## § 6. ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЯ СИ

### 6.1. ОБЩАЯ ХАРАКТЕРИСТИКА ЯЗЫКА И ПРИМЕР ПРОГРАММЫ НА СИ

Язык программирования Си - это универсальный язык с богатым набором операторов и компактным способом записи выражений. Благодаря гибкости, выразительности и компактности своих конструкций Си завоевал наибольшую популярность в среде профессиональных программистов и широко используется при разработке системных и прикладных программ.

Язык Си представляет собой удачный компромисс между желанием располагать теми возможностями, которые обычно предоставляют программисту столь понятные и удобные языки высокого уровня, и стремлением эффективно использовать особенности компьютера. Кроме набора средств, присущих современным языкам программирования высокого уровня (структурность, модульность, определяемые типы данных) в него включены средства для программирования «почти» на уровне ассемблера (использование указателей, побитовые операции, операции сдвига). Большой набор операторов позволяет писать компактные и эффективные программы. Однако, такие мощные средства требуют от программиста осторожности, аккуратности и хорошего знания языка со всеми его преимуществами и недостатками. В отличие от языков программирования типа Паскаль, требующих жесткой дисциплины программирования, ограничивающих свободу программиста, содействующих устранению многих ошибок в программах еще на стадии их трансляции, язык Си предоставляет программисту наибольшую свободу. Однако, ответственность за корректность программ при этом полностью ложится на программиста.

Си был создан Денисом Ритчи как инструмент для разработки операционной системы UNIX и реализован в рамках этой операционной системы. Название языка имеет случайное происхождение: «С» - третья буква английского алфавита. Это наименование говорит о чувстве юмора у создателей языка - его предшественником был язык В («В» - вторая буква английского алфавита).

В настоящее время имеется большое количество систем программирования на Си для разных типов компьютеров. Разработано много библиотек модулей, инструментальных средств разработки и отладки, облегчающих создание новых программ. Программы на Си обладают высокой мобильностью, без каких-либо изменений они переносятся, транслируются и выполняются на машинах различных типов.

Язык Си - компилирующего типа. Текст программы на Си, подготовленный с помощью текстового редактора, для получения объектного модуля обрабатывается компилятором, причем компиляция выполняется в два прохода. При первом проходе (претрансляцни) обрабатываются строки-директивы, начинающиеся со знака #, при втором - транслируется текст программы и создается объектный (машинный) код. Для получения загрузочного (исполняемого) модуля теперь необходимо отредактировать внешние связи объектного модуля - подсоединить к нему соответствующие библиотечные модули.

Рассмотрим простую программу на языке Си. Такой пример позволит нам выявить некоторые основные черты любой программы, написанной на языке Си.

*Программа 82*

#include<stdio.h>

main() /\* Простая программа \*/ <

int num;

num =1,'

printf ("Это моя 1-я программа на языке Си.\n",num);)

Результат выполнения программы:

Это моя 1-я программа на языке Си.

Поясним строки этой программы.

#include<stdio.h> - подключение файла stdio.h, который является частью пакета, имеющегося в любом компиляторе языка Си и описывающего функции ввода-вывода (например, средства взаимодействия программы с терминалом). В качестве имени файла используется аббревиатура английских слов: STandard Input/Output header - стандартный заголовок ввода-вывода. Данная строка не является оператором языка Си. Символ # оказывает, что она должна быть обработана «препроцессором» компилятора. Препроцессор осуществляет некоторую предварительную обработку текста программы перед началом компиляции.

Main() - имя функции (в переводе с английского main - «главная»). Любая программа, написанная на языке Си, состоит из одной или более «функций», являющихся модулями, из которых она собирается. Данная программа состоит из одной функции main. Круглые скобки указывают именно на то, что main() - имя функции. Программа, написанная на языке Си всегда начинает выполняться с функции, называемой main(). Программист имеет возможность выбирать имена для всех используемых им функций, кроме main. В круглых скобках в общем случае содержится информация (параметры), передаваемые этой функции. В нашем примере параметры не требуются и, следовательно, в скобках ничего не содержится.

/\* Простая программа \*/-комментарий. Использование комментариев облегчает процесс понимания программы любым программистом. Большим удобством при написании комментариев является возможность располагать их в той же строке, что и операции, которые они объясняют. Длинный комментарий может помещаться в отдельной строке или даже занимать несколько строк. Все, что находится между символом, указывающим начало комментария /\*, и символом, указывающим его

конец \*/, игнорируется компилятором, поскольку он не в состоянии интерпретировать язык, отличный от Си.

{ } - фигурные скобки отмечают начало и конец тела функции. Фигурные скобки применяются также для того, чтобы объединить несколько операторов программы в сегмент или «блок».

int num; - оператор описания. С помощью такого оператора мы объявляем, что будем использовать в программе переменную num, которая принимает целые (int) значения.

Точка с запятой в конце строки является частью оператора языка Си, а не разделителем операторов, как в Паскале.

Слово int служит «ключевым словом», определяющим один из основных типов данных языка Си. Ключевыми словами называются специальные зарезервированные слова, используемые для построения фраз языка.

В языке Си все переменные должны быть объявлены. Это означает, что, во-первых, в начале программы необходимо привести список всех используемых переменных, а во-вторых, необходимо указать тип каждой из них. Вообще, объявление переменных считается «хорошим стилем» программирования.

num=l; - оператор присваивания. Служит для присваивания переменной num значения 1.

printf("этo моя %d-я программа на языке Си.\n",num); - вызов функции printf() форматного вывода на печать. Этот оператор выводит на печать значение переменной num в формате, содержащемся в строке в кавычках (в данном случае печатается приведенная в кавычках фраза. Символы %d указывают компилятору, где в выводимой строке и в какой форме необходимо вставить значение переменной num. Символ % сигнализирует программе, что, начиная с этой позиции, необходимо вывести значение переменной, a d указывает, что число надо печатать в десятичном формате.

Символы \n не появляются на экране. Эти символы служат директивой начать новую строку на устройстве вывода. Комбинация символов \n на самом деле представляет собой один символ, называемый «новая строка». Для этого символам (\n) не существует соответствующей клавиши на клавиатуре. Символ «новая строка» служит примером «управляющей последовательности», которую невозможно ввести с клавиатуры.

В общем случае обращение к этой функции имеет вид:

printf(<фopмaт>,<выpaжeниel>,<выpaжeниe2>,...,<выpaжeниeN>);

где <выражение1>, <выражение2>, ..., <выражение№» - произвольные выражения, результаты которых надо вывести.

Управляющая строка «формат» содержит объекты двух типов: обычные символы, которые просто копируются в выходной поток (печатаются), и спецификации преобразования значений из внутреннего машинного представления в текстовое для вывода на печатающем устройстве.

Каждая спецификация преобразования начинается с символа % и заканчивается символом преобразования. Между % и символом преобразования могут (не обязательно) находиться

1) знак минус, указывающий, что преобразуемый параметр должен быть выровнен влево в своем поле;

2) целое число, задающее минимальный размер поля: преобразованный параметр будет напечатан в поле-минимум оказанного размера; если в преобразованном параметре символов меньше, чем размещается в указанном поле, то слева будут добавлены пробелы (или справа, если указано выравнивание влево);

3) строка цифр с начальным нулем - лишние позиции поля заполняются нулями, а не пробелами:

4) точка, отделяющая размер поля от последующей строки цифр (только для преобразования строк %s) - строка цифр указывает максимальное число символов, выводимых в одной строке.

Символы преобразования:

d - десятичное число со знаком;

u - десятичное число без знака;

О - восьмеричное число без знака (и без ведущего 0);

х - шестнадцатеричное число без знака (и без ведущего 0);

s - символьная строка;

с - одиночный символ;

f- действительное число в представлении с фиксированной точкой;

е-действительное число в экспоненциальном представлении;

g - наиболее короткое представление действительного числа; и др.

Функцией ввода, аналогичной функции вывода printf(), является scanf() - стандартная функция форматного ввода.

Обращение к этой функции имеет вид

sсаnf(<формат>,<&имя1>,<&имя2>,...,<&имяN>);

где <имя1>, <имя2>, ..., <имяN> - имена переменных, значения которых надо ввести. Наличие символа "&" перед каждым именем обязательно (кроме переменных строкового типа), его смысл будет пояснен ниже.

При обращении к функции scanf выполнение программы приостанавливается, ожидается ввод значений указанных переменных, после чего работа программы продолжается.

В качестве спецификаций в формате можно использовать те же символы, что и в функции printf(). Спецификации формата должны соответствовать количеству и типу вводимых переменных. В управляющей строке функции scanf нет промежуточных управляющих символов между % и символом преобразования, за исключением "\*". Если присутствует символ \*, входное поле просто пропускается и никакого присваивания не производится. Обычные символы (кроме °о) сопоставляются с очередными (отличными от символов промежутков) символами входного потока, и в случае отличия дальнейший ввод игнорируется.

Программа 83

#include<stdio.h>

main()

(

int data,month,year; char name[15],town[15];

printf("Kaк вас зовут? "); scanf("%s",name); printf("Укажите дату, месяц и год вашего рождения.\пДата:");

scanf("%d",%data) ;

printf ("Месяц (числом) :") ; scanf ("%d", Sinonth) ;

printf("Год:"); scant("%d",&year);

printf("А в каком городе? "); scanf("%s",town);

printf("Вот мы и познакомились...\n");

printf("Вас зовут %s ",name); printf("Вы родились в городе %s (%d.%d.%d)",town,data, month,year);)

Результат работы программы:

*Как вас зовут? Иван*

*Укажите дату, месяц и год вашего рождения.*

*Дата : 23*

*Месяц (номером): 02*

*Год: 1054*

*А в каком городе? Новгород*

*Вот мы н познакомились...*

*Вас зовут Иван*

*Вы родились в городе Новгород*

*(23.02.1054)*

### 6.2. ЭЛЕМЕНТЫ СИ: АЛФАВИТ, ИДЕНТИФИКАТОРЫ, ЛИТЕРАЛЫ, СЛУЖЕБНЫЕ СЛОВА

Перечислим основные символы языка Си, образующие его **алфавит**:

1) строчные латинские буквы

abcdefghijklmnopqrstuvwxyz

2) прописные латинские буквы

ABCDEFGHIJKLMNOPQRSTUVWXYZ

3) арабские цифры

0123456789

4) специальные символы

\* звездочка

\_ подчеркивание

+ плюс

( круглая скобка левая

* минус

) круглая скобка правая

/ дробная черта

< меньше

\ обратная дробная черта

> больше

% процент

^ стрелка вверх

! восклицательный знак

[ квадратная скобка левая

= знак равенства ] квадратная скобка правая

? вопросительный знак

# номер

: двоеточие

{ фигурная скобка левая

; точка с запятой

} фигурная скобка правая

& коммерческое ''и"(амперсанд)

| вертикальная черта

' апостроф

~ черта сверху (тильда)

. точка

" кавычки

, запятая пробел

*5) управляющие символы*

\t горизонтальная табуляция

\n перевод строки и возврат каретки

\r возврат каретки

\f перевод страницы

\b возврат на шаг (на один символ)

Множество основных символов расширено буквами русского алфавита (строчными и прописными). Они могут быть использованы только в комментариях, символьных константах и строках.

**Лексемами** называют последовательностисимволов языка (идентификаторы, служебные слова, константы, строки, составные знаки операций, разделители). Лексемы разделяются пробелами и другими неграфическими символами языка.

**Идентификатор** - это последовательность латинских букв, цифр и символа ''\_", начинающаяся с буквы или символа "\_".

Прописные и строчные латинские буквы считаются различными! Например, у и Y - это разные имена. Рекомендуется в именах переменных использовать только строчные буквы.

Примеры правильных идентификаторов:

schetchik get\_line a 12 Parami **\_**ab

Примеры неправильных идентификаторов:

%ab 12abc -x вася

**Литералы** - это неизменяемые объекты языка (константы). Литерал может быть числовым, символьным или строковым Числовые литералы могут быть десятичными (целыми и вещественными, простыми и длинными), восьмеричными, шестнадцатиричными.

Примеры.

*I\** Целые десятичные литералы \*/

57 32000001 /\* длинный\*/ 2е3 5ЕЗ

/\* Вещественные десятичные литералы \*/

0.00 5.37.1е-3 6.34Е-2 .21е+56

Лидирующий нуль (0) указывает на числовой восьмеричный литерал:

030 /\* Десятичное 24 \*/

040 /\* Десятичное 32 - символ пробел \*/

Лидирующий 0х указывает на числовой шестнадцатиричный литерал:

0х22 /\* Десятичное 34 - символ "\*/

0х6С /\* Десятичное 108 - символ i \*/

Символьный литерал - это один символ, заключенный в одинарные кавычки:

-'c"\*"q'-

"\007" /\* Звонок, восьмеричный код после \ \*/

"\х0а" /\* Перевод на новую строку, шестнадцатиричный код после \х \*/

Последовательность символов, заключенных в двойные кавычки, называется строковым литералом. Примеры:

"STRING\n"

"" /\* Строчный литерал состоит из одного символа "\0" \*/

"Очень,"\

"очень,"\

"очень длинный строковый литерал!"

Следующие зарезервированные служебные (ключевые) слова языка запрещено использовать в качестве идентификаторов.

auto - автоматический;

default - по умолчанию;

break -завершить;

do -выполнить;

case -вариант;

double -двойной точности;

char -символьный;

else -иначе;

continue - продолжить;

entry - вход;

extern -внешний;

short -короткий;

for - для;

sizeof - размер;

float -плавающее;

static -статический;

goto - перейти;

struct - структура;

if — если;

switch - переключатель;

int - целое; '

typedef - определение типа;

long -длинное;

union -объединение;

register -регистровый;

unsigned -без знака;

return - возврат;

while - пока.

### 6.3. ТИПЫ ДАННЫХ И ОПЕРАЦИИ В ЯЗЫКЕ СИ. ВЫРАЖЕНИЯ

**Типы данных.** Программа на процедурных языках, к которым относится Си, представляет собой описание операций над величинами различных типов. Тип определяет множество значений, которые может принимать величина, и множество операций, в которых она может участвовать.

В языке Си типы связаны с именами (идентификаторами) величин, т. е. с переменными. С переменной в языке Си связывается ячейка памяти. Тип переменной задает размер ячейки, способ кодирования ее содержимого, допустимые преобразования над значением данной переменной. Все переменные должны быть описаны до их использования. Каждая переменная должна быть описана только один раз.

Описание состоит из спецификатора типа и следующего за ним списка переменных. Переменные в списке разделяются запятыми. В конце описания ставится точка с запятой.

*Примеры**описаний:*

char a,b; /\* Переменные а и b имеют тип

char \*/ intх; /\* Переменная х - типа int

\*/ char sym; /" Описаны переменные sym типа char;

\*/ int count.num; /\* num и count типа int \*/

Переменным могут быть присвоены начальные значения внутри их описаний. Если за именем переменной следует знак равенства и константа, то эта константа служит в качестве инициализатора.

*Примеры:* char backch = '\0';

int i = 0;

Рассмотрим основные типы в языке Си.

***int - целый ("integer").*** Значения этого типа - целые числа из некоторого ограниченного диапазона (обычно от- 32768 до 32767). Диапазон определяется размером ячейки для типа и зависит от конкретного компьютера. Кроме того, имеются служебные слова, которые можно использовать с типом int: short int («short integer» - «короткое целое»), unsigned int («unsigned integer» - «целое без знака»), long int («длинное целое»), которые сокращают или, наоборот, расширяют диапазон представления чисел.

***char***- символьный («character»). Допустимое значение для этоготипа — одинсимвол (не путать с текстом!). Символ записывается в апострофах.

*Примеры:* 'х"2"?'

В памяти компьютера символ занимает один байт. Фактически хранится не символ, а число - код символа (от 0 до 255). В специальных таблицах кодировки указываются все допустимые символы и соответствующие им коды.

В языке Си разрешается использовать тип char как числовой, т. е. производить операции с кодом символа, применяя при этом спецификатор целого типа в скобках - (int).

***float - вещественный (с плавающей точкой).*** Значения этого типа - числа, но, в отличии от char и int, не обязательно целые.

Примеры:

12.87 -316.12 -3.345е5 12.345e-15

***double - вещественные числа двойной точности.*** Этот тип аналогичен типу float, но имеет значительно больший диапазон значений (например, для системы программирования Borland-C от 1.7Е-308 до 1.7Е+308 вместо диапазона от 3.4Е-38 до 3.4Е+38 для типа float). Однако увеличение диапазона и точности представления чисел ведет к снижению скорости выполнения программ и неэкономному использованию оперативной памяти компьютера.

Обратите внимание на отсутствие в этом списке строкового типа. В языке Си нет специального типа, который можно было бы использовать для описания строк. Вместо этого строки представляются в виде массива элементов типа char. Это означает, что символы в строке будут располагаться в соседних ячейках памяти.

Необходимо отметить, что последним элементом массива является символ \0. Это «нуль-символ», и в языке Си он используется для того, чтобы отмечать конец строки. Нуль-символ не цифра 0; он не выводится на печать и в таблице кодов ASCII имеет номер 0. Наличие нуль-символа означает, что количество ячеек массива должно быть. по крайней мере, на одну больше,чем число символов, которые необходимо размещать в памяти.

Приведем пример использования строк.

Программа 84

# include<stdio.h> main()

{

char string[31] ;

scanf("%s",string) ;

printf("%s",string);

}

В этом примере описан массив из 31 ячейки памяти, в 30 из которых можно поместить один элемент типа char. Он вводится при вызове функции scanf("%s",string); "&"отсутствует при указании массива символов.

***Указатели****.* Указатель - некоторое символическое представление адресаячейкипамяти, отведенной для переменной.

Например, &name - указатель на переменную name;

Здесь & - операция получения адреса. Фактический адрес - это число, а символическое представление адреса &name является константой типа «указатель».

В языке Си имеются и переменные типа указатель. Точно так же, как значением переменной типа char является символ, а значением переменной типа int - целое число, значением переменной типа указатель служит адрес некоторой величины.

Если мы дадим указателю имя ptr, то сможем написать такой оператор:

ptr = &name;/\* присваивает адрес name переменной ptr \*/

Мы говорим в этом случае, что prt «указатель на» name. Различие между двумя формами записи: ptr и &name - в том, что prt - это переменная, в то время как &name - константа. В случае необходимости можно сделать так, чтобы переменная ptr указывала на какой-нибудь другой объект:

**ptr** = &bah; /\* ptr указывает на bah, а не на name \*/

Теперь значением переменной prt является адрес переменной bah. Предположим, мы знаем, что в переменной ptr содержится ссылка на переменную bah. Тогда для доступа к значению этой переменной можно воспользоваться операцией «косвенной адресации» \* :

val = \*ptr; /\* определение значения, на которое указывает ptr \*/ Последние два оператора, взятые вместе, эквивалентны следующему:

val = bah;

Итак, когда за знаком *&* следует имя переменной, результатом операции является адрес указанной переменной; &nurse дает адрес переменной nurse; когда за знаком \* следует указатель на переменную, результатом операции является величина, помещенная в ячейку памяти с указанным адресом.

*Пример:* nurse *=* 22;

ptr = &nuse; */\** указатель на nurse \*/

val = \*ptr;

Результат- присваивание значения 22 переменной val.

Недостаточно сказать, что некоторая переменная является указателем. Кроме этого необходимо сообщить, на переменную какого типа ссылается данный указатель. Причина заключается в том, что переменные разных типов занимают различное число ячеек памяти, в то время как для некоторых операций, связанных с указателями, требуется знать объем отведенной памяти.

*Примеры* правильного описания указателей: int \*pi; char \*pc;

Спецификация типа задает тип переменной, на которую ссылается указатель, а символ \* определяет саму переменную как указатель. Описание вида int \*pi; говорит, что pi - это указатель и что \*pi - величина типа int.

В языке Си предусмотрена возможность определения имен типов данных. Любому типу данных с помощью определения typedef можно присвоить имя и использовать это имя в дальнейшем при описании объектов.

Формат: typedef <старый тип> <новый тип> *Пример:* typedef long LARGE; /\* определяется тип large, эквивалентный типу long \*/

Имена производного типа рекомендуется записывать прописными буквами, чтобы они выделялись в тексте программы.

Определение typedef не вводит каких-либо новых типов, а только добавляет новое имя для уже существующего типа. Описанные таким способом переменные обладают точно теми же свойствами, что и переменные, описанные явно. Переименование типов используется для введения осмысленных или сокращенных имен типов, что повышает понятность программ, и для улучшения переносимости программ (имена одного типа данных могут различаться на разных компьютерах).

**Операции.** Язык Си отличается большим разнообразием операций (более 40). Здесь мы рассмотрим лишь основные из них, табл. 3.3.

***Арифметические операции****.* К ним относят

• сложение(+),

• вычитание (бинарное) (-),

• умножение (\*),

• деление (/),

• остаток от деления нацело (%),

• вычитание (унарное) (-) .

В языке Си принято правило: если делимое и делитель имеют тип int, то деление производится нацело, т е. дробная часть результата отбрасывается.

Как обычно, в выражениях операции умножения, деления и нахождения остатка выполняются раньше сложения и вычитания. Для изменения порядка действий используют скобки.

Программа 85

#include<stdio.h>

main()

(

int s;

5 = -3 + 4 \* 5 - 6; printf("%d\n",s);

s = -3 + 4%5 - 6; printf("%d\n",s);

s = -3 \* 4% - 6/5; printf("%d\n",s);

s= (7 + 6)%5/2; printf("%d\n",s);

}

Результат выполнения программы: 11 1 0 1

Таблица 3.3 Старшинство и порядок выполнения операций

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Приоритет | Операция | | Название | Порядок выполнения | |
| Высший | ()  []  ++  --  (тип)  \* | | Круглые скобки  Квадратные скобки Увеличение  Уменьшение  Приведение  Содержимое | Слева направо  Слева направо  Справа налево  Справа налево  Справа налево  Справа налево | |
| 1 | *&*  *-*  !  /*~*  sizeof  \* | | Адрес  Унарный минус Логическое «НЕ»  Инверсия битов  Размер объекта Умножение | Справа налево  Справа налево  Справа налево  Справа налево  Справа налево  Слева направо | |
| 2 | \  % | | Деление  Остаток | Слева направо  Слева направо | |
| 3 | +  - | | Сложение  Вычитание | Слева направо  Слева направо | |
| 4 | »  «  > | | Сдвиг вправо  Сдвиг влево  Больше | Слева направо  Слева направо  Слева направо | |
| 5 | *>=*  *<*  <= | Больше или равно  Меньше  Меньше или равно | | | Слева направо  Слева направо  Слева направо |
| 6 | = =  != | Равно  Не равно | | | Слева направо  Слева направо |
| 7 | & | Битовое «И» | | | Слева направо |
| 8 | ~ | Битовое исключающее «ИЛИ» | | | Слева направо |
| 9 | | | Битовое «ИЛИ» | | | Слева направо |
| 10 | && | Логическое «И» | | | Слева направо |
| 11 | ||  =  += | Логическое «ИЛИ»  Операция присвания Справа налево | | | Слева направо  Справа налево |
| 12 | - =  \*=  /=  %= | Специальная форма операции присваивания | | | Справа налево  Справа налево  Справа налево  Справа налево |

В программировании часто встречаются ситуации, когда надо увеличить или уменьшить значение некоторой переменной на 1. Для этого обычно выполняются оператор присваивания вида: s= s + 1;

В языке Си для этих действий существуют специальные операции:

• увеличение (+ **+),**

• уменьшение (--).

Следующие записи на языке Си являются эквивалентными:

i=i+l и i++; j=j-1 и j--;.

Символы "++" или "- -" записывается после имени переменной или перед ним.

*Пример:*

*s* + +; /\* s увеличить на единицу

\*/ t - -; /\* t уменьшить на единицу

\*/ + + а; /\* а увеличить на единицу

\*/ --b; /\* b уменьшить на единицу \*/

Как и обычные присваивания, увеличение и уменьшение можно использовать в выражениях. При этом существенно, с какой стороны от имени стоит знак "+ +"или "- -". Если знак стоит перед переменной (в этом случае говорят о префиксной форме операции), то сначала выполняется увеличение (уменьшение) значения переменной, а лишь затем полученный результат используется в выражении. Если же знак стоит после переменной (постфиксная форма операции), то в выражении используется старое значение переменной, которое затем изменяется.

Пример:

inti,j,s;

i = j = 2; /\* i и j получают значение 2 \*/

s = (i++) + (++J);

После выполнения этих действий переменные имеют такие значения:

i=3;j=3;s=5.

Операции увеличения ++ и уменьшения - - можно применять топько к переменным, выражения типа s=(i+j)++ являются незаконными. Кроме того, не рекомендуется:

1) применять операции увеличения или уменьшения к переменной, присутствующей в более чем одном аргументе функции,

2) применять операции увеличенияили уменьшения к переменной, которая входит в выражение более одного раза.

***Операции отношения и логические операции***

|  |  |
| --- | --- |
| Больше или равно | >= |
| Больше | > |
| Меньше или равно | <= |
| Меньше | < |
| Равно | == |
| Неравно | != |
| Логическое «и» | && |
| Логическое «или» | || |
| Отрицание «не» | ! |

Логическое значение «ложь» представляется целым нулевым значением, а значение «истина» представляется любым ненулевым значением

Выражения, связанные логическими операциями && и ||, вычисляются слева направо, причем вычисление значения выражения прекращается сразу же, как только становится ясно, будет ли результат истинен или ложен.

Старшинство операции && выше, чем у операции ||.

Программа 86

#include<stdio.h>

main()

(

int x, у, z;

x=l; y=l; z=0; x=x&&y||z; printf("%d\n",x);

x=x|| !y&&z; printf("%d\n",x) ;

x=y=l; z=x++-l; printf("%d\n",x);printf("%d\n",z) ;

z+=-x++ + ++y; printf("%d\n",x) ; printf("%d\n",z);

z=x/++x; printf("%d\n",x); printf("%d\n",z) ;

}

Результат выполнения программы: 1 1 2 0 3 0 4 1

***Битовые операции***

|  |  |
| --- | --- |
| Битовое «и»  Битовое «или»  Битовое исключающее «или»  Сдвиг влево  Сдвиг вправо  Инверсия битов (унарная операция) | & |  ~  «  »  \ ~ |

Программа 87

#include<stdio.h>

main()

(

int у, х, z, k;

x=03; y=02; z=01; k=x|y&z; printf("%d\n",k) ;

k=x|y&~z; printf("%d\n",k) ;

k=x^y&~z; printf("%d\n",k) ;

k=x&y&&z; printf("%d\n",k) ;

x=l; y=-l;

k=!x|x; printf("%d\n",k) ;

k=-x|x; printf("%d\n",k) ;

k=x^x; printf("%d\n",k) ;

x<<=3; printf("%d\n",x);

y<<=3; printf("%d\n",y);

y>>=3; printf("%d\n",y);

}

После выполнения программы получаем следующие результаты:

3 3 1 1 1 -1 0 8 -8 8 1 9 1

**Выражения**. Конструкции, включающие константы (литералы), переменные, знаки операции, скобки для управления порядком выполнения операций, обращения к функциям, называют выражениями.

Если в выражениях встречаются операнды различных типов,то они преобразуются к общему типу в соответствии с определенными правилами:

• переменные типа char интерпретируются как целые без знака (unsigned);

• переменные типа short автоматически преобразуются в int; если одиниз операндов имеет тип unsigned, то другой (другие) также преобразуется к типу unsigned и результат имеет тип unsigned;

• если один из операндов имеет тип int, то другой (другие) также преобразуется ктипу int и результат имеет тип int;

• если один из операндов имеет тип char, то другой (другие) также преобразуется к типу char и результат имеет тип char;

• во время операции присваивания значение правой части преобразуется к типу левой части, который и становится типом результата;

• в процессе преобразования int в char лишние старшие 8 бит просто отбрасываются. Кроме того, существует возможность точно указывать требуемый тип данных, к которому необходимо привести некоторую величину (в скобках перед этой величиной). Скобки и имя типа вместе образуют операцию, называемую *приведением типов.*

Например: z=(int)x+(int)y;

### 6.4. ОПЕРАТОРЫ. УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ ЯЗЫКА

Операторы служат основными строительными блоками программы. В языке Си указанием на наличие оператора служит символ «точка с запятой», стоящий в конце него.

Операторы состоят из выражений. Выражение представляет собой объединение операций и операндов. Операндом называется то, над чем выполняется операция.

**Оператор присваивания**

Общий вид оператора присваивания <Имя> = <Выражение>;

*Пример:* int x, у, а;

х=5;

у=х\*2+7;

а=у/4;

Переменные получают значения: х=5, у=17, а=4.

В языке Си разрешается включать присваивания в выражения, т. е. присваивание **может** рассматриваться как операция с учетом старшинства и влияния скобок.

*Пример:* а=(у=(х=5)\*2+7)/4.

В результате переменная х получает значение 5, причем это значение участвует в дальнейших вычислениях. Затем выполняется умножение (5\*2), сложение (10+7) и еще одно присваивание (у=17). Переменная у получает значение 17, после чего производится деление (17/4), результат которого присваивается переменной а.

В языке Си для компактной записи операторов присваивания имеются специальные операции

+=\_=\*= /= %= Так, следующие две записи на языке Си эквивалентны: i = i+2 и i+=2.

*Пример:*

**int x,y;**

х=у=5;

х+=2; /\* х увеличить на 2, результат х=7 \*/ у- =3; /\* у уменьшить на 3. результат у=2 "7 х\*=у; /\* х умножить на у, результат х=14 \*/ .- х/=++у; /\* разделить х на увеличенный у; результат у=3,х= 12/3 \*/

Операция присваивания сама по себе имеет значение (равное значению выражения, стоящего справа от знака "=") и может входить в выражения.

**Оператор if/else**

Общий вид оператора:

If(<выражение>) <оператор1>

else <оператор2>;

Здесь часть "else <оператор2>" является необязательной, можно применять и одиночный оператор

If(<выражение>) <оператор1>;

Вначале вычисляется значение выражения. Оператор выполняется, если значение выражения истинно. Если выражение ложно (его значение равно нулю), и если есть часть с else, то выполняется оператор2.

*Программа 88* (нахождение наибольшего из двух целых чисел а и b)

#include <stdio.h>

main( )

{

int a,b;

printf("Введите первое число - **");** scanf("%d", &a) ;

printf("Введите второе число - "); scanf("%d", &b) ;

if (a==b)

printf("Заданные числа равны.\n");

else

if (a>b)

printf("Первое число больше второго.");

else printf("Второе число больше первого.");

}

При программировании требуется аккуратно различать знаки = и ==, потому что в ряде случаев компилятор не сможет обнаружить ошибки, связанной с неправильным использованием знаков этих операций, что приведет к неверным результатам.

В качестве оператора! может стоять любой оператор, в частности, снова оператор if/else. При этом может возникнуть неоднозначность, если во вложенной последовательности операторов if/else часть else опускается. Else всегда соответствует ближайшему предыдущему if, не содержащему else.

Например, в конструкции

if(n>0)

if(a>b) z=a;

else z=b;

else относится к внутреннему if. Если требуется отнести else к внешнему if, то необходимо использовать фигурныескобки:

if(n>0)

{

if(a>b) z=a;

} else z=b;

Часто приходится осуществлять выбор более чем из двух вариантов. Чтобы учесть это, конструкция if/else расширяется конструкцией else/if. Распространенный способ выбора по значению из нескольких вариантов:

If (<выражение1>) <оператор1>

else if <выражение2>) <оператор2>

else if (<выражение3>) <оператор3>

else <oпepaтop>;

Выражения просматриваются последовательно сверху вниз; как только какое-то выражение становится истинным, выполняется следующий за ним оператор, и на этом вся цепочка заканчивается. Последняя часть else,как и раньше, может быть опущена.

В языке Си имеется компактный способ записи одного из видов оператора if/else. Он называется «условным выражением» или «тернарной операцией». Такое выражение выглядит в общем виде так:

В1?В2:ВЗ

Сначала вычисляется значение выражения В1. Если оно отлично от нуля (истинно), то вычисляется значение выражения В2, которое и становится значением условного выражения. В противном случае вычисляется значение выражения ВЗ, и оно становится значением условного выражения.

Условное выражение удобно использовать в тех случаях, когда имеется некоторая переменная, которой можно присвоить одно из двух возможных значений. Типичными примерами являются присваивание переменной значения большей из двух величин:

max = (a>b)?a:b;

и нахождение абсолютногозначения числа:

х = (у<0)?-у : у;

**Оператор-переключатель switch**

В тех случаях, когда в программе необходимо произвести выбор одногоиз нескольких вариантов, удобно применять оператор switch. Его синтаксис:

switch (<выражение>)

{

case <константа1>: <список операторов1>;

case <константа2>: <список операторов2>;

…

case <константаN>: <список операторовN>;

default: <список операторов>;

}

Оператор-переключатель выполняется следующим образом. Вычисляется значение выражения в скобках, приведенного после ключевого слова switch, затем программа просматривает список меток, указанных после слов case, до тех пор, пока не находит ту, которая соответствует данному значению. Далее программа переходит к выполнению оператора, расположенного в этой строке. Если подходящей метки не найдется и если существует строка с меткой default:, то будет выполнятся оператор, помеченный этой меткой. В противном случае произойдет переход к оператору, расположенному за оператором switch.

Метки, имеющиеся в операторе switch, должны быть константами или константными выражениями (выражениями, операнды которого константы) целого или символьного типа. Запрещается использовать в качестве метки переменную. Значением выражения в скобках должна быть величина целого или символьного типа. Список операторов варианта может быть либо пустым, либо заканчиваться одним из операторов завершения (break, continue, goto, return). Если у вариантов нет общих частей, то рекомендуется каждый вариант завершать оператором break.

*Программа 89*

#include **<**stdio.h**>**

main ()

{

int **c;**

printf("Введите цифру от 1 до 7:");

c=getchar() ;

printf("\nСоответствующий день недели:");

switch (с)

{

case '1': (printf("\nПонедельник!");break;}

case **'**2': {printf("\nВторник!");break;}

case '3': (printf("\nСреда!");break;}

case '4': {printf("\nЧетверг!");break;}

case '5': (printf("\nПятница!");break;)

case '6': {printf("\nСуббота!");break;}

default:printf("\nBocкpeceньe!") ;

}

}

Если не использовать оператор завершения, то по окончании выполнения списка операторов выбранного варианта произойдет переход на следующий вариант из списка.

**Оператор цикла for**

Оператор

for (<оператор1>;<выражение>;<оператор2>) <оператор3>;

позволяет организовать повторяющийся вычислительный процесс и называется оператором цикла. Как правило, оператор1 и оператор2 являются операторами присваивания или обращениями к функции, а выражение1 - условным выражением.

Цикл for удобно использовать в тех случаях, когда заранее известно количество повторений тела цикла, или имеется явно выраженная переменная, управляющая циклом. В этом случае выражение1 вычисляется один раз и задает инициализацию управляющей переменной.

Выражение1 является условием завершения цикла, а оператор2 задает приращение управляющей переменной.

Например, следующая конструкция

for (i=l; i<n; i++) <оператор>;

является широко распространенной и позволяет «перебрать» первые п натуральных чисел. Из первой строки цикла for можно сразу узнать всю информацию о параметрах цикла: начальное значение переменной i, ее конечное значение, а также насколько увеличивается значение переменной i при каждом выполнении тела цикла.

Любой из операторов и выражений в цикле for может быть опущен, хотя точка с запятой при этом должна оставаться. Если отсутствует оператор1 или оператор2, то он просто выпадает из вычислений. Если же отсутствует выражение1, то считается, что оно всегда истинно.

Например,цикл

for (i=l ;;i++) {

…

}

является бесконечным.

Существуют разнообразные возможности применения цикла for:

1) можно применять операцию уменьшения для счета в порядке убывания вместо счета в порядке возрастания

for(n=10;n>0;n-)

printf("%d \n",n);

2) при желании можно вести счет двойками, десятками и т.д.

for (n=2;n<60;n+=2)

printf("%d\n",n);

3) можно вести подсчет при помощи символов, а не только чисел

for(ch='a';ch<='z';ch++)

printf("Beличинa кода ASCII для %с равна %d.\n",ch,ch);

4) можно проверить выполнение некоторого произвольного условия, отличного от условия, налагаемого на число итераций

for(num=l;num\*num\*num<216;num++)

5) можно сделать так, чтобы значение некоторой величины возрастало не в арифметической, а в геометрической прогрессии

for(n=l;n<1500;n\*=3)

printf("-%d \n",n);

6) в качестве третьего выражения можно использовать любое правильно составленное выражение; какое бы выражение вы ни указали, его значение будет меняться при каждой итерации

for(x= I ;у<=75;у=5\*(х++)+10)

printf("%d, %d\n",x,y);

7) можно даже опустить одно или более выражений (но при этом нельзя опускать символы ";"); необходимо только включить в тело цикла несколько операторов, которые в конце концов приведет к завершению его работы

ans=2;

for(n=3;ans<=25;)

ans=ans\*n;

Тело цикла for(;;)

printfC"\* **\**n");

будет выполняться бесконечное число раз, поскольку пустое условие всегда считается истинным;

8) первое выражение не обязательно должно инициализировать переменную; вместо этого, например, там мог бы стоять оператор printf() некоторого специального вида; необходимо помнить, что первое выражение вычисляется только один раз перед тем, как остальные части цикла начнут выполняться

for(printf("3апоминайте введенные числа!\n");num = 6;)

scanf('"%d",&num);

printf("ЭTO как раз то, что я хочу!\n");

в этом фрагменте первое сообщение оказывается выведенным на печать один раз, а затем осуществляется прием вводимых чисел до тех пор, пока не поступит число 6;

9) параметры, входящие в выражения, находящиеся в спецификации цикла, можно изменить при выполнении операций в теле цикла; предположим, например, что есть цикл со спецификацией следующего вида:

for(n=l;n< l000;n+=delta)

и если после нескольких итераций программа решает, что величина параметраdaltaслишком мала или велика, оператор if внутри цикла может изменить значение этого параметра (в диалоговой программе пользователь может изменитьэтот параметр в процессе выполнения цикла).

В качестве оператора в теле цикла также может быть цикл. На количество вложений циклов друг в друга ограничений нет.

В спецификацию цикла for может быть включено несколько инициализирующих или корректирующих выражений. Например, два вложенных цикла можно записать двумя различными способами:

1. for(i=l;i<10;i++)

for(j=l;j<10;j++)

<оператор>

1. for(i=l,j=l;i<10,j<10;i++,j++)

<оператор>

В нижеследующей программе переменные х, у, z изменяются параллельно.

*Программа 90*

# include<stdio.h>

main()

{

int x,y,z,v,u,zero();

for(x=l,y=l,z=l;x<10;x++,y++,z++)

printf("x=%d y=%d z=%d\n", х, y, z);

}

Результат работы программы:

x=l y=l z=l

x=2 y=2 z=2

x=3 y=3 z=3

x=4 y=4 z=4

x=5 y=5 z=5

x=6 y=6 z=6

x=7 y=7 z=7

x=8 y=8 z=8

x=9 y=9 z=9

**Оператор цикла while**

В общем виде цикл while записывается так:

while (<выражение>) <оператор>;

Цикл while является «условным» циклом, использующим предусловие (т.е. условие на входе). Он называется условным, потому что выполнение оператора зависит от истинности условия, описываемого с помощью выражения. .

Если выражение «истинно» (или в общем случае не равно нулю), то оператор, входящий в цикл while, выполняется один раз, а затем выражение проверяется снова. Эта последовательность действий, состоящая из проверки и выполнения оператора, периодически повторяется до тех пор, пока выражение не станет ложным (или в общем случае равным нулю). После этого управление передается оператору, следующему за оператором цикла while.

При построении цикла while необходимо включить в него какие-то конструкции, изменяющие величину проверяемого выражения так, чтобы в конце концов оно стало ложным. В противном случае выполнение цикла никогда не завершится.

***Пример*** *1.* Алгоритм Евклида.

*Программа 91*

#include<stdio.h>

main()

{

int x,y;

scanf("\n%d",&x); scanf("\n%d",&y) ;

while (x!=y)

{

if (x>y) x=x-y; else y=y-x;

)

printf("\n%d",x);

)

***Пример*** *2.* Проверить, содержит ли квадрат натурального числа n цифру 3.

*Программа 92*

#include<stdio.h>

main()

{

int i,n,a;

scanf("%d",&n); n=n\*n; i=10000;

while(i>=l)

{

a=n/i; /\* если результат целочисленного деления n/i

меньше 1,то а=0 \*/

n=n-a\*i;

if(a==3) goto abc; else i=i/10;

)

printf("цифры 3 в числе n^2 нет");

goto cd;

abc: printf("цифра 3 в числе n^2 есть");

cd: ;

}

**Оператор цикла do/while**

В языке Си имеется также конструкция цикла с постусловием (условие на выходе), где истинность условия проверяется после выполнения каждой итерации цикла. Этот подход реализуется с помощью цикла do/while.

Тело цикла do/while всегда выполняется по крайней мере один раз, поскольку проверка условия осуществляется только после его завершения.

Форма записи:

do <оператор>

while (<выражение>);

**Оператор break**

Оператор break дает возможность выйти из операторов цикла for, while, do/while, а также из переключателя switch без проверки условия. Оператор break приводит к немедленному выходу из самого внутреннего охватывающего его цикла или из переключателя.

Оператор break может использоваться для выхода из цикла в тех случаях, когда заданы два разных условия прекращения его работы.

*Программа 93*

# include<stdio.h>

main()

{

int x=l,y,z;

printf("Мы будем вычислять значение функции y=2\*x+z\n");

printf("Введите значение z:"); scant("%d",&z);

while(x<1000)

(

y=2\*x+z;x++; if y=100 ( printf(= 100\n"); break; }

} if y=100

printf("Цикл закончен!!!") ;

else printf ("Цикл закопчен!!! Но y<>lOO.");

}

Наличие оператора break позволяет использовать «бесконечные циклы». В ряде случаев весьма удобно записать заголовок цикла без всякого условия в виде

for(;;),

а выход из цикла оформить с помощью операторов break.

**Оператор продолжения continue**

Оператор continue вызывает преждевременное завершение выполнения тела цикла и переход к следующему шагу цикла. Оператор continue действует только на самый внутренний цикл, частью которого он является.

*Программа 94*

#include<stdio.h>

main()

(

int x,y,z;

printf("Мы будем вычислять значение функции y=2\*x+z\n");

printf("в промежутках [1;6] и [13;18].");

printf("Введите значение z*= "};* scant("%d",&z) ;

for(x=l;x<18;x++)

{

if ((x>6) and (x<13))

continue;

y=2\*x+z; printf(= %d, у= %d",x,y);

}

}

**Оператор безусловного перехода goto**

Оператор перехода предназначен для безусловной передачи управления в заданную точку программы. Его выполнение заключается в передаче управления оператору, помеченному заданной меткой.

В качестве метки используется идентификатор. Метка отделяется от оператора, к которому она относится, двоеточием. Помеченный оператор должен находиться в пределах той же функции, что и оператор goto. Может быть помечен любой оператор, но никакие два разных оператора не должны иметь одинаковые метки (внутри одной и той же функции). Кроме того, один оператор может быть помечен несколькими метками.

Форма:

goto <метка>;...

<метка>: <оператор>

**Составные операторы и блоки**

Символы "{" и "}" используются для объединения описаний и операторов в составной оператор или блок, так что все конструкции, заключенные в фигурные скобки, оказываются синтаксически эквивалентными одному оператору. Точка с запятой никогда не ставится после первой фигурной скобки, которая завершает блок.

Составной оператор - последовательность операторов, заключенная в фигурные скобки (при необходимости его используют в качестве «внутреннего» оператора в операторах ветвления и цикла). Он называется также «блоком». Блоки не могут быть вложенными.

*Пример /\** I -й вариант \*/

*Программа 95*

#include<stdio.h>

main()

{

int sam,index=0;

/\* В цикл включен только оператор присваивания. Печать данных \*/ /\* будет произведена только один раз - после завершения цикла \*/

while(index++<10)

sam=10\*index+2;

printf("sum= %d\n",sam) ;

)

Результат работы программы:

sam = 102

/\* II -й вариант \*/

*Программа 96*

#include<stdio.h>

main ()

(

int sam,index=0;

/\* Наличие фигурных скобок гарантирует, что оба оператора \*/

/\* являются частью цикла while, и печать результатов будет \*/

/\* производиться на каждом шаге работы цикла. Весь составной\*/

/\* оператор рассматривается как один оператор, являющийся \*/

/\* составной частью оператора while. \*/

while(index++<10)

{

sam=10\*index+2; printf("sum = %d\n",sam);

}

}

Результат работы программы:

sam = 12

sam = 22

sam = 32

sam = 42

sam = 52

sam = 62

sam = 72

sam = 82

sam = 92

sam = 102

***Пустой оператор*** состоит только из точки с запятой (;) и используется для обозначения пустого тела управляющего оператора.

### 

### 6.5. СТРУКТУРА ПРОГРАММЫ НА СИ. ПОНЯТИЕ О ФУНКЦИЯХ

Программа на языке Си представляет собой набор последовательно описанных функций (процедуры и подпрограммы в языке Си считаются частным случаем функций). Каждая функция - самостоятельная единица программы, предназначенная для решения определенной задачи (или подзадачи). При описании она имеет следующий вид:

Тип\_функцни Имя (<список аргументов>)

<описания аргументов>

{

<описания>

<операторы>

}

Отметим, что список аргументов может быть пустым (однако, скобки после имени функции сохраняются). В этом случае, естественно, нет и их описаний.

Имеется одна главная функция (с именем main), с которой начинается выполнение программы. Функции могут обращаться к другим функциям посредством конструкций вызова. Вызов функции используется при вычислении значения выражения. В результате вызова функция возвращает вычисленное значение, которое и является значением вызова функции. Попутно функция может преобразовывать значения своих аргументов. Такой результат вызова функции называется побочным эффектом.

В модуле, вызывающем данную функцию, тип возвращаемого ею значения должен быть описан (даже если это неопределенное значение) вместе с описанием переменных.

*Пример.* Пусть необходимо вывести на экран словосочетание «Простая функция без аргументов» 15 раз, используя функции.

*Программа 97*

#include<stdio.h>

main ()

(

int i,print() ;

for (i=l;i<=15;i++) print();

{

print() /\* вызываемая функция без аргументов \*/

)

printf ("Простая функция без аргументов\n»);

}

Аргументы передаются по значению путем копирования в соответствующие (по порядку) параметры, указанные в определении функции. Соответствие аргументов и параметров по количеств) и типу не контролируется в языке Си. Такой контроль может быть выполнен с помощью дополнительных средств отладки.

Следует различать формальные аргументы, используемые при описании функций, и фактические, задаваемые при вызове функций. Формальный аргумент - переменная в вызываемой программе, а фактический аргумент - конкретное значение, присвоенное этой переменной вызывающей программой. Фактический аргумент может быть константой, переменной или даже более сложным выражением. Независимо от типа фактического аргумента он вначале вычисляется, а затем его величина (в данном случае некоторое целое число) передается функции (см. программу 97). Можно задавать список аргументов, разделенных запятыми. *Пример:* программа 98, которая находит числа х, у, z, принадлежащие отрезку [1;20] и удовлетворяющие условию x^2 = у^2 + z^2*.*

*Программа 98*

#include<stdio.h>

main()

(

int х, у, z, zero() ;

char p, q, ch() ;

x=2; y=45; z=0;

q=’o’;

printf("х» "); zero(x);

printf("x+y+(x+y)^z= "); zero(x+y+(x+y)^z) ;

printf("q= "); ch(q);

}

int zero(u)

int u;

(

printf("%d\n",u);

)

char ch(u)

char u;

{

printf("%c\n",u);

)

Результат работы программы:

x=2

x+y+(x+y)^z= 94

q=0

*Программа 99*

#include<stdio.h>

main()

(

int x,y,z;

int zero();

printf("Следующие величины могут быть сторонами прямоугольного треугольника:\n");

for (х=1;х<=20;х++)

for (у=1;у<=20;у++)

for (z=l;z<=20;z++)

if (y\*y+z\*z==x\*x)

zero(x,у,z);

}

int zero(f,g,h)

int f,g,h;

(

printf ("x= %d, y= %d, 2=%d\n",f,g,h);

)

Результат работы программы:

следующие величины могут быть сторонами прямоугольного треугольника

х= 5, у= 3, z= 4

х= 5, у= 4, *z=* 3 x= 10, y= 6, z= 8

x= 10, y=8, z=6 x= 13, y=5, z= 12

x= 13, y= 12, z= 5 x*=* 15, y= 9, z= 12

x= 15, y= 12, z=9 x=17, y=8, z=15 x= 17, y= 15,

z=8 x=20, y=12, z=16

x=20, y= 16, z= 12

Завершает выполнение данной функции и передает управление вызывающей функции ***оператор return****; в* главной функции main он же вызывает завершение выполнения всей программы. Оператор return может содержать любое выражение:

return (<выражение>);

Если выражение не пусто, то вычисляется его значение, которое и становится значением данного вызова функции.

Достижение «конца» функции (правой закрывающей фигурной скобки) эквивалентно выполнению оператора return без возвращаемого значения (т.е. оператор return в конце функции может быть опущен).

*Пример.* Данная программа вычисляет факториал, если число меньше 8; если вводимое число больше или равно 8, то выводится сообщение «Очень большое число».

*Программа 100*

#include<stdio.h>

main()

{

int n, s() ;

printf("Введите число ");

scant("%d", &n) ;

if (n<8) printf(=%d", s(n)) ;

else printf("Очень большое число");

)

int s(x) /\* определение функции с параметром \*/

int x;

{

int y,p=l;

for (y=x; y>0; y- ) p\*=y;

return(p); /\* Возвращает в основную программу значение р \*/

}

Результат работы программы:

1. Введите число 4

р=24

2.Введите число 9

Очень большое число

*Пример:* предположим, что нужно вычислить x2 (для некоторого неотрицательного целого у) (очевидный способ реализации возведения в целочисленную степень -многократное умножение, но существует более эффективный алгоритм, приведенный ниже).

*Программа 101*

#include<stdio.h>

main()

(

int а, Ь, x, у, z;

int odd() ;

printf("\nВведите x, у через пробел: ");

scanf("%d %d", &х, &у); а=х; Ь=у; z=l;

while (b!=0)

if (odd(b))

{ z=z\*a; b- -;}

else

( a=a\*a; b=b/2;}

printf("\n%d", z);

}

int odd(t)

int t;

(

return((t%2==0)? 0:1);

)

Результат работы программы:

Введитеx, у через пробел: 15 2

225

Если функции необходимо вернуть несколько значений, можно использовать два различных приема:

• применить глобальные переменные (в этом случае кроме изученных ранее характеристик переменных (имени, типа, значения), используется еще одна – класс памяти, см.ниже);

• применить переменные типа «указатель» в качестве аргументов функции. При вызове функции информация о переменной может передаваться функции в двух видах. Если мы используем форму обращения

function 1(х);

то происходит передача в функцию значения переменной х. Изменение этого значения в вызывающую функцию не возвращается. Если мы используем форму обращения

function2(&x);

то происходит передача адреса переменной х. Теперь измененное значение переменной х может использоваться в вызывающей функции. Первая форма обращения требует, чтобы определение функции включало в себя формальный аргумент того же типа,что и х:

function l(num)

int num;

Вторая форма обращения требует, чтобы определение функции включало в себя формальный аргумент, являющийся указателем на один из объектов соответствующеготипа:

function2(x)

int \*x;

Обычно пользуются первой формой, если входное значение необходимо функции для некоторых вычислений или действий, и второй формой, если функция должна будет изменять значения переменных в вызывающей программе. Выше вторая форма вызова \же применялась при обращении к ф\нкции scanf(). Когда мы хотим ввести некоторое значение в переменную num, мы пишем scanf(''%d",&num). Данная функция читает значение, затем, используя адрес, который ей дается, помещает это значение в память.

*Пример:* пусть необходимо поменять местами заданные значения переменных х и у.

*Программа 102*

#include<stdio.h>

main()

{

int х, у;

int interchange(); /\* описание функции типа int \*/

x=l; y=3;

printf("Имели... x=l y=3\n") ;

interchange(&x, &y); /\* обращение к функции (в данном случае передаются адреса переменных) \*/

printf("Получили... x=%d y=%d", х, у);

}

/\* вызываемая функция \*/

int interchange(u, v)

int \*u, \*v;

(

int p;

p=\*\i; \*u=\*v; \*v=p;

}

Результат работы программы:

Имели х=1 у=3

Получили х=3у=1

В этой программе в вызове функции interchange(&x,&y) вместо передачи значений х и у мы передаем их адреса. Это означает, что формальные аргументы и и v, имеющиеся в спецификации interchanage(u,v), при обращении будут заменены адресами и, следовательно, они должны быть описаны как указатели.

Поскольку х и у целого типа, u и v являются указателями на переменные целого типа, и мы вводим следующее описание:

int \*u,\*v; int p;

Оператор описания используется с целью резервирования памяти. Мы хотим поместить значение переменной х в переменную р, поэтому пишем: р=\*u; Вспомните, что значение переменной u - это &х, поэтому переменная и ссылается на х. Это означает, что операция \*u дает значение х, которое как раз нам и требуется. Мы не должны писать, например, так:

р = u; /\* неправильно \*/

поскольку при этом происходит запоминание адреса переменнойх, а не ее значения.Аналогично, оператор \*u = \*v соответствует оператору х = у.

Тип функции определяется типом возвращаемого ею значения, а не типом ее аргументов. Если указание типа отсутствует, то по умолчанию считается, что функция имеет тип int. Если значения функции не принадлежат типу int, то необходимо указать ее тип в двух местах.

1. Описать тип функции в ее определении:

char pun(ch,n) /\* функция возвращает символ \*/

int n;

char ch;

2. Описать тип функции также в вызывающей программе. Описание функции должно быть проведено наряду с описаниями переменных программы; необходимо только указать скобки (но не аргументы) для идентификации данного объекта как функции.

main()

{

char rch,pun();

### 6.6. КЛАССЫ ПАМЯТИ

Помимо изученных ранее характеристик переменных (имени, типа, значения), в ряде случаев оказывается важной еще одна - класс памяти. Класс памяти характеризует время существования и место хранения объекта в программе.

Для обозначения класса памяти в языке Си используются следующие служебные слова:

auto extern

register static

Автоматические объекты (auto) являются локальными по отношению к блоку и хранятся внутри того блока, где они описаны. Автоматические переменные можно инициализировать произвольными выражениями, включающими константы и ранее описанные переменные и функции.

Автоматические объекты существуют только во время выполнения данного блока и теряют свои значения при выходе из него. При каждом вхождении в блок им присваиваются начальные значения, заданные в описании. Если начальные значения не заданы, то значения автоматических объектов при входе в блок не определены. До сих пор в этом параграфе рассматривались именно автоматические объекты.

Объекты, описанные внутри блока с классом памяти register, называются регистровыми переменными. Они подчиняются всем правилам, касающимся автоматических переменных. Описание register указывает компилятору, что данная переменная будет часто использоваться. Когда это возможно, переменные, описанные как register, располагаются в машинных регистрах. Это приводит к меньшим по размерам и более быстрым программам.

Объекты, описанные внутри функции с добавлением класса памяти extern или описанные вне функции без оказания класса памяти, относятся к внешним объектам. Внешние объекты хранятся вне любой функции, входящей в состав программы, и существуют в течение выполнения всей программы. Они могут быть использованы для передачи значений между различными, в том числе и отдельно компилируемыми, функциями. Сами функции также являются внешними объектами, поскольку правила языка Си не позволяют определять одни функции внутри других. Внешние переменные можно инициализировать только выражениями с константами и указателями на ранее описанные объекты. По умолчанию (если не задана инициализация) внешние объекты получают нулевые начальные значения.

Внешние объекты делятся на внешние глобальные и внешние статические.

Важно различать описание внешнего объекта и его определение. Описание указывает свойства объекта (тип, размеры и т.д.); определение же вызывает еще и отведение памяти установке начального значения, если используется инициализация.

Например, появившиеся вне определения какой-либо функции строчки

int max;

char save[maxline];

определяют внешние переменные max и save, вызывают отведение для них места в памяти и служат в качестве описания для остальной части этого файла. В тожевремя строчки

extern int max;

extern char save[];

описывают в остальной части данного блока переменную max как int, a save как массив типа char (размеры которого указаны в другом месте), но не создают переменных и не отводят для них места в памяти.

Во всех блоках, составляющих исходную программу, должно содержаться только одно определение внешней переменной; другие блоки могут содержать описания extern для доступа к ней.

*Программа 103*

#include<stdio.h>

int i=0;

/\* Класс памяти переменной - внешний. Область действия переменной -любая программа, \*/

/\* загружающаяся с данным файлом. Время существования *i=0 -* все время выполнения программы. \*/

main() /\* Блок уровня 1. \*/

(

auto int i=l;

/\* В блоке 1 область действия *i=l -* функция main(). Время \*/

/\* существования i=l - все время выполнения главной функции /\*

/\* main(). /\*

printf("%d\n", i) ;

/\* Если две переменные имеют одно и то же имя, то по этому \*/

/\* имени обращаются к внутренней переменной, внешняя \*/

/\* непосредственно недоступна, поэтому после выполнения \*/

/\* блока 1 программа напечатает i=l. \*/

{ /\* Блок уровня 2. \*/

int i=2;

/\* Класс памяти переменной i=2 - auto. Область \*/

/\* действия i=2 - блок 2, время существования - время \*/

/\* существования блока 2. блок 2, время существования -\*/

/\* время существования блока 2. \*/

printf("%d\n", i) ;

/\* Программа напечатает i=2. \*/

{ /\* Блок уровня 3. \*/

i+=l; printf("%d\n", i);

/\* Печатается самая внутренняя переменная с именем i,/\*

/\* которая после выполнения операции данного блока \*/

/\* становится равной 3. \*/

}

/\* Возвращение к блоку уровня 2. \*/

printf("%d\n", i) ;

/\* Опять печатается i=3. \*/

)

/\* Возвращение к блоку уровня 1. \*/

printf("%d", i) ;

/\* Переменная i=3 исчезает. Теперь самой внутренней переменной \*/

/\* с именем i будет i=l. \*/

)

*Программа 104*

#include<stdio.h>

int a;

main()

(

extern int a;

int P ();

a=6; P();

)

int P()

(

extern int a;

printf("a=%d",a);

}

Результат работы программы:

a=6

Областью действия статического внешнего объекта является модуль, в котором описан данный объект. При этом где-либо во внешних описаниях (т.е. вне определения функций) должно быть расположено определение внешнего статического объекта в виде

static <Спецификация типа> <Спецификация данных>;

На основании определения под объект отводится память и может быть произведена инициализация. Статические переменные можно инициализировать только выражениями с константами и с указателями на ранее описанные объекты.

При первом входе в соответствующую локальную область (блок или модуль) статические переменные инициализируются один раз (по умолчанию - нулем). При последующих входах в данную область статические переменные получают те значения, которые они имели при последнем выходе из области.

По умолчанию все функции данного модуля, расположенныениже определениястатического объекта, включаются в его область действия - в них не обязательно дополнительно описывать объект для получения к нему доступа. Функции, определения которых расположены в модуле выше определения внешнего статического объекта, для пол\чения доступа к нему должны содержать описание этого объекта с классом памяти extern.

Константы являются объектами статического класса памяти.

Функция может быть определена как статический внешний объект. В этом случае она будет доступной в любой точке данного мод\ля и не доступной за пределами модуля.

*Программа 105*

#include<stdio.h>

main()

(

int count;

int trystat () ;

for (count=l; count<=3; count++)

(

printf ("Итерация %d:\n", count);

trystat() ;

}

)

trystat ()

{

int fade=l;

static int stay=l;

printf("fade = %d и stay = %d\n", fade++, stay++) ;

}

Результат работы программы:

Итерация 1:

fade = 1 и stay = 1

Итерация 2:

fade = 1 и stay = 2

Итерация 3:

fade = 1 и stay = 3

Если мы лишь немного видоизменим в программе функцию trystat()

trystat()

{

int fade=l;

int stay=l;

printf("fade = %d и stay = %d\n", fade++, stay++);

}

то получим следующий результат:

Итерация 1:

fade = 1 и stay = 1

Итерация 2:

fade = 1 и stay = 1

Итерация 3:

fade = 1 и stay = 1

### 

### 6.7. ФУНКЦИИ ВВОДA-ВЫВОДА

Средства ввода-вывода не являются составной частью языка Си. Имеется ряд библиотечных функций Си. обеспечивающих стандартною систему ввода-вывода для программ на Си. Макроопределения, описания переменных и определения этих функций содержатся в файле стандартных заголовков stdio h Поэтому, как указывалось выше, каждая пользовательская программа должна содержать в начале ссылку

#include <stdio.h>.

В примерах программ мы неоднократно использовали форматные функции ввода **(scanf())** и вывода **(printf())**. Набор стандартных функций ввода и вывода значительно шире и включает большое число функций для работы с данными различного типа, различными устройствами, буферизованного и небуферизованного, форматного и бесформатного ввода и вывода.

Система ввода-вывода Си обеспечивает некоторый уровень абстракции между программистом и используемым устройством. Эта абстракция называется потоком, а фактическое устройство ввода-вывода называется файлом. Буферизованная файловая система преобразует каждое физическое устройство в логическое устройство,называемое потоком. Существуют потоки двух типов: текстовые и двоичные.

Текстовый поток - это последовательность символов, которая организуется в строки, завершающиеся символами новой строки. Обработка текстового потока предполагает преобразование данных из текстового (внешнего) представления в машинное (внутреннее) или наоборот. При обработке двоичного потока последовательность его байтов взаимно однозначно соответствует байтам во внешнем устройстве.

В языке программирования Си файл - это логическое понятие, которое система может относить к чему угодно (от дисковых файлов до терминалов). Поток связывается с конкретным файлом выполнением операции «открыть» Как только файл открывается, можно обмениваться информацией между ним и программой. Закрытие выводимого потока заставляет ЭВМ записывать содержимое этого потока на внешнее устройство Этот процесс обычно называется промыванием потока. В начале выполнения программы ЭВМ открывает три предопределенных текстовых потока stdin, stdout и stderr, связанных со стандартными устройствами ввода-вывода (консоль - клавиатура и дисплей). Допускается переадресация ввода-вывода к другим устройствам.

Простейшими функциями консольного ввода-вывод являются функция **getche(),** которая читает символ с клавиатуры, и функция **putchar()**, которая печатает символ на экране. Функция getche() ждет, пока не будет нажата клавиша, а затем возвращает ее значение, автоматически выдавая на экран «эхо» нажимаемой клавиши. Функция putchar() записывает ее символьный аргумент на экран в текущуюпозицию курсора.

Ниже приводится пример простой программы, которая принимает один символ с клавиатуры и выводит его на экран.

*Программа 106*

#include<stdio.h>

main()

(

char ch;

ch *=* getchar() ;

putchar(ch);

)

Есть две важные версии функции getche(). Первая – **getchar()** - буферирует ввод до тех пор, пока не введен возврат каретки. Второй версией является функция **getch()**, которая работает точно так же, как getchar(), за исключением того, что getch() не возвращает на экран эхо введенного символа.

Функции **gets()** и **puts()** позволяют читать и писать цепочки символов (строки) с консоли. Функция gets() читает цепочку символов, которая вводится с клавиатуры (ввод оканчивается возвратом каретки), помещает ее с адреса, который указывает ее аргумент - указатель символа. Функция puts() выводит на экран ее аргумент - цепочку символов, а затем символ новой строки. Например, нижеследующая программа читает цепочку в массив str и тут же печатает ее.

*Программа 107*

main ()

(

char str[80] ;

gets (str) ;

puts(str) ;

)

Функция puts() занимает меньше памяти и работает быстрее, чем printf() при выводе символьных цепочек, поэтому программисты часто используют функцию puts() в тех случаях, когда важно иметь высоко минимизированныйкод.

Таблица 3.4

**Некоторые функции буферизованной сисгемы ввода-вывода**

|  |  |
| --- | --- |
| Имя | Функция |
| fopen()  fclose()  putc()  getc()  fseek()  fprintf()  fscanfl()  feof()  ferror()  rewind()  remove() | Открывает поток  Закрывает поток  Выводит символ в поток  Вводит символ из потока  Ищет указанный байт в потоке  Форматный вывод в поток  Форматный ввод из потока  Возвращает истину, если достигается метка EOF (конец файла)  Возвращает истину, если встретилась ошибка  Устанавливает начальную позицию файла  Стирает файл |

Для работы с файлами в Си используются функции буферизованной системы ввода-вывода, табл. 3.4. Обращение к ним использует указатель файла, который определяет различные характеристики файла, включая его имя, статус и текущую позицию; используется связанным с этим файлом потоком для привязки каждой функции буферированного ввода-вывода к месту, над которым она выполняет свои операции. Указатель файла является переменной типа FILE, которая определяется в файле заголовковstdio.h.

Функция **fopen()** вызывается так:

fореn(<имя\_файла>, <режим>);

Имя файла должно быть цепочкой символов, которая составляет правильное имя файла для операционной системы и может включать спецификацию пути. Режим задает желаемый статус открытия, табл.3.5.

Таблица 3.5

**Значения режимов в Турбо-Си**

|  |  |
| --- | --- |
| Режим | Смысл |
| "r"  "w"  "а"  "r+"  "w+"  ''а+" | Открыть файл для чтения  Создать файл для записи  Добавлять в файл  Открыть файл для чтения/записи  Создать файл для чтения/записи  Открыть или создать файл для чтения/записи |

Например, для того чтобы открыть файл с именем test для записи, можно написать

fp = fopen("test", "w");

где fp -переменная типа FILE\*. Переменная fp является указателем файла.

Следующий оператор обнаруживает любую ошибку при открытии файла, такую, как, например, попытку открыть защищенный от записи диск или заполненный диск, прежде чем состоится попытка записи на него:

if((fp = fopen("tesf, "w"))==NULL)

{

рuts("Нельзя открыть файл!\n");

exit(l);

}

NULL - это макро, которое определяется в файле заголовка stdio.h.

Функция **putc()** в виде

рuts(<символ>, fp);

используется для записи символа в поток, который предварительно открыт для записи с помощью функции fopen(); fp - возвращаемый функцией fopen() указатель файла.

Функция **getc()** в виде

getc(fp)

используется для чтения символов, которые она возвращает из потока, открытого в режиме чтения функцией fopen(). fp является указателем файла типа FILE, который возвращается функцией fopen(). В тех случаях, когда достигается конец файла, функция getc() возвращает маркер его конца EOF. Например, для чтения текстового файла до маркера конца файла можно использовать следующие операторы:

ch = setc(fp);

while(ch!=EOF)

{

ch = getc(fp);

}

Функция **feof()** использует аргумент указателя-файла и возвращает 1, если достигнут конец файла, или 0, если не достигнут. Например, приведенная ниже программа читает двоичный файл до тех пор, пока ЭВМ не встречает маркер конца файла:

while(!feof(fp)) ch = getc(fp);

Функцию **fdose()** используют для закрытия потока, который был открыт с помощью функции foреn(). Все потоки необходимо закрыть перед завершением программы. Аргументом функции является указатель файла, который закрывается.

Функции foреn(), getc(), putc() и fdose() составляют минимальный набор функций ввода-вывода. Простым примером использования функций putc(), foреn() и fdose() является программа, которая приведена ниже. Эта программа просто читает символы с клавиатуры и записывает иx в дисковый файл до тех пор, пока не введен знак $. Имя выходного файла задается из командной строки. Например, если вы назовете программу ktod («клавиша - на диск»), то набор на клавиатуре ktod test будет позволять вам вводить строки текста в файл с именем test.

*Программа 108*

#include .h"

main(argc,argv) /\*ktod - клавиша на диск \*/

int argc;

char \*argv[];

(

FILE \*fp;

char ch;

if(argc!=2)

{

printf("Bы забыли ввести имя файла\n);

exit(l);

)

if((fp=fopen(argv[l], "w"))== NULL)

(

printf("He может открыть файл\n);

exit(l);

}

do

(

ch = getchar();

putc(ch, fp);

)

while (ch!='s');

fclose (fp) ;

}

Еще одним примером является программа **dtos**, которая будет читать любой ASCII файл и выводить его содержимое на экран.

*Программа 109*

#include "studio.h"

main (argc, argv) /\*dtos-wicK на экран\*/

int argc;

char \*argv[] ;

(

FILE \*fp;

char ch;

if(argc!=2) {

printf("Вы забыли ввести имя файла\n"};

exit(l);

}

if((fp=fopen(argv[l], "r"))==NOLL)(

printfC'He может открыть файл\n"};

exit(l);

}

ch=getc(fp); /\* читать один символ \*/

while(ch!=EOF)

{

putchar(ch); /\* печать на экран \*/

ch=getc(fp);

}

fclose(fp) ;

}

Под управлением буферизованной системы ввода-вывода можно выполнять операции чтения и записи с произвольным доступом с помощью функции **fseek()**, которая устанавливает файловую позицию.

Например, для чтения 234-го байта в файле с именем test можно использовать следующую функцию:

funcl()

{

FILE \*fp;

if((fp=fopen("test" ,"r"))==NULL)

{

printf("He могу открыть фаил\n");

exit(l);

}

fseek(fp,234L,0);

return getc(fp); /\*читать один символ в 234-й позиции\*/

}

В дополнение к рассмотренным до сих пор основным функциям ввода-вывода буферизованная система ввода-вывода включает функции **fprintf()** и **fscanf()**. Эти функции ведут себя точно так же, как функции printf() и scanf(), за исключением того, что они работают с дисковыми файлами. Обращения к функциям fprintf() и fscanf() имеют следующий вид:

fprintf(1р,<формат>,<список аргументов>);

fscanf(1р,<формат>,<список аргументов>);

где fp является файловым указателем, который возвращается вызовом функции fopen().

### 6.8. ДИРЕКТИВЫ ПРЕПРОЦЕССОРА

Препроцессор - это программа, которая производит некоторые, иногда весьма значительные, манипуляции с первоначальным текстом программы перед тем, как он подвергается трансляции.

Оператор препроцессора - это одна строка исходного текста, начинающаяся с символа #, за которым следуют название оператора и операнды.

Операторы препроцессора могут появляться в любом месте программыи их действие распространяется на весь исходный файл.

Весьма часто используют следующие операторыпрепроцессора:

#include

#define

Более специфичными являются директивы #pragma, #if, #error и др.

Важная возможность препроцессора - включение в исходный текст содержимого других файлов. Эта возможность, в основном, используется для того, чтобы снабжать программы какими-то общими для всех данными, определениями.

Например, чрезвычайно часто в начале программы на языке Си встречается препроцессорная конструкция

#include <stdio.h>

Когда исходный текст программы обрабатывается препроцессором, на место этой инструкции ставится содержимое расположенного в некоем стандартном месте файла stdio.h, содержащего макроопределения и объявления данных, необходимых для работы функций из стандартной библиотеки ввода-вывода. Для использования различных математических функций необходимо подключать файл описаний math.h. Функции, оперирующие со строками, описаны в файле string.h, функции общего назначения - в stdlib.h, функции даты и времени - в time.h, диагностика - в assert, h и т.д.

Директива #define позволяет дать в программе макроопределения (или задать макросы). Оператор макроопределения имеет вид

#define <макроимя> <строка лексем> или

#define <макроимя(<список параметров>)> <строка лексем>

Макроимя - это идентификатор. Строка лексем - это последовательность лексем от Макроимени до конца строки. Точка с запятой в конце макроопределения не ставится.

Препроцессорная обработка макроопределения сводится к тому, что любое появление Макроимени (макровызов) в качестве отдельной лексемы в тексте программы, расположенном после макроопределения, ведет к замене этого Макроимени на указанную Строку лексем.

Например, прочитав определения

#defmePI3.14159

#defineE2.711828

препроцессор заменит в программе все имена PI и Е на соответствующие числовые константы.

Препроцессор языка Си позволяет переопределять не только константы, но и целые программные конструкции. Например, можно написать определение

#define forever for(;;)

и затем всюду писать бесконечные циклы в виде forever

Определение макроимени с параметрами имеет некоторую специфику. Список параметров макроимени - это список идентификаторов, разделенных запятыми. Следующая после списка параметров строка лексем также может содержать эти параметры, которые при макровызове будут заменены на соответствующие аргументы.

Макровызов должен быть отдельной лексемой и состоять из макроимени и заключенного в круглые скобки списка аргументов. При обработке макровызова препроцессор заменяет каждый параметр в строке лексем на соответствующий аргумент макровызова.

В следующих программах иллюстрируются некоторые применения операторов препроцессора.

*Пример 1:* найти большее из двух чисел.

*Программа 110*

#include<stdio.h>

#define MAX(X,Y) ((X)>(Y) ? (X) : (Y))

main()

{

int x,y;

scanf ("%d %d", &x, &y); printf ("%d", MAX(x, y) );

)

Результат работы программы:

1. 5

5

*Пример 2.*

*Программа 111*

#include<stdio.h>

#define S (x) x\*x

#define P(x) printf("x равен %d.\n",x)

main()

{

int x=4;

int z ;

z = S(x); P(z); z = 3(2);

P(z);

P(S(x));

P(S(x+2));

P(100/S(2));

P(S(++x)) ;

}

Результат работы программы:

x равен16.

x равен 4.

x равен 16.

x равен 14.

x равен 100.

x равен 30.

Оператор препроцессора #pragma позволяет записывать самые различные указания компилятору (зависящие от конкретного компилятора). Например, следующие два предложения препроцессора

#pragma recursive

#pragma nonrec

устанавливают режим всех функций программы по умолчанию рекурсивным или нерекурсивным. Указание препроцессора

#pragma optimize time

воспринимается компилятором таким образом, что он старается сгенерировать объектный код, отличающийся более высокой скоростью выполнения, чем в случае, когда он должен быть более компактным.

### 6.9. СИ И ПАСКАЛЬ

При знакомстве с языком Си, особенно после изучения Паскаля и Бейсика, погружение в детали его изобразительных средств может затушевать важную мысль: хотя на Си можно написать практически любую прикладную программу, он изначально для этого не предназначен. Си является результатом эволюционного развития языков создания системных программных средств. Если в прикладном программировании эволюция шла от Фортрана к Алголу, Коболу, Паскалю и т.д., то в системном - от Ассемблеров, привязанных к архитектуре ЭВМ, к Си, для которого созданы трансляторы, делающие его хоть и независимым от архитектуры, но не меняющим основного предназначения.

С помощью Си можно сделать то, что на Паскале сделать невозможно (или почти невозможно) - например, написать фрагмент операционной системы (или новую операционную систему), утилиты и т.п. Так, ряд трансляторов с Паскаля написаны на Си; обратное невозможно представить. В то же время, не раз отмечалось, что прикладные программы, написанные на Паскале, отличаются большей надежностью, чем написанные на Си; их легче читать, передавать от одного программиста другому для совершенствования и сопровождения. Это связано с тем, что Паскаль содержит существенно больше ограничений и является языком более высокого уровня с сильной типизацией данных. Для языка же. который предназначен для разработки системного программного обеспечения, чем меньше ограничений, тем лучше; так, в Си возможны неявные преобразования всех базовых типов данных и указателей друг в друга, что крайне желательно при создании системных средств, но при невнимательности программиста приводит к ошибкам, не улавливаемым транслятором с Си (Паскаль же подобные недопустимые операции пресекает немедленно).

Разумеется, сказанное выше не следует абсолютизировать. Программисты, привыкшие к Си, успешно пишут на нем программы различных классов. Это касается не только Си - вспомните об экспертных системах, написанных на Бейсике. В то же время, при массовом программировании придерживаться «разделения труда» между языками представляется более естественным.