ТЕМА 1
1.1. СТРУКТУРА ЯЗЫКА ПРОГРАММИРОВАНИЯ СИ

Происхождение и достоинства языка

Язык программирования ***Си*** был разработан и реализован в 1972 году сотрудником фирмы AT&T Bell Laboratories Денисом Ритчи *во время создания операционной системы UNIX (ОС UNIX).* Он планировался для замены ассемблера, чтобы иметь возможность создавать такие же эффективные и короткие программы, но не зависеть от конкретного процессора, и во многом похож на язык программирования Паскаль и имеет дополнительные возможности для работы с памятью.

На языке ***Си*** написаны: *операционная система* **Unix**, *компиляторы и интерпретаторы* языков Фортран, Паскаль и Бейсик, программы, которые, используются для решения физических и технических проблем, *компьютерной графики* и даже производства *мультипликационных фильмов*.

Программа, которая написанная на языке ***Си*** является:

- ***надежной и читаемой,*** т.к. структура языка позволяет использовать нисходящее проектирование, структурное программирование и пошаговую разработку модулей;
- ***эффективной,*** т.к. наилучшим образом используются возможности современных ЭВМ и отличаются компактностью и быстротой исполнения.
- **переносимой (или мобильной),** т.к. программа, написанная на языке Си для одной вычислительной системы, может быть перенесена с небольшими изменениями или вообще без них, на другую.

Программный модуль

С помощью языка программирования создается текст, описывающий ранее составленный алгоритм, который называется – **исходным модулем** (в ***Си*** – расширение *.cpp).

Для получения рабочей программы, необходимо этот текст перевести в последовательность команд для процессора, что выполняется при помощи специальных программ, которые называются трансляторами.

Трансляторы бывают *двух видов*: **компиляторы** и **интерпретаторы**.

Компилятор транслирует текст исходного модуля в машинный код за один непрерывный процесс.

Интерпретатор выполняет исходный модуль программы в режиме оператор за оператором, по ходу работы, переводя каждый оператор, написанный на высоком уровне языка программирования, на машинный язык.

Консольное приложение – это приложение, которое с точки зрения программиста является программой DOS, но может использовать всю доступную оперативную память. Этот тип приложения запускается в особом окне, которое называется «Окно MS-DOS».

Структура программы

Программа на языке **Си** имеет следующую структуру:

```
#директивы препроцессора
[глобальные переменные]
[прототипы пользовательских функций]
void main ( ) //функция, с которой начинается выполнение
              //каждой программы
{
   операторы:
        описания
        присваивания
        пустой оператор
        составной
```

выбора
циклов
перехода

Так же

вызов различных функций

}

[описание пользовательских функций]

Простой пример программы написанный на языке *Cи* представлен на рисунке 1.1

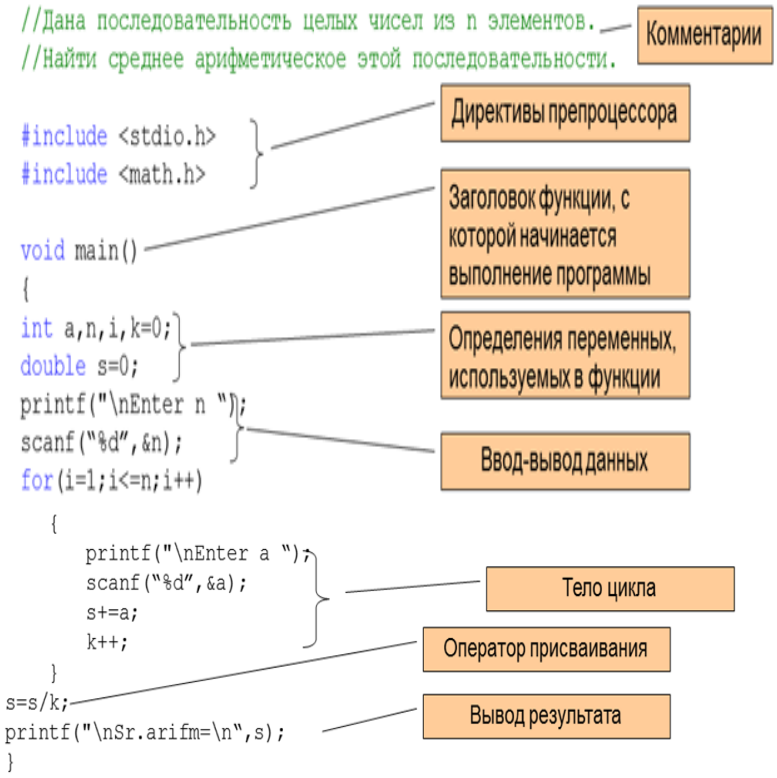

Рис. 1.1

Директивы препроцессора

Директивы препроцессора - управляют преобразованием текста программы до ее компиляции.

Задача препроцессора - преобразование текста программы до ее компиляции.

Правила препроцессорной обработки определяет программист с помощью директив препроцессора.

Директива начинается с **#**.

Примеры:

I. **#define** - указывает правила замены в тексте.

#define ZERO 0.0

Означает, что каждое использование в программе имени **ZERO** будет заменяться на **0.0**

II. **#include** < имя заголовочного файла>

Предназначена для включения в программу текста из каталога «Заголовочных файлов», поставляемых вместе со стандартными библиотеками.

Каждая библиотечная функция ***C*** имеет соответствующее описание в одном из заголовочных файлов. Список заголовочных файлов определен стандартом языка.

Использование директивы **include** не подключает соответствующую стандартную библиотеку, а только позволяют вставить в текст программы описания из указанного заголовочного файла.

Подключение кодов библиотеки осуществляется после компиляции. Хотя в заголовочных файлах содержатся все описания стандартных функций, в код программы включаются только те функции, которые используются в программе.

После выполнения препроцессорной обработки в тексте программы не остается ни одной препроцессорной директивы.

Основные элементы языка программирования

1. Символы – это основные знаки, с помощью которых пишутся все тексты программы.

Символы или а***лфавит языка*** включают в себя:

- Прописные и строчные латинские буквы и знак подчеркивания;
- Арабские цифры от 0 до 9;
- Специальные знаки: { } , | [ ] ( ) + - / % * . \ ' : ; & ? < > = ! # ^
- Пробельные символы (\t, \n пробел)

2. Лексемы образуются из символов, и имеет самостоятельный смысл.

2.1 Идентификаторы – это последовательность латинских букв, цифр и знака подчеркивания. Первым символом должна быть буква или знак подчеркивания (но не цифра). Пробелы в идентификаторах не допускаются. Регистр учитывается.
Примеры трех разных идентификаторов:

Prog1 PROG1 prog1

2.2 Ключевые (зарезервированные) слова – это слова, которые имеют специальное значение для компилятора. Их нельзя использовать в качестве идентификаторов. (true, false, int, float, switch … и.т.д.)

2.3 Знаки операций – это один или несколько символов, определяющих действие над операндами. (+ - * / % < > >= <= == != << >> ! & | && || * ++ и.т.д.)

2.4 Разделители – скобки, точка, запятая пробельные символы.

2.5 Литералы или константы – неизменяемые величины

Константа – это лексема, представляющая изображение фиксированного числового, строкового или символьного значения.

Константы делятся на:

- ***целые,*** которые могут быть десятичными, восьмеричными и шестнадцатеричными;
- ***вещественные***, которые могут иметь две формы представления с фиксированной точкой и с плавающей точкой;
- ***перечислимые,*** обычные целые константы, которым приписаны уникальные и удобные для использования обозначения с плавающей точкой;
- ***символьные,*** один или два символа, заключенные в апострофы. Последовательности, начинающиеся со знака \, называются *управляющими.*
- ***строковые,*** последовательность символов, заключенная в кавычки, которые могут содержать и управляющие символы.

3. Выражения образуются из лексем и символов и задают правила вычислений некоторого значения.

Если выражение формирует *целое* или *вещественное число*, то оно называется **арифметическим.**

a+b+64 // арифметическое выражение

Пара *арифметических выражений*, объединенная *операцией сравнения*, называется **отношением**.

```
c -4 > d*e        // отношение
```

4. ***Оператор*** образуется из символов, выражений и лексем, и задает оконченное описание некоторого действия.

Любое выражение, заканчивающееся точкой с запятой, рассматривается как *оператор.* Выполнение оператора - это вычисление данного выражения.

Примеры:

; - пустой оператор

i++; постфиксная форма унарного оператора

a+=2; краткая форма оператора присваивания

x=a+b; полная форма оператора присваивания

1.2. ВСТРОЕННЫЕ ТИПЫ ДАННЫХ ЯЗЫКА СИ

Основная цель любой программы состоит в обработки данных.

Данные – это область в оперативной памяти компьютера, где можно хранить некоторое значение, которое используется в программе.

Оперативная память представляет собой последовательность пронумерованных ячеек.

Программа – это последовательность команд (инструкций), которые помещаются в памяти и выполняются процессором в указанном порядке.

Базовые типы данных

Данные, значения которых можно изменить во время программы называются **переменными,** а не изменяемые данные называются **константами.**

Переменные представляют собой имена области памяти, в которых находятся данные определённых типов. С помощью переменных можно как записывать данные в область памяти, так и считывать данные из области памяти.

В зависимости ***от размещения и доступа***, различают **внутренние** (находятся в оперативной памяти) и **внешние** (на внешних устройствах) данные.

В зависимости **от вида данных** переменные могут быть, либо базовые, либо составные типы.

Базовыми называются такие данные, которые не могут быть расчленены на составные части, большие, чем биты, т.е. всегда известен размер и размещение в памяти данные базового типа, которые являются неделимой единицей.

Составными называются такие данные, составными частями которых являются другие данные – базовыми или в свою очередь составные.

Любые данные, т.е. константы, переменные, значения функции или выражения, характеризуются своими типами.

Типы данных определяют:

- **внутреннее представление** данных в вычислительных машинах;
- **диапазон** (или множество) значений, которые может принимать переменная;
- **операции**, которые применимы для переменных данного типа.

Базовые типы:

- *символ*
- *перечисления*
- *арифметические типы* (целочисленный и вещественный)
- *void*
- *логический*

Составные типы: (*массивы, структуры, объединения, классы*)

Целый тип int

Значениями этого типа являются *целые числа*. В 16-битных операционных системах под него отводится 2 байта, в 32-битных – 4 байта.

Если перед **int** стоит спецификатор **short**, то под число отводится 2 байта, а если спецификатор **long**, то 4 байта.

От количества отводимой под переменную памяти зависит множество допустимых значений, которые может принимать переменная.

Модификаторы **signed** и **unsigned** также влияют на множество допустимых значений, которые может принимать переменная:

unsigned short int – занимает 2 байта, диапазон 0 … 65 536;

unsigned long int – занимает 4 байта, диапазон 0 … +4 294 967 295.

Переменные типа **unsigned нельзя переполнять**, т.е. при записи значения, находящееся вне области допустимых значений, то произойдет потеря старших значений разрядов.

Вещественный тип (float и double)

Внутреннее представление вещественного числа состоит из 2 частей: **мантиссы** и **порядка.** ***Мантисса*** – это численное значение со знаком, ***порядок*** – это целое со знаком, определяющее значимость мантиссы.

Пример:

Число 54123, будет выглядеть следующим образом:$0.54123*10^4$

Число ***0.54123*** – это **мантисса**, а ***4*** –это **порядок** *в десятичной системе счисления*.

Символьный тип char

Значениями этого типа являются элементы конечного упорядоченного множества символов. Каждому символу ставится в соответствие число, которое называется кодом символа.

Под величину символьного типа отводится 1 байт. Тип **char** может использоваться со спецификаторами **signed** и **unsigned.**

Для кодировки используется код ASCII (American Standard Code for international interchange). Код ASCII Стандарт кодирования символов латинского алфавита, цифр и вспомогательных символов или действий в виде однобайтового двоичного кода (1 байт = 8 бит). Первоначально стандарт определял только 128 символов, используя 7 битов (от 0 до 127). Использование всех восьми битов позволяет кодировать еще 128 символов. Дополнительные символы могут быть любыми, им отводятся коды от 128 до 255. национальные алфавиты кодируются именно в этой части ASCII-кода. Символы с кодами от 0 до 31 относятся к служебными и имеют самостоятельное значение только в операторах ввода-вывода.

Логический тип bool

Тип **bool** называется логическим. Его величины могут принимать значения **true** (истина) и **false** (ложь). Внутренняя форма представления **false** – 0, любое другое значение интерпретируется как **true –** (любое не нулевое значение).

Тип void

К основным типам также относится тип **void**. Множество значений этого типа – пусто.
Невозможно создать переменную этого типа, но можно использовать указатель.

Декларация данных

Все данные, с которыми работает программа, необходимо декларировать, т.е. объявить компилятору об их присутствии. При этом возможны две формы декларации:

– *объявление*, т.е. описание, не приводящее к выделению памяти;

– *определение*, при котором под данные выделяется объем памяти в соответствии с его типом; в этом случае данные можно инициализировать, т.е. задать его начальное значение.

Переменная – это область памяти, в которой хранятся данные определенного типа. У переменной есть *имя* и *значение*. Имя переменной служит для обращения к области памяти, в которой храниться значение.

Общий вид определения переменных:
TypeID name [= инициализатор];

TypeID – это любой тип данных языка C/C++
name – это идентификатором.
Инициализатор – это начальное значение, которое задается переменной.
Например: **int** j = 10, m = 3, n = j + m;

1.3. ОПЕРАТОРЫ И ВЫРАЖЕНИЯ

Выражения используются для вычисления значений (определенного типа) и состоят из операндов, операций и скобок. Каждый операнд может быть, в свою очередь, выражением.

Знак операции – это один или более символов, определяющих действие над операндами. Операции делятся на унарные, бинарные и тернарные – по количеству участвующих в них операндов; выполняются в соответствии с приоритетами – для изменения порядка выполнения операций используются круглые скобки.

Большинство операций выполняется слева направо, например, a+b+c → (a+b)+c. *Исключение: унарные операции, операции присваивания и тернарная условная операция (?:) – справа налево*.

Арифметические операции

Арифметические операции – бинарные, их обозначения:
+ (сложение); – (вычитание); * (умножение);
/ (деление, для int операндов – с отбрасыванием остатка);
% (остаток от деления целочисленных операндов со знаком первого операнда – деление «по модулю»).
Операндами традиционных арифметических операций (+ – * /) могут быть *константы, переменные, функции, элементы массивов, указатели, любые арифметические выражения*.

Примеры представления математических формул в выражениях написанных на языке *Си* представлены на рисунке 1.2.

Математическая формула:	Выражение на C/C++
$x + yz - \frac{a}{b+c}$	x + y*z – a/(b+c)
b^2 -4ac	b*b – 4*a*c
$\frac{1}{x^2+x+3}$	1/(x*x +x+3)
$\frac{a+b}{c-d}$	(a+b)/(c-d)

Рис. 1.2

Порядок выполнения операций:
1) выражения в круглых скобках;
2) функции;
3) операции * / (выполняются слева направо);
4) операции – + (слева направо).

1.4.ФУНКЦИИ ВВОДА ВЫВОДА ДАННЫХ

Функции вывода данных

Функция форматированного вывода данных

Для вывода информации чаще всего используется функция форматированного вывода данных:

int *printf* (const char*, …);

управляющая строка - указывает компилятору вид выводимой информации и содержит *спецификации преобразования, управляющие символы и комментарии*.

Функция **printf()** не обеспечивает автоматического перехода курсора на новую строку, так что многократное обращение к ней можно использовать для поэтапной сборки выходной строки.

Для регулировки курсора, на экране результатов, необходимо использовать ***управляющие символы:***

\n переводит курсор на новую строку
\t переводит курсор на горизонтальную табуляцию
\a звуковой сигнал
\b сдвиг текущей позиции влево
\r сдвиг в начало строки
\' вывод апостроф
\" вывод кавычки
\\ вывод левая косая черта
\? вывод вопросительный знак
%% вывод символа %

спецификация преобразования имеет вид:

% [*флаг*] [*размер поля* . *точность*] *спецификация*

спецификация - тип выводимой информации.

Спецификация *для символьного типа*:
%**c** один символ
%**s** символьная строка

Спецификации для *целого типа*:
%d **int**
%hd **short int**
%ld **long int**

Спецификации для *вещественного типа*:
%f **float**
%lf **double**

флаг может принимать следующие значения:
- выравнивание влево выводимого числа;
+ выводится знак положительного числа;

размер поля – минимальная ширина поля, т.е. длина числа, при недостаточной ширине поля выполняется автоматическое расширение;

точность – количество цифр в дробной части числа;

список вывода - печатаемые объекты (константы, переменные или выражения, вычисляемые перед выводом) по количеству, порядку следования и типу должны соответствовать спецификациям преобразования в управляющей строке.

Функция вывода строк

Функция ***puts()*** выводит на экран дисплея только строку символов, автоматически добавляя к ней символ перехода на начало новой строки (\n).

Примеры:

1.
puts("Hello!!!"); тоже, что **printf("Hello!!!\n");**

2.
puts(""); тоже, что printf("\n");

3.
char str[100]="Hello!!!!";
puts(str); **printf("%s\n",str);**

Функция вывода строк

Функция ***putchar***() выдает на экран дисплея только один символ без добавления символа ‘\n’.
1.
***putchar*(‘a’); printf(“%c”, ‘a’);**
2.
char ch = 96;
putchar(ch); printf(“%c”,ch);

Функции ввода данных

Функция форматированного ввода данных

Для форматированного ввода информации используется функция:

int *scanf* (const char*, …);

управляющая строка – *может содержать только спецификации* (%d, %hd, %f, %lf, %c, %s и т.д.). Причем количество, тип и порядок следования спецификаций должны совпадать с количеством, типом и порядком следования вводимых данных, иначе результат ввода непредсказуем.

список ввода - использует указатели на переменные, т.е. их адреса, а не просто имена переменных. Для обозначения адреса перед именем переменной записывается символ **&.**

Например:

```
int course;
float grant;
char name[20];
printf (" Укажите курс, стипендию, имя \n ");
scanf ("%d%f%s",&course, &grant,  name);
```

Ввод данных для функции ***scanf*** завершается пробелом или *enter*-ом. Вводить данные можно как в одной строке через пробел, так и в разных строках.

Если до ввода символа что-либо вводилось, то для того чтобы была возможность ввести символ обязательно необходимо очистить буфер клавиатуры.

Очистка буфера клавиатуры осуществляется с помощью функции **fflush(stdin);**

Функция ввода строки

Функция ***gets()*** – используется **только для ввода символьных строк.** Ввод строки завершается нажатием клавиши Enter.

```
char name[20];
gets(name);
```

Функция ввода символа

Функция ***getch*()**

- находится в библиотечном файле **conio.h**
- возвращает код нажатой пользователем клавиши;
- причем вводимый символ не отображается на экране результатов;
- для получения расширенного кода функциональных или курсорных клавиш необходимо повторно вызвать данную функцию.

```
char ch;
ch = getch();
printf("\nsimvol %c  kod = %d\n",ch, ch);
```