выполняться программа, и из этой функции, по мере необходимости, вызываются другие функции.

Простейшее определение функции имеет следующий формат:

тип_возвращаемого_значения имя ([параметры])

{

операторы, составляющие тело функции

}

Как правило, функция используется для вычисления какого-либо значения, поэтому перед именем функции указывается его тип. Если функция не должна возвращать значение, указывается тип void.

При этом:

- Тело функции заключается в фигурные скобки.
- Функции не могут быть вложенными.
- Каждый оператор заканчивается точкой с запятой (кроме составного оператора).

Структура Си-программы

Программа может состоять из нескольких *модулей* (исходных файлов) и, как правило, содержит *директивы препроцессора*. Пример структуры программы, содержащей функцииmain, f1 и f2:

```
директивы препроцессора
описания
int main()
 {
операторы главной функции
 }
 int f1() {
            операторы функции f1
           }
int f2()    {
           операторы функции f2
             }
```

```
/* Пример простой программы*/
#include <stdio.h>
int main()
{
printf(“Hello World!”);
return 0;
}
```

функции ввода/вывода в стиле Си

Основные *функции ввода/вывода* в стиле С:

int scanf (const char* format...) // ввод

int printf(const char* format...) // вывод

Они выполняют форматированный ввод и вывод произвольного количества величин в соответствии со строкой формата format. Строка формата содержит символы, которые при выводе копируются в поток (на экран) или запрашиваются из потока (с клавиатуры) при вводе, и *спецификации преобразования*, начинающиеся со знака %, которые при вводе и выводе заменяются конкретными величинами.

```
#include <stdio.h>
 int main() {
 int i;
printf("Введите целое число\n");
scanf("%d", &i);
printf("Вы ввели число %d, спасибо!", i);
return 0; }
```

Первая строка этой программы — директива препроцессора, по которой в текст программы вставляется заголовочный файл **<stdio.h>**, содержащий описание использованных в программе функций ввода/вывод. Все директивы препроцессора начинаются со знака #.

Третья строка — описание переменной целого типа с именем **i** .

Функция **printf** в четвертой строке выводит приглашение Введите целое число и переходит на новую строку в соответствии с управляющей последовательностью \n.

Функция **scanf** заносит введенное с клавиатуры целое число в переменную i, а следующий оператор выводит на экран указанную в нем строку, заменив спецификацию преобразования на значение этого числа.

функции ввода/вывода в стиле Си++

Та же программа с использованием *библиотеки классов C++* <iostream>:

```
#include <iostream>
using namespace std;
int main()
 {
 int i;
cout << "Введите целое число\n";
cin >> i;
cout << "Вы ввели число" << i << ", спасибо!";
    return 0;
}
```

Стандартные библиотеки

Язык Си имеет богатую поддержку в виде **более 450 библиотечных подпрограмм** - функций и макросов, которые вы можете вызывать из своих Си-программ для решения широкого спектра задач, включая ввод/вывод низкого и высокого уровня, работу со строками и файлами, распределение памяти, управление процессами, преобразование данных, математические вычисления и многое другое.

Директивы препроцессора

Препроцессором называется первая фаза компилятора. Инструкции препроцессора называются директивами. Они должны начинаться с символа **#**, перед которым в строке могут находиться только пробельные символы.

Поиск файла, если не указан полный путь, ведется в стандартных каталогах включаемых файлов. Вместо угловых скобок могут использоваться кавычки (" ") — в этом случае поиск файла ведется в каталоге, содержащем исходный файл, а затем уже в стандартных каталогах. Заголовочные файлы обычно имеют расширение **.h** .

Директива препроцессора # include

Для подключения библиотек используется директива препроцессора

 # include [имя файла]

Директива #include <имя_файла> вставляет содержимое указанного файла в ту точку исходного файла, где она записана.

Например:

#include <stdio.h>

#include “mylib.h”

Директива препроцессора #define

Директива **#define** определяет подстановку в тексте программы. Она используется для определения *символических констант*.

Формат определения символической константы:

#define имя текст_подстановки /*Все вхождения имени заменяются на текст подстановки */

Примеры:

#define M 1000

#define Vasia “Василий Иванович”

Некоторые стандартные библиотеки

ALLOC.H Описание функций управления памятью (allocation, deallocation и др.)

BIOS.H Описание различных функций, используемых при обращении к подпрограммам BIOS (базовой системе ввода-вывода).

CONIO.H Описание различных функций, используемых в обращении к подпрограммам DOS ввода-вывода с клавиатуры.

GRAPHICS.H Содержит прототипы графических функций.

MATH.H Содержит описание прототипов математических функций

STDIO.H Определяет типы и макросы, необходимые для стандартного пакета ввода-вывода. Определяет так же стандартные потоки ввода-вывода stdin, stdout и описывает подпрограммы ввода/вывода.

STDLIB.H Описывает некоторые подпрограммы общего назначения: подпрограммы преобразования, поиска, сортировки и другие.

STRING.H Описывает несколько подпрограмм обработки строк и работы с памятью.

Функция форматного вывода PRINTF

Мы уже использовали наиболее употребительную функцию вывода в Си - подпрограмму printf. Ее целью является запись информации на экран.

Ее формат как прост, так и гибок:

printf(<строка формата>, <объект>, <объект>, ...);

Функция форматного ввода SCANF

Для интерактивного режима ввода, вероятно, можно использовать в большинстве случаев функцию scanf. scanf это функция ввода, по смыслу эквивалентная printf; ее формат выглядит так:

scanf(<строка формата>,<адрес>,<адрес>,...)

Однако scanf имеет одно очень важное отличие: объекты, следующие за строкой формата, должны быть адресами, а не значениями. Например, в программе содержится следующий вызов:

```
scanf("%d %d", &a, &b);
```

Этот вызов сообщает программе, что она должна ожидать от вас ввода двух десятичных (целых) чисел, разделенных пробелом; первое будет присвоено a, а второе b. Заметим, что здесь используется операция адреса (&) для передачи адресов a и b функции scanf.

Строка формата

Строка формата - это строка, которая начинается и заканчивается двойными кавычками ("как эта"); цель **printf** - запись этой строки на экран. Перед выводом **printf** заменяет все дополнительно перечисленные объекты в строке в соответствии со спецификациями формата, указанными в самой строке. Пример оператора **printf** :

printf(“ d= %f \n",d);

Строка формата

%f в строке формата - это спецификация формата. Все спецификации формата начинаются с символа процента (%) и (обычно) сопровождаются одной буквой, обозначающей тип данных и способ их преобразования.

%f, используемое в спецификации, говорит о том, что ожидается некоторое вещественное число. Вот несколько других обще- используемых спецификаций формата:

- %u целое число без знака
- %ld длинное целое число
- %p значение указателя
- %d целое число
- %e число с плавающей точкой в экспоненциальной форме
- %c символ
- %s строка
- %x или %X целое в шестнадцатиричном формате.

Строка формата

Вы можете задать ширину поля, помещая ее между % и буквой, например, десятичное поле шириной 4 задается, как %4d. Значение будет напечатано сдвинутым вправо (впереди пробелы), так что общая ширина поля равна 4.

Или например %6.2f будет обозначать что отведено 6 позиций под вещественное число, причем 2 – под дробную часть.

Если нужно напечатать знак %, то вставьте %%

Строка формата

\n в строке не является спецификацией формата, а употребляется (по историческим мотивам) как управляющая (escape) последовательность, и представляет специальный символ, вставляемый в строку. В этом случае \n вставляет символ в начале новой строки, поэтому после вывода строки курсор передвинется к началу новой строки.

- \f (перевод формата или очистка экрана)
- \t (табуляция)
- \b (забой <-)
- \xhhh (вставка символа с кодом ASCII hhh, где hhh содержит от 1 до 3 16-ричных цифр)

Если вам нужно напечатать обратную косую черту, то вставьте \\.

Пример Си-программы

```
#include <stdio.h>
#include <math.h>
main()
{
float x1,y1,x2,y2;
printf("Введите два числа:  ");
scanf("%f %f %f %f”,&x1,&y1,&x2,&y2);
float d = sqrt(pow((x2-x1),2)+pow((y2-y1),2));
printf(“d= %f \n", d);
}
```

Другие функции вывода PUTS, PUTCHAR

Имеются две другие функции вывода, которые могут вас заинтересовать: puts и putchar. Функция puts выводит строку на экран и завершает вывод символом новой строки.

Рассмотрим пример программы :

```
#include <stdio.h>
 main ()
 {
   puts("Привет, студент ВКИ НГУ");
 }
```

Заметим, что в конце строки опущен \n; это не нужно, так как puts сама добавляет этот символ.

Наоборот, функция putchar записывает единственный символ на экран и не добавляет \n. Оператор putchar(ch) эквивалентен printf("%c",ch).

Функции GETS, GETCH

```
main ()
        {
          char name [60];
          printf("Как вас зовут: ");
          scanf (“%s”,name);
          printf ("Привет, %s\n", name);
         }
```

Если ввести фамилию и имя, то выведется только фамилия. Потому, что введенный вами после имени пробел сигнализирует scanf о конце вводимой строки.

Решить эту проблему можно, используя функцию gets.

```
 main ()
        {
          char name [60];
          printf("Как вас зовут: ");
          gets (name);
          printf ("Привет, %s\n", name);
         }
```

Функции GETS, GETCH

```
# include <stdio.h>
# include <conio.h>
main ()
        {
          clrscr();
          char name [60];
          printf("Как вас зовут: ");
          gets (name);
          printf ("Привет, %s\n", name);
          getch();
        }
```

Здесь функция getch ожидает ввода любого символа.

Функция getch читает единственный символ с клавиатуры, не выдавая его на экран (в отличии от scanf и gets). Заметим, что у нее нет параметра. Если же присвоить функцию символьной переменной, то она получит значение нажатого символа.

Например : c=getch();

Операция присваивания

Самой общей операцией является присваивание, например, p=a/b или ch = getch(). В Си присваивание обозначается одним знаком равенства (=); при этом значение справа от знака равенства присваивается переменной слева.

Можно применять также последовательные присваивания, например: sum=a= b. В таких случаях присваивание производится справа налево, то есть b будет присвоено a, которая в свою очередь будет присвоена sum, так что все три переменных получат одно и то же

Арифметические операции. Инкремент. Декремент.

Си поддерживает обычный набор арифметических операций:

- **умножение (*)**
- **деление (/)**
- **определение остатка от деления (%)**
- **сложение (+)**
- **вычитание (-)**

Для увеличения значений переменной на единицу используют инкремент (**++**), для уменьшения на единицу декремент (**--**).

Так :

x++; // увеличивает x на единицу

y--; // уменьшает y на единицу

Примеры постфиксных и префиксных операций:

a=2;	**a=2;**
x=18-2*a++;	**x=18-2*(++a);**
Получим :	
a=3	**a=3**
x= 14	**x=12**

Комбинированные операции

Ниже приводятся некоторые примеры выражений и способы их сокращения:

a = a + b; сокращается до a += b;

a = a - b; сокращается до a -= b;

a = a * b; сокращается до a *= b;

a = a / b; сокращается до a /= b;

a = a % b; сокращается до a %= b;

Пример 1

Вычислите X^3 и X^{10} , используя четыре операции умножения для заданного целого значения X.

```
#include <stdio.h>
int x,x2,x3,x5,x10;
    main()  {
        printf("\n Программа расчета X^3 и X^10\n");
        puts("Введите значение X");
        scanf("%i",&x);
       x2=x*x;
       x3=x*x2;
       x5=x2*x3;
       x10=x5*x5;
        printf("%i в 3-ей степени  = %i \n",x,x3);
        printf("%i в 10-ой степени = %i",x,x10);
     }
```

Пример 2

Дан угол в радианах. Вычислить синус и косинус угла.

```
#include <stdio.h>
#include <math.h>
float Angle,Result1,Result2;
main()
   {
    printf("\n Программа вычисления sin и cos угла\n");
    puts("Задайте значение угла в радианах");
    scanf("%f",&Angle);
    Result1=sin(Angle);
    Result2=cos(Angle);
    printf("Синус угла равен %f \n",Result1);
    printf("Косинус угла равен %f",Result2);
   }
```