1. Препроцессор - программа, подготавливающая код к компиляции. Препроцессор удаляет из кода комментарии, преобразует код в соответствии с макросами и выполняет иные директивы, начинающиеся с символа «#» (такие как #include, #define, разнообразные директивы типа #pragma).

Директивы — это специальные команды, которые препроцессор распознаёт и выполняет. Все директивы начинаются со знака #. Если первый непробельный символ в строке это — #, то препроцессор будет пытаться распознать в ней свою директиву.  
Существуют следующие директивы:  
— Подключение файлов: #include, #include\_next.

Встретив в исходнике эту директиву, препроцессор заменяет её на содержимое файла, имя которого указанно в параметре.

— Условная компиляция: #if, #ifdef, #ifndef, #else, #elif and #endif.

Применяется, когда в зависимости от значения различных макросов, нужно компилировать, или нет, тот или иной кусок кода, или установить другие макросы.

— Диагностика: #error, #warning, #line.

Назначение дириктивы **#error** предельно просто — остановить компиляцию с сообщением об ошибке, указанном после директивы. Её можно использовать совместно с директивами условной компиляции для того, чтоб убедиться установлен ли какой-то важный макрос и, что он имеет правильное значение.  
Также существует директива **#warning**, аналогична #error, но не прерывает компиляцию, а выдаёт предупреждение.  
Директива **#line** служит для задания номеров строк и имени файла, показываемых в сообщениях об ошибках и возвращаемые специальными макросами \_\_LINE\_\_ и \_\_FILE\_\_.

— Дополнительная информация компилятору: #pragma

Эта директива используется в основном для передачи компилятору различных нестандартных опций, таких как уровни оптимизаций для отдельных частей программы, выравнивания структур, параметров компоновщика и много чего ещё.

— Макроопределения: #define

Существуют два типа макросов: макрос-объект(object-like macro) и макрос-функция(function-like macro), оба типа объявляются с помощью директивы **#define**. Рассмотрим сначала макросы-объекты. Объявляются они как:  
**#define ИМЯ\_МАКРОСА [замещающий текст]**  
Всё, что идёт после имя макроса до конца строки является замещающим текстом.

Второй вид макросов — это макро-функции (function-like macros). Определяются они с помощью той-же директивы #define, после которой (сразу без пробелов) в круглых скобках идёт список разделённых запятыми аргументов.

2. Как минимум одну - функцию main.

3. Функция main вызывается при старте программы после инициализации нелокальных объектов со статической длительностью хранения. Это точка входа в программу, которая исполняется в гостевом окружении (то есть с операционной системой). Точки входа в автономные программы (boot loaders, OS kernels, и т.п.) зависят от реализации.

4. Есть два вида комментариев: многострочный комментарий и однострочный комментарий.

/\*Многострочный комментарий \*/

Комментарии могут находиться в любом месте программы, за исключением середины ключевого слова или идентификатора. Многострочные комментарии не могут быть вложенными. То есть в одном комментарии не может находиться другой.

// Однострочный комментарий

Однострочные комментарии особенно полезны тогда, когда нужны краткие, не более чем в одну строку пояснения. Однострочный комментарий может находиться внутри многострочного комментария.

Комментарии должны находиться там, где требуется объяснить работу кода. Например, в начале всех функций, за исключением самых очевидных, должен быть комментарий, который сообщает, что именно делает функция, как она вызывается и что возвращает.

5. Операция присваивания обозначается знаком = , и используется для реализации оператора присваивания. В отличие от других языков в языке С оператор присваивания может использоваться “внутри” различных управляющих операторов, речь о которых пойдет далее.

В фрагменте If ((c=a-b)<0) printf (“число а меньше чем b“) сначала вычисляется величина a-b, которая присваивается переменной c, затем сравнивается ее значение с нулем. Кроме того, в языке С имется возможность многократного присваивания, например: a=b=c=x+y. Здесь сначала вычисляется значение x+y, затем оно присваивается переменной c, потом b, и лишь затем a. В левой части оператора присваивания должно стоять выражение, которому можно присвоить значение. Такое выражение в языке С, например просто переменная, называется величиной lvalue. Выражение 3 = 3 ошибочно, так как константе нельзя присвоить никакое значение: константа не является величиной lvalue.

В языке С имеются дополнительные операции присваивания +=, -=,/=, \*=, %=.

Вместо выражения a=a+7 можно использовать выражение a+=7. Здесь += аддитивная операция присваивания, в результате выполнения которой величина, стоящая справа, прибавляется к значению переменной, стоящей слева. Аналогично выполняются остальные дополнительные операции присваивания.

Дополнительные операции присваивания имеют тот же приоритет, что и операция =, т. е. ниже, чем приоритет арифметических операций. Необходимо отметить, что операция a+=7 выполняется быстрее, чем операция а=а+7.

6. Функция printf() предназначена для форматированного вывода. Она переводит данные в символьное представление и выводит полученные изображения символов на экран. При этом у программиста имеется возможность форматировать данные, то есть влиять на их представление на экране.

Общая форма записи функции printf():

printf("**СтрокаФорматов**", объект 1, объект 2, ..., объект n);

**СтрокаФорматов** состоит из следующих элементов:

* управляющих символов;
* текста, представленного для непосредственного вывода;
* форматов, предназначенных для вывода значений переменных различных типов.

Объекты могут отсутствовать.

Управляющие символы не выводятся на экран, а управляют расположением выводимых символов. Отличительной чертой управляющего символа является наличие обратного слэша '\' перед ним.

Основные управляющие символы:

* '\n' — перевод строки;
* '\t' — горизонтальная табуляция;
* '\v' — вертикальная табуляция;
* '\b' — возврат на символ;
* '\r' — возврат на начало строки;
* '\a' — звуковой сигнал.

Форматы нужны для того, чтобы указывать вид, в котором информация будет выведена на экран. Отличительной чертой формата является наличие символа процент '%' перед ним:

* %d — целое число типа int со знаком в десятичной системе счисления;
* %u — целое число типа unsigned int;
* %x — целое число типа int со знаком в шестнадцатеричной системе счисления;
* %o — целое число типа int со знаком в восьмеричной системе счисления;
* %hd — целое число типа short со знаком в десятичной системе счисления;
* %hu — целое число типа unsigned short;
* %hx — целое число типа short со знаком в шестнадцатеричной системе счисления;
* %ld — целое число типа long int со знаком в десятичной системе счисления;
* %lu — целое число типа unsigned long int;
* %lx — целое число типа long int со знаком в шестнадцатеричной системе счисления;
* %f — вещественный формат (числа с плавающей точкой типа float);
* %lf — вещественный формат двойной точности (числа с плавающей точкой типа double);
* %e — вещественный формат в экспоненциальной форме (числа с плавающей точкой типа float в экспоненциальной форме);
* %c — символьный формат;
* %s — строковый формат.

7.

8.

int dir( )/\* dir\*/ **/\*описание функции dir и многострочный комментарий с ее именем\*/**

{ **/\*начало функции\*/**

printf("добрый день\n"); **/\*печать на экран текста «добрый день» и перевод строки \*/**

printf(" мы рады вас видеть\n"); **/\*печать на экран текста «рады вас видеть» и перевод строки \*/**

} **/\*конец функции\*/**

int main () **/\*описание функции main, начало программы\*/**

{ **/\*начало функции\*/**

dir (); **/\*вызов функции dir\*/**

printf("надеемся на плодотворную работу \n"); **/\*печать на экран текста «надеемся на плодотворную работу» и перевод строки \*/**

printf("и замечательные успехи\n"); **/\*печать на экран текста «замечательные успехи» и перевод строки\*/**

} **/\*конец функции\*/**

9. printf("сейчас %d год \n", year);

printf() // функция вывода на экран;

"сейчас %d год \n" // первый аргумент;

%d // целое число типа int со знаком в десятичной системе счисления;

\n // перенос на следующую строку;

year // второй аргумент.

10. В языке различают понятия "тип данных" и "модификатор типа". Тип данных - это, например, целый, а модификатор - со знаком или без знака. Целое со знаком будет иметь как положительные, так и отрицательные значения, а целое без знака - только положительные значения. В языке Си можно выделить пять базовых типов, которые задаются следующими ключевыми словами:

* char - символьный;
* int - целый;
* float - вещественный;
* double - вещественный двойной точности;
* void - не имеющий значения.

Дадим им краткую характеристику:

* Переменная типа char имеет размер 1 байт, ее значениями являются различные символы из кодовой таблицы, например: 'ф', ':', 'j' (при записи в программе они заключаются в одинарные кавычки).
* Размер переменной типа int в стандарте языка Си не определен. В большинстве систем программирования размер переменной типа int соответствует размеру целого машинного слова. Например, в компиляторах для 16-разрядных процессоров переменная типа int имеет размер 2 байта. В этом случае знаковые значения этой переменной могут лежать в диапазоне от -32768 до 32767.
* Ключевое слово float позволяет определить переменные вещественного типа. Их значения имеют дробную часть, отделяемую точкой, например: -5.6, 31.28 и т.п. Вещественные числа могут быть записаны также в форме с плавающей точкой, например: -1.09e+4. Число перед символом "е" называется мантиссой, а после "е" - порядком. Переменная типа float занимает в памяти 32 бита. Она может принимать значения в диапазоне от 3.4е-38 до 3.4e+38.
* Ключевое слово double позволяет определить вещественную переменную двойной точности. Она занимает в памяти в два раза больше места, чем переменная типа float (т.е. ее размер 64 бита). Переменная типа double может принимать значения в диапазоне от 1.7e-308 до 1.7e+308.
* Ключевое слово void (не имеющий значения) используется для нейтрализации значения объекта, например, для объявления функции, не возвращающей никаких значений.

Объект некоторого базового типа может быть модифицирован. С этой целью используются специальные ключевые слова, называемые модификаторами. В стандарте ANSI языка Си имеются следующие модификаторы типа:

* unsigned
* signed
* short
* long

Модификаторы записываются перед спецификаторами типа, например: unsigned char. Если после модификатора опущен спецификатор, то компилятор предполагает, что этим спецификатором является int. Таким образом, следующие строки:

long а;

long int а;

являются идентичными и определяют объект а как длинный целый. Таблица иллюстрирует возможные сочетания модификаторов (unsigned, signed, short, long) со спецификаторами (char, int, float и double), а также показывает размер и диапазон значений объекта (для 16-разрядных компиляторов).

|  |  |  |
| --- | --- | --- |
| Тип | Размер в байтах (битах) | Интервал изменения |
| char | 1 (8) | от -128 до 127 |
| unsigned char | 1 (8) | от 0 до 255 |
| signed char | 1 (8) | от -128 до 127 |
| int | 2 (16) | от -32768 до 32767 |
| unsigned int | 2 (16) | от 0 до 65535 |
| signed int | 2 (16) | от -32768 до 32767 |
| short int | 2 (16) | от -32768 до 32767 |
| unsigned short int | 2 (16) | от 0 до 65535 |
| signed short int | 2 (16) | от -32768 до 32767 |
| long int | 4 (32) | от -2147483648 до 2147483647 |
| unsigned long int | 4 (32) | от 0 до 4294967295 |
| signed long int | 4 (32) | от -2147483648 до 2147483647 |
| float | 4 (32) | от 3.4Е-38 до 3.4Е+38 |
| double | 8 (64) | от 1.7Е-308 до 1.7Е+308 |
| long double | 10 (80) | от 3.4Е-4932 до 3.4Е+4932 |

Все переменные до их использования должны быть определены (объявлены). При этом задается тип, а затем идет список из одной или более переменных этого типа, разделенных запятыми. Например:

int a, b, c;

char x, y;

В языке различают понятия объявления переменной и ее определения. Объявление устанавливает свойства объекта: его тип (например, целый), размер (например, 4 байта) и т.д. Определение наряду с этим вызывает выделение памяти (в приведенном примере дано определение переменных). Переменные можно разделять по строкам произвольным образом, например:

float a;

float b;

Переменные в языке Си могут быть инициализированы при их определении:

int a = 25, h = 6;

char g = 'Q', k = 'm';

float r = 1.89;

long double n = r\*123;

Выясним теперь, где в тексте программы определяются данные. В языке возможны глобальные и локальные объекты. Первые определяются вне функций и, следовательно, доступны для любой из них. Локальные объекты по отношению к функциям являются внутренними. Они начинают существовать, при входе в функцию и уничтожаются после выхода из нее. Ниже показана структура программы на Си и возможные места в программе, где определяются глобальные и локальные объекты.

int a; /\* Определение глобальной переменной \*/

int function (int b, char c); /\* Объявление функции (т.е. описание ее заголовка)\*/

void main (void)

{ //Тело программы

int d, e; //Определение локальных переменных

float f; //Определение локальной переменной

...

}

int function (int b, char c) /\* Определение функции и формальных

параметров (по существу - локальных

переменных) b и c \*/

{ //Тело функции

char g; //Определение локальной переменной

...

}

Отметим, что выполнение программы всегда начинается с вызова функции main( ), которая содержит тело программы. Тело программы, как и тело любой другой функции, помещается между открывающей и закрывающей фигурными скобками.

В языке Си все определения должны следовать перед операторами, составляющими тело функции. Если они сделаны в функции, то соответствующие объекты будут локальными, а если вне функций, то глобальными.

Наряду с переменными в языке существуют следующие виды констант:

* *вещественные,* например 123.456, 5.61е-4. Они могут снабжаться суффиксом F (или f), например 123.456F, 5.61e-4f;
* *целые,* например 125;
* *короткие целые,* в конце записи которых добавляется буква (суффикс) H (или h), например 275h, 344H;
* *длинные целые,* в конце записи которых добавляется буква (суффикс) L (или l), например 361327L;
* *беззнаковые,* в конце записи которых добавляется буква U (или u), например 62125U;
* *восьмеричные,* в которых перед первой значащей цифрой записывается нуль (0), например 071;
* *шестнадцатеричные,* в которых перед первой значащей цифрой записывается пара символов нуль-икс (0x), например 0x5F;
* *символьные* - единственный символ, заключенный в одинарные кавычки, например 'О', '2', '.' и т.п. Символы, не имеющие графического представления, можно записывать, используя специальные комбинации, например \n (код 10), \0 (код 0). Эти комбинации выглядят как два символа, хотя фактически это один символ. Так же можно представить любой двоичный образ одного байта: '\NNN', где NNN - от одной до трех восьмеричных цифр. Допускается и шестнадцатеричное задание кодов символов, которое представляется в виде: '\х2В', '\хЗ6' и т.п.;
* *строковые* - последовательность из нуля символов и более, заключенная в двойные кавычки, например: "Это строковая константа". Кавычки не входят в строку, а лишь ограничивают ее. Строка представляет собой массив из перечисленных элементов, в конце которого помещается байт с символом '\0'. Таким образом, число байтов, необходимых для хранения строки, на единицу превышает число символов между двойными кавычками;
* *константное выражение,* состоящее из одних констант, которое вычисляется во время трансляции (например: а=60+301);
* типа *long double,* в конце записи которых добавляется буква L (или l), например: 1234567.89L.

11. 40Hours – идентификатор начинается с цифры;

Get Data – использован пробел;

box-22 – использован знак «-»;

cost\_in\_$ - использован знак «$»;

int – идентификатор совпадает с именем функции.

12. В Си прописные и строчные буквы – различные символы. Поэтому машина воспримет их как две разные переменные.

13.

|  |  |
| --- | --- |
| переменная | выражение |
| alpha | 2856 |
| num | alpha |
| rate | 0.36 |
| ch | ‘b’ |

14. - вычитание и унарный минус;

+ сложение;

\* умножение;

/ деление;

% деление по модулю;

Операции сложения, вычитания, умножения и деления действуют так же, как и в большинстве других языках программирования. Они могут применяться ко всем встроенным типам данных. Операции выполняются слева направо, т. е. сначала вычисляется выражение левого операнда, затем выражение, стоящее справа от знака операции. Если операнды имеют один тип, то результат арифметической операции имеет тот же тип. Поэтому, когда операция деления / применяется к целым переменным или символьным переменным, остаток отбрасывается. Операция деление по модулю % дает остаток от целочисленного деления. Такая операция может применяться только к целочисленным переменным.

15.

++ увеличение на единицу (инкрементация );

- - уменьшение на единицу (декрементация ).

Язык С предоставляет программисту еще две очень полезные и специфические операции – унарные операции ++ и —. Операция ++ прибавляет единицу к операнду, операция – вычитает единицу из операнда. Обе операции могут следовать перед операндом или после операнда (префиксная и постфиксная формы). Три написанные ниже оператора дают один и тот же результат, но имеют различие при использовании в выражениях.

a++ – значение переменной a сначала используется в выражении, и лишь затем увеличивается на единицу;

++a – переменная a сначала увеличивается на единицу, а затем ее значение используется в выражении.

16. 20

17. 25

18. . < //меньше

> //больше

<= //меньше или равно

>= //больше или равно

= //равно

!= //не равно

19. && //и

|| //или

! //не

20. Есть два простых способа ввести строку с клавиатуры. Первый способ – воспользоваться функцией scanf() со спецификатором ввода %s. Надо помнить , что функция scanf() вводит символы до первого пробельного символа. Второй способ – воспользоваться функцией gets(), объявленной в файле stdio.h. Функция gets() позволяет вводить строки, содержащие пробелы. Ввод заканчивается нажатием клавиши Enter. Обе функции автоматически ставят в конце строки нулевой байт. В качестве параметра в этих функциях используется имя массива.

21. Стандартная библиотека ANSI Си состоит из 24 заголовочных файлов, каждый из которых можно подключать к программному проекту при помощи одной директивы. Каждый заголовочный файл содержит объявления одной или более функций, определения типов данных и макросы. Содержание этих заголовочных файлов перечисляется ниже.

|  |  |
| --- | --- |
| **<[assert.h](https://ru.wikipedia.org/wiki/Assert.h" \o "Assert.h)>** | Содержит макрос утверждений, используемый для обнаружения логических и некоторых других типов ошибок в отлаживаемой версии программы. |
| **<[complex.h](https://ru.wikipedia.org/wiki/Complex.h" \o "Complex.h)>** | Набор функций для работы с комплексными числами. (Появилось в **C99**) |
| **<[ctype.h](https://ru.wikipedia.org/wiki/Ctype.h" \o "Ctype.h)>** | Содержит функции, используемые для классификации символов по их типам или для конвертации между верхним и нижним регистрами независимо от используемой кодировки (обычно ASCII или одно из её расширений, хотя есть и реализации, использующие EBCDIC). |
| **<[errno.h](https://ru.wikipedia.org/wiki/Errno.h" \o "Errno.h)>** | Для проверки кодов ошибок, возвращаемых библиотечными функциями. |
| **<[fenv.h](https://ru.wikipedia.org/wiki/Fenv.h" \o "Fenv.h)>** | Для управления средой, использующей числа с плавающей точкой. (Появилось в **C99**) |
| **<[float.h](https://ru.wikipedia.org/wiki/Float.h" \o "Float.h)>** | Содержит заранее определенные константы, описывающие специфику реализации свойств библиотеки для работы с числами с плавающей точкой, как, например, минимальная разница между двумя различными числами с плавающей точкой (\_EPSILON), максимальное число цифр точности (\_DIG) и область допустимых чисел (\_MIN, \_MAX). |
| **<[inttypes.h](https://ru.wikipedia.org/wiki/Inttypes.h" \o "Inttypes.h)>** | Для точной конвертации целых типов. (Появилось в **C99**) |
| **<**[**iso646.h**](https://ru.wikipedia.org/wiki/Iso646.h)**>** | Для программирования в кодировке ISO 646. (Появилось в **NA1**) |
| **<[limits.h](https://ru.wikipedia.org/wiki/Limits.h" \o "Limits.h)>** | Содержит заранее заданные константы, определяющие специфику реализации свойств целых типов, как, например, область допустимых значений (\_MIN, \_MAX). |
| **<[locale.h](https://ru.wikipedia.org/wiki/Locale.h" \o "Locale.h)>** | Для setlocale() и связанных констант. Используется для выбора соответствующего языка. |
| **<[math.h](https://ru.wikipedia.org/wiki/Math.h" \o "Math.h)>** | Для вычисления основных математических функций |
| **<[setjmp.h](https://ru.wikipedia.org/wiki/Setjmp.h" \o "Setjmp.h)>** | Объявляет макросы setjmp и longjmp, используемые для нелокальных переходов |
| **<[signal.h](https://ru.wikipedia.org/wiki/Signal.h" \o "Signal.h)>** | Для управления обработкой сигналов |
| **<[stdarg.h](https://ru.wikipedia.org/wiki/Stdarg.h" \o "Stdarg.h)>** | Для доступа к различному числу аргументов, переданных функциям. |
| **<[stdbool.h](https://ru.wikipedia.org/wiki/Stdbool.h" \o "Stdbool.h)>** | Для булевых типов данных. (Появилось в **C99**) |
| **<[stdint.h](https://ru.wikipedia.org/wiki/Stdint.h" \o "Stdint.h)>** | Для определения различных типов целых чисел. (Появилось в **C99**) |
| **<[stddef.h](https://ru.wikipedia.org/wiki/Stddef.h" \o "Stddef.h)>** | Для определения нескольких стандартных типов и макросов. |
| **<[stdio.h](https://ru.wikipedia.org/wiki/Stdio.h" \o "Stdio.h)>** | Реализует основные возможности ввода и вывода в языке Си. Этот файл содержит весьма важную функцию printf. |
| **<[stdlib.h](https://ru.wikipedia.org/wiki/Stdlib.h" \o "Stdlib.h)>** | Для выполнения множества операций, включая конвертацию, генерацию псевдослучайных чисел, выделение памяти, контроль процессов, окружения, сигналов, поиска и сортировки. |
| **<[string.h](https://ru.wikipedia.org/wiki/String.h" \o "String.h)>** | Для работы с различными видами строк. |
| **<[tgmath.h](https://ru.wikipedia.org/w/index.php?title=Tgmath.h&action=edit&redlink=1" \o "Tgmath.h (страница отсутствует))>** | Для типовых математических функций. (Появилось в **C99**) |
| **<[threads.h](https://ru.wikipedia.org/w/index.php?title=Threads.h&action=edit&redlink=1" \o "Threads.h (страница отсутствует))>** | Заголовочный файл <threads.h> наряду с <stdatomic.h> предоставляет поддержку для параллельного программирования. (Появилась в **C11**) |
| **<[time.h](https://ru.wikipedia.org/wiki/Time.h" \o "Time.h)>** | Для конвертации между различными форматами времени и даты. |
| **<[wchar.h](https://ru.wikipedia.org/wiki/Wchar.h" \o "Wchar.h)>** | Для обработки «широких» потоков и нескольких видов строк при помощи «широких» символов (поддержка набора языков). (Появилось в **NA1**) |
| **<[wctype.h](https://ru.wikipedia.org/wiki/Wctype.h" \o "Wctype.h)>** | Для классификации «широких» символов. (Появилось в **NA1**) |

22. . main()

{

int length, square;

float result;

printf(“ Введите значение длины окружности и ее площади через пробел: \n”);

scanf(“%d%d”,&length,&square);

result = length/square;

printf(“Результат - %f\n”,result);

}

23. main()

{

int a=10;

int b=70;

a++;

++b;

printf(“a=%d b=%d\n”,a,b);

printf(“a=%d b=%d\n”,a++,++b);

}