1.История языка С.

Первая версия языка Си была разработана в середине 60 ых годов для разработки операционной системы Unix в лаборатории Bell. Один из самых первых разработчиков этого языка, Кен Томсон (Ken Thompson), решил, что требуется язык для создания более сложных языков программирования. Он создал такой язык и назвал его «B». В процессе развития своего творения Томсон постоянно боролся с ограничением ресурсов памяти, что теперь очень похоже на встраиваемые системы. Деннис Ричи (Dennis Ritchie) решил расширить язык «B» свойством генерировать малый по объему код, который сможет соперничать с кодом, написанным на ассемблере. В 1973 году важнейшие свойства этого нового языка «C» были получены.

Возрастающая популярность Си заложена в его переносимости. Компиляторы Си были созданы для многих платформ (так в сообществе разработчиков называют процессорное ядро МК), отчего его популярность еще больше выросла. Наиболее бурно Си стал использоваться в 80 годах, когда стал основным языком для создания программ персональных компьютеров.

Американский национальный институт стандартизации (American National Standards Institute — ANSI) в 1982 году учредил комитет X3J11 для разработки стандарта языка Си. В 1989 доклад комитета был передан в Международную организации стандартизации (International Organization for Standardization — ISO) и международную электротехническую комиссию (International Electrotechnical Commission — IEC) и был утвержден в качестве стандарта ISO/IEC 9899-1990. За этим стандартом последовало неизбежное развитие языка, которое было узаконено в 1999 стандартом ISO/IEC 9899. И Си стал языком, который наиболее часто используется в компьютерной индустрии.

2.Алгоритм и св-ва алгоритма.

Алгоритм - точное предписание исполнителю совершить определенную последовательность действий для достижения поставленной цели за конечное число шагов.

Свойства алгоритма:

• ***Дискретность*** (прерывность, раздельность) – алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов. Каждое действие, предусмотренное алгоритмом, исполняется только после того, как закончилось исполнение предыдущего.

• ***Определенность*** – каждое правило алгоритма должно быть четким, однозначным и не оставлять места для произвола. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

• ***Результативность (конечность)*** – алгоритм должен приводить к решению задачи за конечное число шагов.

• ***Массовость*** – алгоритм решения задачи разрабатывается в общем виде, то есть, он должен быть применим для некоторого класса задач, различающихся только исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.

На основании этих свойств иногда дается определение алгоритма,
например: *“Алгоритм – это последовательность математических, логических или вместе взятых операций, отличающихся детерменированностью, массовостью, направленностью и приводящая к решению всех задач данного класса за конечное число шагов”*.

3.Типы данных: целые числа.

Целые типы отличаются объемом памяти и наличием знака. Применение разных типов данных позволяет оптимизировать затраты памяти.

Основной целочисленный тип в языке С – int (integer). Объем выделяемой памяти в языке не фиксирован. Объем зависит от компьютера и компилятора.

На основе типа инт формируются другие целые типы. Для этого используют спецификатор, и они изменяют диапазон и наличие знака. Short, long, signed, unsigned

Размер памяти short<=int=>long

signed – тип со знаком unsigned – без знака

short=short int=signed short int

long=long int=signed long int

4.Типы данных: ВЕЩЕСТВЕННЫЕ ЧИСЛА

Вещественные числа имеют дробные части (3.14159) и экспоненты (2.579x10**24). Они также известны как числа с плавающей точкой.

Хранятся в виде двух частей:мантисса+порядок

Мантисса – значащие цифры ,порядок -это степень

Например: $0.0034 = 3.4*10^{-3}$

3.4- мантисса 10^{-3}- порядок

Кроме значащих цифр также хранится знак мантиссы и знак порядка.

Вещественные числа всегда со знаком.

Есть 3 варианта вещественных типов:float – вещ. Тип одинарн. 4 байта, double – 8 байт, long double – 10 байт

Тип void скорее декоративный тип. Используются:1.Для указания типа ф-ии, которая не возвращает значение. 2.Пустой список аргументов ф-ии. 3. Базовый тип указателей 4.Приведение типов

5. Типы данных: символы и логические значения.

Логический тип bool 1 байт (true=1, false=0)

Символьный тип char 1 байт. Переменная типа char хранит числовой код 1-го символа.

Коды формируются на основе таблицы ASCII (256 символов).

Тип char считается целочисленным типом. В языке С символ записывается в одинарных кавычках!

6.СИМВОЛЬНЫЕ ЧИСЛОВЫЕ И СТРОКОВЫЕ КОНСТАНТЫ

Константа — это ограниченная последовательность символов алфавита
языка, представляющая собой изображение фиксированного (неизменяемого) объекта.
Константы бывают числовые, символьные и строковые. Числовые константы делятся на целочисленные и вещественные.

Целочисленные константы

Целочисленные данные в языке Си могут быть представлены в одной из следующих систем счисления:
1.Десятичные:Последовательность цифр (0 — 9), которая начинаются с цифры, отличной от нуля. Пример: 1, -29, 385. Исключение - 0
2.Восьмиричные:Последовательность цифр (0 — 7), которая всегда начинается с 0. Пример: 00, 071, -052, -03.
3.Шестнадцатиричные:Последовательность шестнадцатеричных цифр (0 — 9 и A — F), которой предшествует присутствует 0x. Пример:
Двоичная система представления данных непосредственно в языке Си не поддерживается. Однако можно воспользоваться файлом binary.h, в котором определены двоичные константы в пределах байта.

Вещественные константы

Константа с плавающей точкой (вещественная константа) всегда представляется числом с плавающей точкой двойной точности, т. е. как имеющая тип double, и состоит из следующих частей:

- целой части — последовательности цифр;
- точки - разделителя целой и дробной части;
- дробной части — последовательности цифр;
- символа экспоненты e или E;
- экспоненты в виде целой константы (может быть со знаком).

Любая часть (но не обе сразу) из нижеследующих пар может быть опущена:

- целая или дробная часть;
- точка или символ e (E) и экспонента в виде целой константы.

Примеры вещественных констант

345.
3.14159
2.1E5
.123E3
4037e-5

По умолчанию компилятор присваивает вещественному числу тип double. Если программиста не устраивает тип, который компилятор приписывает константе, то тип можно явно указать в записи константы с помощью следующих суффиксов: F (или f) — float для простых вещественных констант, L (или l) — long double для вещественных констант двойной расширенной точности.

Примеры:

· 3.14159F — константа типа float, занимающая 4 байта;
· 3.14L — константа типа long double, занимающая 10 байт.

Символьные константы

Символьная константа — это один символ, например: 'z'. В качестве символьных констант также могут использоваться управляющие коды, не имеющие графического представления. При этом код управляющего символа начинается с символа '\' (обратный слеш).

Строковые константы

Строковая константа — это последовательность символов, заключенная в кавычки, например:
"Это строковая константа"
Кавычки не входят в строку, а лишь ограничивают её. Технически строковая константа представляет собой массив символов, и по этому признаку может быть отнесена к разряду сложных объектов языка Си.
В конце каждой строковой константы компилятор помещает '\0' (нуль-символ), чтобы программе было возможно определить конец строки. Такое представление означает, что размер строковой константы не ограничен каким-либо пределом, но для определения длины строковой константы её нужно полностью просмотреть.
Поскольку строковая константа состоит из символов, то она имеет тип char.
Количество ячеек памяти, необходимое для хранения строковой константы на 1 больше количества символов в ней (1 байт используется для хранения нуль-символа).
Символьная константа 'x' и строка из одного символа "x" - не одно и то же.
Символьная константа - это символ, используемый для числового представления буквы x, а строковая константа "x"содержит символ 'x' и нуль-символ '\0' и занимает в памяти 2 байта. Если в программе строковые константы записаны одна за другой через разделители, то при выполнении программы они будут размещаться в последовательных ячейках памяти.

7.Структура программы

В начале программы находятся команды, которые подключают другие файлы. Эти команды называются предпроцессорные директивы. #include. Может быть несколько разных include с указанием разных подключаемых файлов по необходимости. Любая программа на языке С должна содержать функцию.
Функция: имя ( ). Обязательная функция в программе называется main ( )(в скобках аргументы). В скобках после названия ф-ии могут присутствовать, так называемые, аргументы ф-ии main, они используются для передачи данных из командной строки при запуске программы. Если они не используются, то скобки могут быть пустыми. Перед именем ф-ии пишется тип результата, который получается после работы ф-ии.
int тип результата main ( ) имя ф-ии int – целочисленный тип данных
В языке С для вычисления блока операторов используются фигурные скобки. { }
Все операторы ф-ии должны помещаться только внутри скобок. Поскольку ф-ия возвращает результат внутри ф-ии должен быть оператор return 0;
Выполнение оператора return приводит к завершению работы ф-ии, операторы после не выполняются.
Минимальная программа на языке С

```
#include “stdafx.h”
int main ( )
{
return 0;}
```

Программа = переменные(хранение данных) + операторы (команды, которые изменяют данные)
После объявления переменной ее значение известно.

10.УСЛОВНЫЙ ОПЕРАТОР И УСЛОВНАЯ ОПЕРАЦИЯ

Условный оператор предназначен для выбора одного из двух альтернативных действий и имеет следующую форму записи:
условный оператор =
"if" "("выражение")" оператор_1 ["else" оператор_2]
При выполнении условного оператора вначале вычисляется <выражение>. Если его значение не равно нулю, то выполняется <оператор_1>, а <оператор_2> пропускается. Если значение выражения равно нулю, то выполняется <оператор_2>, а <оператор_1> пропускается.

8.ОПЕРАТОРЫ ЦИКЛА

Цикл-это команда ,которая обеспечивает многократный повтор некоторых действий .

Для корректного выполнения цикла необходимо соблюдать след. Правила, которые гарантируют выполнение цикла требуемое количество раз и его завершение.

В языке С существуют три оператора цикла. Циклы отличаются формой записи . любой цикл можно преобразовать в любую форму.

While- цикл с предусловием

While (условие) {тело цикла}

Если тело цикла состоит из одного оператора ,то {}не обязательны

Если в теле > одного оператора, то {} обязательны.

Пример:

```
Int a=0;I
While (a<10)
A=a+1
```

До начала цикла значение переменной цикла должно быть таким, чтобы цикл мог начать свое выполнение, то есть чтобы условие цикла оказалось истинным , иначе цикл не будет выполняться.

ПРАВИЛА ВЫПОЛНЕНИЯ ЦИКЛА

1.Обычно в цикле используется переменная, которая присутствует в условии цикла , такая переменная называется переменная цифра

2.a<10

Условие цикла должно быть таким, чтобы он мог завершиться через некоторое количество шагов

Внутри цикла переменная цикла должна менять свое значение, таким образом чтобы на некотором шаге условие стало ложным и цикл завершился.

FOR

В языке си цикл for аналогичен по способу работы циклу while, отличается в лучшую сторону компактной записью

Пример:

```
For (int =0;i<10;i++)
{}
Int  a; int b;
For(a=0;a<10;a++)
B=a+10
```

DO WHILE С ПОСТУСЛОВИЕМ

Постусловие-условие цикла проверяется после выполнения действий , если условие истинно, то продолжается ,если ложно, то заканчивается. Таким образом этот цикл выполняется хотя бы один раз.**Пример:**

```
Do
{тело цикла}
While (условие)
Int a=0,b;
Do
{b=a+10;
A++;}
```

9.Операторы перехода: goto, break. Continue, return.

goto, break. Continue, return изменяют порядок выполнения программы. Их применение не очень желательно потому что затрудняется текст выполнения программы.

Break;

I цикл в этом случаем break завершает текущий цикл. В цикле такие операторы используются при наступлении некоторого события, после которого выполнение цикла должно завершать событие опред. Через условный оператор.

II цикл заканчивает каждый пункт case.

Пример цикла с break: Найти сумму всех эл-ов массива до первого отрицательного элемента.

```
int a [10]={7,2,3,4,6,9,17,10,8,16,1};
int n=10; s=0
for (int i=0; i<n; i++)
if(a[i]>0)
s=s+a[i];
else break;
```

Continue прекращает выполнение текущей операции цикла и выполнение перехода к началу цикла. Н-р: посчитать кол-во четных чисел в массиве

```
int a[10]={…}
int n=10, count=0;
for(int i=0;i<n;i++)
{if(a[i]%d!=0)
continue I <>
count++;}
```

Счетчик count увеличивается только, если число четное.

Go to оператор перехода на метку. Позволяет перейти в любую точку программы и выше и ниже по тексту. Метка – это идентификатор, указывающий на какую строчку надо перейти.

```
int x=1;
loop 1;
x++;
if(x<10) go to loop 1;
```

go to не рекомендуется использовать. Применение в редких случаях. Н-р для перехода из глубоко вложен. Циклах.

11.Оператор множественного выбора

Применяется для выбора одного действия из набора.

switch (переключатель)(выражение)
{case конст1:оператор п1 break;
case конст2:оператор п2 break;
case конст3:оператор п3 break;
default:оператор п}

Выражение обычно является переменной, значение которой используется для выбора пункта. Тип это переменной должен совпадать с типом констант. Тип переменной констант можетможет быть либо целым, либо символьным. Все константы должны иметь разные значения, но один тип данных. Значение переменной последовательно сравнивается с каждой константой до обнаружения совпадения. После совпадения выполняются операторы по соотв. Пункту. В конце каждого блока операторов по case должен стоять оператор break.

Break – оператор, который прекращает выполнение текущего блока и передает правление оператору, который следует после switch.

Default – необязательная часть (может и не быть), выполняется, если не было совпадения ни с одной константой.

Switch используется, например, для орг-ии меню в программе.

13.Бинарные операции

Операция – действие, которое включается в выражение. Операции характеризуются знаком, смыслом действия, кол-ом операторов.

Унарные 1 i++
Бинарные 2 a+b
Тенарные 3

Операция преобладает приоритетом, которая показывает порядок выполнения нескольких операций в одном вложении.

Бинарные:
*, /, %, +, -, << >>(побитовый сдвиг влево и вправо)
<, <=, >, >=, ==, !=
&, ^, I (побитовые и, искл. Или, или)
&& и, II лог. Или =
*=, /=, %=, +=, -=, <<=, >>=, &=, I=, ^=
.(точка) - последовательное вычисление

Операции ++, -- Инкремент +1, декримент-1
Префиксная форма ++x; --x; Сначала значение увелич. На единицу, потомиспользуется.
Постфиксная ф-ма (имя переменной, знак операции) x++; x--; Знач-ие переменной исп., а потом увелич. На единицу.

12.УНАРНЫЕ ОПЕРАЦИИ

++, --, sizeof,~,!,-,+,&
*-разадресация(взятие адреса)
New,delete-выделение памяти, освобождение памяти в с++
Операции ++ --
Инкремент +1, декримент -1
Префиксная форма
++x ;--x;
Сначала значение увеличивается на еденицу , потом используется
Постфиксная форма (имя переменной , знак операции)
x++;x--; значение переменной используется, а потом увеличивается на еденицу
sizeof-определяет размер объекта или типа в байтах
sizeof y=sizeofx; выражение
sizeof y=sizeof(int); тип
&-унарная операция , взятие адреса переменной

```
Int a,* pint;
Pint=&a;
```

&-переменная ,* разадресация(разименовывание)
New,delete-операции для работы с динамической памятью в с++
Поразрядные операции (побитовые)
(&,|,^,~)
Поразрядные операции применяются к целочисленным операторам и работают с их двоичной формой
&-и
|-или
^-искл .или
~-побитовое отрицание не

20.Указатели и массивы. Работа с элементами массива через указатель

Указатели- переменная , которая содержит адрес другой переменной. Если переменная содержит адрес некоторого другого элемента , то говорят, что переменная указывает на этот элемент

Указатель имеет тип данных , по типу определяют сколько байт адресуется указателю

Указатели используют для работы с массивами и динамич. Структурами данных

Динамич. Структуры данных означают , что память выделяется спец. Командой во время работы программы

Массив- индексы a[i], указатели*(a+i)

Имя массива это указатель константы на первый элемент

Если константа, то значение менять нельзя

Действие с указателями в массиве выполняются быстрее, чем с указателями.

```
Int a[10],*i,k;
K=a[2]
i=a+2;
k=*(a+3);//k=a[3];
```

14.ПРИВЕДЕНИЕ ТИПОВ

Приведе́ние ти́па (*type conversion*) — преобразование значения переменной одного типа в значение другого типа. Выделяют*явное* и *неявное* приведения типов.

При явном приведении указывается тип переменной, к которому необходимо преобразовать исходную переменную.

При неявном приведении преобразование происходит автоматически, по правилам, заложенным в данном языке программирования.

Также в языке могут быть заданы специальные функции для приведения

Неявное приведение типа

Само приведение происходит как во время присваивания значения переменной, так и при операциях сравнения, вычисления выражения. При использовании в выражении несколько различных типов значения одного или нескольких подтипов может быть осуществлено преобразование к более общему типу (*супертипу*), с большим диапазоном возможных значений.

В языке Си:

```
double  d;  // вещественный тип
long    l;  // целый тип
int     i;  // целый тип
if (d > i)      d = i;
if (i > l)      l = i;
if (d == l)     d *= 2;
```

Каждый раз при выполнени операции сравнения или присваивания переменные разных типов будут приведены к единому типу. Следует с осторожностью использовать неявное приведение типа. При переводе числа из вещественного типа в целочисленный, дробная часть откидывается. Обратное приведение из целочисленного типа к вещественному также может привести к понижению точности, что связано с различным представлением вещественных и целочисленных чисел на машинном уровне. К примеру, вещественный тип *single* стандарта IEEE 754 не может точно представить число 16777217, в то время как 32-битный целочисленный тип может. Это может привести к ситуациям, когда сравнение на равенство одного и того же числа, представленного типами (*int* и *single*) будет выдавать ложный результат (числа не равны друг другу).

Явное приведение типа

В языке С

Для явного приведения типов некоторой переменной перед ней следует указать в круглых скобках имя нового типа, например:

int X; int Y = 200; char C = 30; X = (int)C * 10 + Y;//переменная С приведена к типу int.

Если бы в этом примере не было выполнено явное приведение типов, то компилятор предположил бы, что выражение С * 10 + У переменной Х было бы присвоено значение 640, а не корректное 3200. В результате приведения типа переменная С распознается компилятором как 16-ти разрядная, и описанной выше ошибки не возникает

15.Функции ввода-вывода в языках С и С++

Ввод и вывод выполняется через специальные стандартные ф-ии. С(printf() scanf())

Применение Cin, cout. Для использования cin, cout необходимо включить библиотеку

#include<iostream> using namespace std;

вывод cout. Cout может выводить строкиили значения переменных, несколько компонентов разделяются знаком <<

```
cout<<"Helo";
int a=3;
cout <<"a="<<a;
```

Перевод строки cout<<"\n"<<endl;

Ввод cin cin>>a;

Ф-ии scanf и printf. Ф-ии дают возможность взаимодействовать с программой. Ф-ии «работают» во многом одинаково – каждая использует «управляющую строку» и «список аргументов».

Инструкции, передаваемые ф-ии printf зависят от того, какого типа эта переменная. Например, при выводе на печать целого числа применяется формат %d, а при выводе символа %c.

Также, как для ф-ии printf, для ф-ии scanf указывается управляющая строка и следующей за ней список аргументов. Основное различие 2-х этих ф-ии заключается в особенности: ф-ия printf использует имена переменных константы и выражения, в то время как ф-ия scanf – только указатели на переменные.

printf("Summa=%d");

16.Форматирование при выводе. Управляющие последовательности

Форматированный вывод

Функция форматированного вывода printf получает в качестве аргументов строку формат и аргументы, которые необходимо вывести в соответствии с форматом, и возвращает число выведенных символов. В случае ошибки возвращает отрицательное значение и устанавливает значение ferror. Если произошло несколько ошибок, errno равно EILSEQ.

int printf (const char * format, ...);

Функция проходит по строке и заменяет первое вхождение %<спецификатор формата> на первый аргумент, второе вхождение %<спецификатор формата> на второй аргумент и т.д. Далее мы будем просто рассматривать список флагов и примеры использования.

Общий синтаксис спецификатора формата

%[флаги][ширина][.точность][длина]спецификатор

Спецификатор – это самый важный компонент. Он определяет тип переменной и способ её вывода.

Управляющие последовательности представляют собой последовательность символов для представления специальных символьных констант.

При введении управляющая последовательность начинается с символа обратный слеш '\' (обязательный первый символ), затем пишется комбинация латинских букв, либо цифр.

```
#include <stdio.h>
#include <conio.h>
void main() {
   //функция не получает никаких аргументов, кроме строки
   printf("Hello world");
   getch();}
```

'\a'	сигнал-звонок
'\b'	возврат на одну позицию (на один символ)
'\f'	перевод (прогон) страницы
'\n'	перевод строки (переход на новую строку)
'\r'	возврат каретки (курсора) к началу строки
'\t'	горизонтальная табуляция
'\v'	вертикальная табуляция
'\?'	знак вопроса
'\''	апостроф (одиночная кавычка)
'\"'	кавычка (символ двойной кавычки)
'\\'	обратная косая черта
'\ddd'	восьмеричное представление символьной константы, где d – восьмеричная цифра (от 0 до 7)
'\xddd'	шестнадцатеричное представление символьной константы, где d – шестнадцатеричная цифра (от 0 до F)

Последовательность вида '\ddd' и '\xddd' позволяют представить символ из набора кодов ЭВМ как последовательность восьмеричных или шестнадцатеричных цифр соответственно. Например, символ возврата каретки может быть представлен различными способами:

'\r' – общая управляющая последовательность;

'\015' – восьмеричная управляющая последовательность;

'\x00D' – шестнадцатеричная управляющая последовательность.

В языке C++ нет встроенных средств ввода и вывода – они осуществляется с помощью функций, типов и объектов, которые находятся в стандартных библиотеках. Существует два основных способа:

1. форматированный ввод-вывод данных (функции, унаследованные из C);
2. стандартные потоки ввода-вывода данных (объекты C++).

18.ОДНОМЕРНЫЕ И МНОГОМЕРНЫЕ МАССИВЫ

Массивы- это множество элементов , которые имеют одинаковый тип данных. Массив имеет имя, которое одновременно является указателем на первый элемент массива

При объявлении массива создается тип элементов: имя и размер

Одномерный массив

Имя

Тип массив [размер];

Int a[10];a=const

Многомерный массив

Тип имя [размер 1][размер 2][размер 3]

Нумерация начинается <0

A[0],a…

```
Const int n=10;
Void main()
{int a[n];
}
```

Размер массива должен быть целым числом, это должна быть const

Обращение к элементу массива выполняется через index

Нет контроля выхода за границу массива. При объявлении массива возможна его инициализация, т. е. присваивание начальных значений Например:

Int b[4]={10,7,8,9,};

Размер массива рекомендуется задавать через глобальную константу вне функции

Матрицы- двумерные массивы. При объявлении задается два размера : количество строк и столбцов. Размеры рекомендуется задавать через глобальные константы

Тип имя массива [21][r2]

```
Const int n=3;
Const int m=4;
Void main()
{int matz[n][m];
```

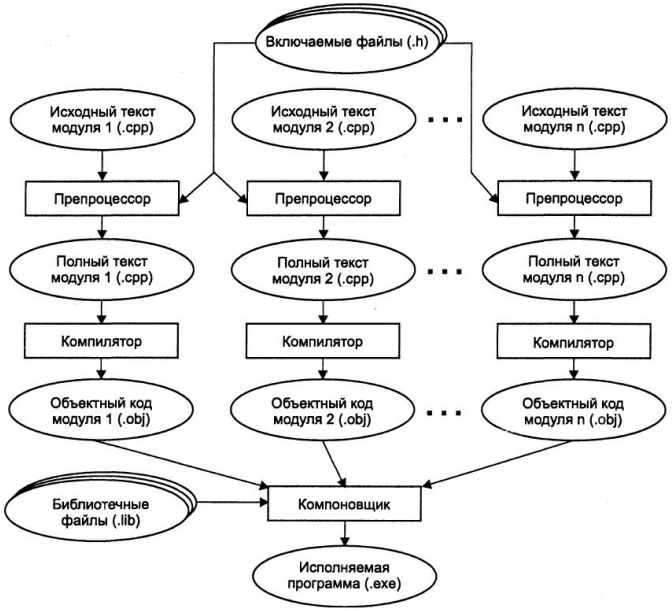
Инизиализация

Int m [][]={{1,1},{0,2},{1,0}};

Int mt [3][2]={1,1,0,2,1,0};

В памяти элементы матрицы располагаются последовательно по строкам.

17.Этапы создания исполняемой программы. Компиляция и интерпритация.



Этапы создания исполняемой программы

Создание исполняемого файла из исходного текста программы предполагает выполнение этапов а и б (компиляции и компоновки).
Исполняемый файл – это файл, который может быть обработан или выполнен компьютером без предварительной трансляции. Обычно исполняемый файл получается в результате компиляции и компоновки объектных модулей и содержит машинные команды и/или команды операционной системы.
Интерпретатор анализирует и тут же выполняет программу покомандно, по мере поступления ее исходного кода на вход интерпретатора.
Алгоритм работы простого интерпретатора:
1) прочитать инструкцию;
2) проанализировать инструкцию и определить соответствующие действия;
3) выполнить соответствующие действия;
4) если не достигнуто условие завершения программы, прочитать следующую инструкцию и перейти к пункту 2.Утверждение «Языковый процессор, который построчно анализирует исходную программу и одновременно выполняет предписанные действия, а не формирует на машинном языке скомпилированную программу, которая выполняется впоследствии» справедливо для интерпретатора

Компиляция в программировании – преобразование программы, представленной на одном из языков программирования, в коды на машинно-ориентированном языке, которые принимаются и исполняются непосредственно процессором. **Результатом компиляции является объектный файл** с необходимыми внешними ссылками для компоновщика. Программа уже переведена в машинные инструкции, однако еще не полностью готова к выполнению.
Компилятор – это программа, предназначенная для трансляции исходного текста программы с высокоуровневого языка в объектный код. Входной информацией для компилятора является описание алгоритма или программа на языке программирования. На выходе компилятора **–** эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код).
Компоновка – это один из этапов создания исполняемого файла.
Компилировать **–** проводить трансляцию машинной программы с проблемно-ориентированного языка на машинно-ориентированный язык (создание объектного кода) для ее исполнения. Результатом компиляции является объектный файл с необходимыми внешними ссылками для компоновщика. Программа уже переведена в машинные инструкции, однако еще не полностью готова к выполнению. В объектном файле имеются ссылки на различные системные функции. Даже если в программе явно не упомянута ни одна функция, необходим, по крайней мере, один вызов системной функции **–** завершение программы и освобождение всех принадлежащих ей ресурсов.
Компоновщик – модуль системы программирования или самостоятельная программа, которая собирает результирующую программу из объектных модулей и стандартных библиотечных модулей. Этот процесс называется компоновкой, его результатом и будет исполняемый файл. С процедурой интерпретации компоновка не связана.
При работе с программами существуют этапы:
а) компиляции
б) компоновки
в) интерпретации
г) исполнения программы.

Режим интерпретации можно использовать при отладке программ на языке высокого уровня.
Интерпретация в разработке программ – процесс непосредственного покомандного выполнения программы без предварительной компиляции, «на лету». В большинстве случаев интерпретируемая программа работает намного медленнее, чем скомпилированная программа, но не требует затрат на компиляцию, что в случае небольших программ может повышать общую производительность. Интерпретация связана с получением переменными значений в процессе работы программы.
Интерпретация и компиляция не позволяют изменить семантику (смысл) языка программирования. Обычно **при описании семантики** в рамках операционного подхода исполнение конструкций языка программирования трактуется с помощью некоторой воображаемой (абстрактной) ЭВМ.
Интерпретация и компиляция не позволяют изменить **синтаксис языка** – набор правил построения фраз алгоритмического языка, позволяющий определить осмысленные предложения в этом языке.

19.Указатели: объявление, операции, инициализации.

Указатель – переменная, которая содержит адрес другой переменной. Адрес – большое число, № ячейки в памяти. Указатель имеет тип данных, по типу определяется сколько байт адресуется указателю.

Объявление указателя:

тип * имя;

Операции с указателями:&-указатель получения адреса

int a; x=&a;

* - разадресация (значение по адресу)

*x=15;

++, -- в этих операциях указатель изменяется на то кол-во байт, которое соотв. Типу указателя.

Арифм. Действия с указателем имеют смысл при работе с массивами через указатель.

Нельзя!указатели +, *, /, побитовые операции

Можно указатели -, ук + const

int *p; p=?

*p++=10;

*,++ имеют одинаковый приоритет, поэтому выполняются справа на лево. Однако инкремент имеет постфиксную форму (p++), т.е. сначала берется значение, а потом выполняется увелечение.

Т.о. порядок будет след.: по адресу пишем 10(присваивание), потом увеличиваем указатель.

(*p)++; - значение по адресу увеличить на 1

int a, *pa, b;

a=7; b=5;

pa=&a;

b=*pa+10

a=*pa+a;

Для обозначения пустого указателя часто используется const pa=NULL;

Указатели используются для работы с массивами и динамическими структурами данных.

Динамические структуры данных означают, что память выделяется специальной командой во время работы программы.

C - функции malloc( ) free( )

C++ - операции nen delete

Инициализация – присваивания начальных значений переменной.

21.Работа с текстовыми файлами

Файлы используются для постоянного хранения информации. Переменная находится в оперативной памяти только во время работы программы, а файлы хранятся на жестком диске. Для работы с файлом требуется указатель на файл.

FILE *pfile; - библиотечный тип, который описывает конструкцию для доступа к файлу.

Для работы с файлом используется набор библиотечной функции.

stdio.h

std.afx(visual studio)

1.Обявить указатель на файл

2.Открыть файл в open_s( )

pfile=fopen(“t.txt”, “r”);(старый вариант)

r-read – открытие файла для чтения (можно читать, нельзя писать, файл должен существовать)

w-write – режим записи (если файла нет, он будет создан, если файл есть, то старое содержимое будет потеряно)

a-append – добавление, запись в конец существующего файла

fopen_s(&pfile, “t1.txt”, “r”); (новый вариант)

3.Рекомендуется проверить успешно ли открылся файл

if(pfile==NULL)

printf(“ошибка”)

else//обработка

Для работы с файлами его необходимо прочитать с винчестера и записать в оперативную память. Поскольку файл может быть большого размера, чтение выполняется фрагментами(блоками). Блоки файла записываются в участок памяти, который называется буфер. При открытии файла в зависимости от режима открытия указатель файла устанавливается на начало или на конец. Позиция указателя автоматически изменяется после каждой операции ввода/вывода(чтения записи). Позиция меняется на некоторое кол-во символо, которое прочитано или записано.

4.Цикл чтения из файла

чтение выполняется в цикле до тех пор пока не встретился символ конца файла.

while(!feof()pfile)

{…..}

5.Чтение из файла

fgets(s,100, pfile);

Читает указанное число символов, пока не встретился символ перевода строки или конец файла. В конец строки дописывается ‘\0’ символ

a=fgets(pfile);

6.После завершения работы с файлами желательно его закрыть

fclose(pfile);

Особенно важно, если была запись.

23.Способы передачи параметров в функцию (по значению, по адресу)

передача по адресу. Функция получает адреса фактических параметров.

Адрес тоже самое, что и указатель. Номер ячейки памяти

Через адреса функция может возвращаться к исходным значениям и их изменять.

Если функция должна возвращать несколько результатов, то используют передачу по адресу.

Так же передачу по адресу используют если переменная имеет большой размер

Передача параметров по значению

При передаче параметра вызываемой процедуры или функции по значению, изменение значения формального параметра внутри процедуры или функции никак не влияет на фактический параметр, передаваемый при вызове процедуры или функции. Указание, что параметр нужно передавать по значению, осуществляется с помощью ключевого слова Знач.

Пример:

```
Процедура Процедура2(Знач ФормальныйПараметр1)
    ФормальныйПараметр1 = 2 * 3;
КонецПроцедуры
ФактическийПараметр = 10;
Процедура2(ФактическийПараметр);
// будет выведено значение 10 - изменение формального
// параметра внутри процедуры НЕ повлияло на изменение
// значения фактического параметра, переданного при
// вызове процедуры: параметр передан по значению!
Сообщить (Фактический Параметр);
```

22.Организация подпрограмм(функции).

функция – относительно самостоятельная часть программы, которая выполняет некоторую задачу . функцию можно использовать несколько раз с разными данными . функция является подпрограммой

функции- подпрограмма ,которая возвращает значение, то есть после ее работы формируется какой то результат, который является решением задачи

процедуры- подпрограммы, которые не возвращают значения. Вызывается отдельным оператором

в языке с нет синтаксической разницы между функцией и процедурой, любая подпрограмма функция

оформление функции:

функция состоит из заголовка. Часто заголовок функции помещают отдельно от тела функции и помещают вначале программы перед main

тогда полный текст функции заголовок и тело размещаются после main

в любой программе на с должна присутствовать функция main . когда начинает работать программа управление передается функции main. Вызов остальных функций выполняется из main

если функция организована как функция, она должна возвращать значения и внутри не должен присутствовать оператор return

тип возвращения значения должен соответствовать типу результата в заголовке функции

return завершает работу функции после него не выполняются действия. Обычно return последний оператор в функции. Однако может быть несколько return в одной функции с применением if

если функция как процедура, то тип результата void, внутри функции нет return оператор return может вернуть только одну переменную , чтобы получить от функции несколько ответов нежно использовать передачу параметров функции через указатель

вызов функции- то же самое что и использование . вызов означает, что данная функция будет работать с теми данными ,которые указаны как ее параметры . вызовов может быть несколько с разными данными

прототип функции должен располагаться раньше чем вызов

ВЫПОЛЕНИЕ ПРОГРАММЫ С ФУНКЦИЯМИ

После запуска программы начинается выполняться функция main. Перед тем как функция будет выполняться, в оперативной памяти выделяется место для хранения переменных функции. Если в тексте функции main присутствует вызов др. функции , то при его выполнении работа функции main останавливается, выделяется память под переменные новой функции и ей передается управление, начинают выполнять операторы новой функции. Если в функции присутствует return , то значение в операторе return передается в точку вызова. После этого функция завершает свою работу и память, которая была выделена для ее переменных ,освобождается

Если в функции нет return ,то выполняются все операторы до конца и так же завершается работа функции и освобождается память, управление передается в точку вызова функции. Дальше продолжает выполняться функция main . она присутствует в памяти все время выполнения программы . завершение функции main означает завершение программы

24.Рекурсивные функции

Функция, которая вызывает сама себя, называется *рекурсивной функцией*.

Рекурсия - вызов функции из самой функции.

Пример рекурсивной функции - функция вычисления факториала.

```
#include <stdio.h>
int fact(int num) {
 if(num==1)   return(1);
 else return(num*fact(num-1));  // рекурсивный вызов
}
int main() {
 int a, r;
 printf("a= ");
 scanf("%d",&a);
 r = fact(a);
 printf("%d! = %d",a,r);
 getchar();getchar();
 return(0);
}
```

29.Переименование типов (typedef)

Для удобства и понятности программы, можно задать типу новое имя с помощью ключевого слова typedef:

```
typedef тип новое_имя [ размерность ] ;
```

Квадратные скобки - элементом синтаксиса. Размерность может отсутствовать.

Примеры:

Переопределение

```
typedef unsigned int UINT:
typedef char Msg[100];
typedef struct{
char fio[30];
int date, code;
double salary;} Worker;
```

Использование вместо стандартных:

```
UINT i , j ; // переменные типа unsigned int
Msg str[10]; // массив из 10 строк по 100 символов
Worker staff[100]; // массив из 100 структур
```

25.Структуры:объявление, операции, инициализация, способы доступа к полю.

Структуры относятся к типам данных, которые формируют пользователи
Struct
Union(объединение)
Enumeration(перечисление)
Структура-конструкция, которая объединяет несколько переменных производственных типов. Части структуры называются –поля
Struct имя типа структуры
{ тип поле 1;
Тип поле 2;
Тип поле 3;
} список переменных;
Комментарий :может отсутствовать тип структуры или список переменных
Struct {int A;}x;
Struct name
{int A;}x;
Тип у поля может быть любым кроме типа самой структуры
Но тип поля может быть указателем, на тип структуры
Struct name {name *A;};
Используется при работе с динамическими структурами данных, например со списками
Массив –набор элементов одного типа
Структура-набор элементов разного типа
Доступ к полю:
X- переменная структура
X.A
* через указатель
Struct name
{int A;}*pName;
pName-A=10;
указатель-поле
инициализация
name vasin ={19};
переменную одного структурного типа, можно присваивать
name Va, Za;
Va.A=19;
Za=Va;
Массив структур позволяет хранить данные

```
Name stud [10];
```

Пример:

```
Struct student
{char name [20];
Char family[20];
Int ball[3];};
Student table [3];
Table [1].name [0]
```

```
Int tablecount- индекс последнего записанного элемента
Tablecount=-1 нет данных в таблице
For(int i=0,i<n;i++)//по массиву структур
For(int j=0;j<3;j++)//по баллам
If (table[i]. ball[j]==5)
Cout<<table[i].family;
```

26.Битовые поля

Битовые поля в си объявляются с помощью структур. Они позволяют получать доступ до отдельных битов или групп битов. Доступ до отдельных битов можно осуществлять и с помощью битовых операций, но использование битовых полей часто упрощает понимание программы.

Синтаксис объявления битового поля

```
struct <имя> {
        <тип> <имя>: <размер>;
        ...
}
```

Размер структуры, содержащей битовые поля, всегда кратен 8. То есть, если одно поле содержит 5 бит, а второе 4, то второе поле начинается с восьмого бита и три бита остаются неиспользованными.

Неименованное поле может иметь нулевой размер. В этом случае следующее за ним поле смещается так, чтобы добрать до 8 бит.

Если же адрес поля уже кратен 8 битам, то нулевое поле не добавит сдвига.

Кроме того, если имеются обычные поля и битовые поля, то первое битовое поле будет сдвинуто так, чтобы добрать до 8 бит.

27.Перечисление(enum)

В практике программирования встречаются случаи, когда выражение может принимать значения только из заранее определенного конечного множества значений. Для таких задач язык С предоставляет программисту способ задания именованных целочисленных констант: enum (перечисление).

Вместо того, чтобы помнить и использовать в тексте программы числовые значения, можно один раз сопоставить каждой константе имя и далее пользоваться этими именами, а компилятор сам в каждом нужном месте просто подставит значение соответствующей константы.

enum [имя_пользовательского_типа] {список_именованных_констант};

28.ОБЪЕДИНЕНИЯ

Объединение-это еще агрегатный тип данных ,который программист формирует из совокупности других типов- базовых или ранее определенных пользовательских.

Основные действия с объединениями(объявление объединения, копирование экземпляров, передача в качестве параметра в функцию и т.д) внешне выглядят так же, как аналогичные действия со структурами

Например, объявление объединения выглядит так:

Union B{ //ключевое слово union говорит компилятору о том, что далее следует объявление пользовательского типа данных, этот тип назван именем