1. ***Арифметические операции (23шт). Оператор присваивания, как бинарный оператор, его значение*.**

4 унарных оператора:

**+ и –** (для себя: не путать со сложением и вычитанием – это знак числа)

**++ и –** инкрементирование, декрементирование

3 условных:

**&&** И

**||** ИЛИ

**!** НЕ

5 бинарных:

**+ - \* / %**

6 операторов присваивания:

**=** (x=y) присваивание

**+=** (x+=y) сложение с присваиванием

**-=** (x-=y) вычитание с присваиванием

**\*=**(х\*=у) умножение с присваиванием

**/=** (х/=у) деление с присваиванием

**%=** (х%=у) деление с остатком с присваиванием

6 для работы с битами:

**&** арифметическое и

**|** арифметическое или

**~** арифметическое не

**^** арифметическое исключающее или

**<<k** сдвиг влево на к разрядов

**>>k** сдвиг вправо на к разрядов

6 операций отношения:

**==** эквивалентно

**!=** не равно

**<** меньше

**>** Больше

**<=** меньше или равно

**>=** больше или равно

Операция присваивания обозначается символом = и выполняется в 2 этапа:

* вычисляется выражение в правой части;
* результат присваивается операнду, стоящему в левой части.

Он является бинарным, так как операция выполняется над двумя объектами.

В случае если объекты в левой и правой части операции присваивания имеют разные типы используется операция явного приведения типа.

1. ***Бессылочная реализация списков. Определение переменных, необходимых для реализации. Определение функции добавления элемента в бессылочный список.***

Бессылочная реализация списков – реализация с помощью 2 стеков – в одном стеке храним начало списка, в другом – конец списка. Место между 2 стеками – текущее положение.  Текущий элемент списка лежит на вершине первого стека. При необходимости переместиться к следующему элементу, значение вершины второго стека извлекается и помещается на вершину первого стека. При необходимости переместиться к предыдущему элементу, значение вершины первого стека извлекается и помещается на вершину второго стека.

Переменные, необходимые для реализации:

CStack <T> begin, end (2 стэка)

Функция добавления элемента в конец списка:

void operator<<(const T&x){end<<x};

1. ***В каком случае допустимо использование #include "mysuperlib.h"? В каком случае допустимо использование #include <mysuperlib.h>?***

#include "mysuperlib.h":

**" "** используется для включения файлов, написанных и определенных самим пользователем. При использовании двойных кавычек (**" "**) препроцессор получает доступ к текущей директории, в которой находится исходный "mysuperlib.h". Это избавляет пользователя от многократной записи определенной функции. После записи пользовательский файл может быть импортирован в любое место программы с помощью препроцессора #include.

#include <mysuperlib.h>:

**< >** используется для включения системных заголовочных файлов, расположенных в стандартных системных каталогах - файлов, которые содержат C/C++ объявления функций и макроопределения для совместного использования между несколькими исходными файлами. При импорте файла с использованием угловых скобок(**< >**) препроцессор использует предопределенный путь к каталогу для доступа к файлу.

1. ***Дано вещественное число x . Пользуясь только операцией умножения и присваивания, получить в переменные y и z значения x в степени 5 и x в степени 13 соответственно за пять умножений.***

#include <iostream>

using namespace std;

int main()

{

int x, y, z;

cin>>x;

y=**x\*x**;

z=**y\*x**;

y=**y\*z**;

z=**y\*y\*z**;

cout<<"x^5="<<y<< endl;

cout<<"x^13="<<z;

return 0;

}

1. ***Двунаправленные списки. Описание типов, необходимых для реализации (два типа). Определение одной из функции списка.***

Двунаправленный список – это структура данных, состоящая из последовательности элементов, каждый из которых содержит информационную часть и два указателя на соседние элементы. При этом два соседних элемента должны содержать взаимные ссылки друг на друга.

В таком списке каждый элемент (кроме первого и последнего) связан с предыдущим и следующим за ним элементами. Каждый элемент двунаправленного списка имеет два поля с указателями: одно поле содержит ссылку на следующий элемент, другое поле – ссылку на предыдущий элемент и третье поле – информационное. Наличие ссылок на следующее звено и на предыдущее позволяет двигаться по списку от каждого звена в любом направлении: от звена к концу списка или от звена к началу списка, поэтому такой список называют двунаправленным.

Типы, необходимые для реализации:

***typedef struct SList2\_***

***{***

***int value;***

***struct SItem\_ \*prev, \*next;***

***}SList2;***

здесь данные будут храниться в члене структуры ***value***, а ссылки, соответственно, на предыдущий/следующий объекты представлены указателями ***prev*** и ***next***. Признаком того, что данный элемент является первым в списке, может служить нулевое значение указателя ***prev***, а признаком конца списка может служить нулевое значение указателя ***next.***

Для работы со списком следует определить два указателя: указатель на головной элемент списка и указатель на текущий элемент списка:

***TList2 \*head=NULL, \*current=NULL;***

Признаком пустоты списка служит нулевое значение указателя ***head.*** Установка текущего элемента на начало списка сводится к присвоению ***current=head;***

Вставка элемента со значением ***value*** после текущего может быть произведена следующим образом:

***TList2 \*InsertData(int value, TList2 \*&head, TList2 \*&current)***

***{***

***TList2 \*New=( TList2 \*)malloc(sizeof(TList2));***

***New->value = value;***

***if(head == NULL) //Случай пустого списка***

***{current =head = New; head->next= head->prev=NULL; return New;}***

***if(current==NULL) //Случай вставки в начало списка***

***{***

***current=New; current->next=head; current->prev=NULL;***

***head->prev=New;***

***head=New;***

***return New;***

***}***

***New->next = current->next; New->prev=current;***

***if(current->next != NULL) current->next->prev = New;***

***current ->next = New;***

***return New;***

***}***

Вызывать эту функцию надо следующим образом

***InsertData( value, head, current);***

1. ***Дек. Различные реализации. Описания типов для различных реализаций. Реализация одной из функций дека.***

Деком называется структура данных, представляющих собой упорядоченное множество элементов, в котором элементы можно добавлять в начало и конец множества и извлекать их можно с обеих сторон.

Создание исполнителя дек предполагает наличие следующих функций

Ø инициализация

Ø добавление элемента в начало дека

Ø добавление элемента в конец дека

Ø взятие/извлечение элемента из начала дека

Ø взятие/извлечение элемента с конца дека

Ø проверка: пуст ли дек?

Ø очистка дека

struct Deque

{

int data[N]; //массив данных

int last; //указатель на конец

};

void Creation(Deque \*D) //создание дека

{ D->last=0; }

bool Full(Deque \*D) //проверка дека на пустоту

{ if (D->last==0) return true; else return false; }

//

Контейнерные классы предназначены для хранения данных, организованных определенным образом. Для каждого типа контейнера определены методы работы с его элементами, не зависящие от конкретного типа данных, которые хранятся в контейнере. Поэтому один и тот же вид контейнера можно использовать для хранения данных различных типов.

Двусторонняя очередь (дек) - последовательный контейнер\*\*, поддерживающий доступ к произвольным элементам и обеспечивает вставку и удаление из обоих концов очереди за постоянное время. Операции с элементами внутри очереди занимают время, пропорциональное количеству перемещаемых элементов. Доступ к элементам очереди осуществляется за постоянное время (оно несколько больше, чем для вектора).

\*\*Последовательные контейнеры обеспечивают хранение конечного количестваоднотипных элементов в виде непрерывной последовательности.

Дек может выполняться как на двусвязном списке, так и на массиве

Дек на списке:

typedef int value\_t;

typedef struct node\_t {

value\_t value;

struct node\_t \*next;

struct node\_t \*prev;

} node\_t;

typdef struct deque\_t {

node\_t \*head;

node\_t \*tail;

} deque\_t;

Дек на массиве:

typdef int value\_t;

typedef struct deque\_arr\_t {

value\_t \*storage; // хранилище для дека

int capacity; // ёмкость хранилища

int size; // реальный размер дека

} deque\_arr\_t;

void push\_back(deque\_t \*deque, value\_t value) {

node\_t \*node;

node = (node\_t \*) malloc(sizeof(node\_t));

node->value = value;

node->next = NULL;

if ((\*deque)->tail) {

node->prev = (\*deque)->tail;

(\*deque)->tail->next = elem;

}

(\*deque)->tail = elem;

if ((\*deque)->head == NULL)

(\*deque)->head = elem;

}

1. ***Деревья поиска. Определение. Основные операции. Описание типов, необходимых для реализации (два типа). Реализация функции поиска элемента без рекурсии.***

Деревья поиска – такие бинарные деревья (деревья, в которых, как правило, задан корневой элемент или корень и для каждой вершины существует не более двух потомков), что для любой вершины дерева **a** ключи всех вершин в правом поддереве больше или равны ключа **a**, а в левом – меньше. Неравенства можно заменить на строгие, если известно, что в дереве нет равных элементов.

Основные операции, которые можно совершать с бинарными деревьями:

         создание дерева

         очистка дерева

         удаление дерева

         пусто ли дерево?

         поиск элемента в дереве (т.е. элемента с ключом, равным заданному)

         добавление элемента в дерево

         удаление элемента из дерева

         поиск минимального и максимального элемента в дереве

         если рассмотреть дерево поиска, как упорядоченную по возрастанию последовательность элементов, то: для текущего элемента поиск следующего/предыдущего

         для данной вершины дерева ***v***разбиение дерева поиска ***T***на два дерева поиска ***T1***и ***T2***таких, что все элементы в ***T1*** меньше или равны***v***, и все элементы в ***T2*** больше или равны***v***

         для двух деревьев поиска ***T1***и ***T2***таких, что все элементы в ***T1*** меньше или равнывсех элементов в ***T2***(будем далее для таких деревьев говорить, что ***дерево T1 меньше или равно дерева  T2***:  ***T1*** ***T2***): слияние деревьев в одно дерево поиска ***T***

***\*для дерева поиска – реализация только с родительским элементом***

Описание типов:

template <class T> class CNode

{ T v;

CNode <T> \*par, \*left, \*right;

CNode() { par=left=right=NULL;

};

Template <class T> class CTree

{

CNode<T> \*root;

public:

///

};

Поиск элемента без рекурсии проведем с помощью стека.

Код в http://espressocode.top/inorder-tree-traversal-without-recursion/

1. ***Директивы препроцессора: #define, #undef, #if..#else..#endif, #ifdef, #pragma once. Правила хорошего тона при оформлении оператора #define.***

Директива #define принимает две формы: определение констант; определение макросов.

Директива #undef переопределяет константу или препроцессорный макрос, ранее определенный с помощью директивы #define.

**Директивы #if, #elif, #else, #endif:**

#if <ограниченное-константное-выражение> [<текст>]

[#elif **<**ограниченное-константное-выражение> <текст>]

[#elif **<**ограниченное-константное-выражение**>** <текст>]

[#else **<**текст**>**]

#endif

Каждая директива **#**if в исходном файле должна соответствовать закрывающей директиве **#endif** . Между директивами **#if** и **#endif** может использоваться любое число директив **#elif** , но допускается не более одной директивы **#else** . Директива **#else** , если она есть, должна быть последней директивой перед **#endif**.

Препроцессор выбирает участок текста для обработки на основе вычисления <ограниченного-константного-выражения>, следующего за каждой директивой **#if** или **#elif**. Выбирается <текст>, следующий за <ограниченным-константным-выражением> со значением истина (не нуль), вплоть до ближайшей директивы **#elif**, **#else**, или **#endif**, ассоциированной с данной директивой **#if**.

Если ни одно ограниченное константное выражение не истинно, то препроцессор выбирает <текст>, следующий за директивой **#else**. Если же директива **#else** отсутствует, то никакой текст не выбирается.

Директива #ifdef проверяет, был ли ранее определен макрос или символическая константа как #define. Если — да, компилятор включает в программу код, который находится между директивами #ifdef и #else, если nameToken ранее определен не был, то выполняется код между #else и #endif, или, если нет директивы #else, компилятор сразу переходит к #endif.

Директива #pragma используется для доступа к специфическим расширениям компилятора. Совместное использование директивы #pragma c лексемой once просит компилятор включить файл заголовка только один раз, независимо от того, сколько раз она был импортирован.

Правила хорошего тона для define:

Все аргументы define всегда заключаются в скобки. Аргументы, передаваемые в define, не должны меняться в процессе передачи. Директивы define задаются большими буквами:

#define SWAP(a,b) {t=(a); (a)=(b); (b)=t;}

1. ***Директивы препроцессора: #include. Правила оформления (с помощью директив #ifdef/#define и #pragma once)***

Директива #include вставляет код из указанного файла в текущий файл, то есть, просто подключив другой файл, мы можем пользоваться его функциями, классами, переменными. Заголовочные файлы обычно находятся либо в текущей директории, либо в стандартном системном каталоге.

* Эти директивы использываются при "правильном" оформлении include-файла  
  include-файл выглядит так:  
  [#ifndef](https://vk.com/im?sel=238291837&st=%23ifndef) \_\_INCLUDEFILE1\_H\_\_  
  [#define](https://vk.com/im?sel=238291837&st=%23define) \_\_INCLUDEFILE1\_H\_\_  
  /\*include-файл\*/  
  [#endif](https://vk.com/im?sel=238291837&st=%23endif)  
  Работает это так:  
  Сначала проверяется объявлен ли до этого этот include-файл (при этом название может быть любым,  
  а также вместо него далее в файле заменяется на "пустоту")  
  Далее объявляется include-файл  
  В конце ставится директива показывающая конец проверки условия ifndef  
  Это обеспечивает однократное включение файла в проект  
  Если случайно (или специально) будет добавлен include-файл еще раз,  
  То так как include-файл был уже ранее добавлен, то "внутренность"  
  файла будет игнорирована компилятором  
  [#pragma](https://vk.com/im?sel=238291837&st=%23pragma) once  
  Эта директива работает аналогично.  
  Достаточно добавить эту директиву в начале include-файла  
  [#pragma](https://vk.com/im?sel=238291837&st=%23pragma) once  
  /\*include-файл\*/

То так как include-файл был уже ранее добавлен, то "внутренность"  
файла будет игнорирована компилятором

1. ***Для значений какого типа данных отводится 8 бит?*** char
2. ***Идеально-сбалансированные деревья. Определение. Построение. Какие операции можно выполнять? Связь с идеально-сбалансированными’ деревьями.***

Бинарное дерево называется ***идеально сбалансированным***, если длины всех ветвей, начинающихся в корне дерева и заканчивающихся в узле с хотя бы одним из нулевых указателей ***v->left***и ***v->right***, отличаются не более чем на 1. Следующее условие равносильно условию идеально сбалансированности дерева: длины любых двух ветвей, начинающихся в одной вершине дерева, отличаются не более чем на 1

Для построения идеально сбаланированного’ дерева ( а оно является идеально сбалансированным) по набору из ***N*** элементов упорядочим этот набор. После этого алгоритм построения дерева сводится к разбиению полученной последовательности *{ai}i=1,…,N*на последовательности *{ai}i=1,…,[N/2]*и *{ai}i=[N/2]+2,…,N*. Эти последовательности либо имеют равную длину (для нечетных ***N***), либо их длина отличается не более, чем на единицу (для четных ***N***). В корень создаваемого дерева помещаем элемент *a[N/2]+1*, а левое и правое поддеревья строим таким же алгоритмом, соответственно, для последовательностей *{ai}i=1,…,[N/2]*и *{ai}i=[N/2]+2,…,N*.

 Идеально сбалансированные деревья поиска удобны для операции поиска элемента в дереве. (отметим, что здесь мы рассматриваем только алгоритмы поиска, основанные на сравнениях). Все остальные операции (добавления элементов, удаления элементов и т.д.) в идеально сбалансированном дереве производятся с помощью полной перестройки дерева, т.е. они производятся медленно.

В литературе встречается другое (неравносильное) определение идеально сбалансированных деревьев. Будем называть ***идеально сбалансированными’***деревьями такие деревья, для которых для каждой вершины количество элементов в левом и правом поддереве отличается не более, чем на единицу.

Связь: *Идеально сбалансированное’ дерево является идеально сбалансированным.*

1. ***Идеально-сбалансированные’ деревья. Определение. Построение. Какие операции можно выполнять? Связь с идеально-сбалансированными деревьями.***

В литературе встречается другое (неравносильное) определение идеально сбалансированных деревьев. Будем называть ***идеально сбалансированными’***деревьями такие деревья, для которых для каждой вершины количество элементов в левом и правом поддереве отличается не более, чем на единицу.

Для построения идеально сбаланированного’ дерева по набору из ***N*** элементов упорядочим этот набор. После этого алгоритм построения дерева сводится к разбиению полученной последовательности *{ai}i=1,…,N*на последовательности *{ai}i=1,…,[N/2]*и *{ai}i=[N/2]+2,…,N*. Эти последовательности либо имеют равную длину (для нечетных ***N***), либо их длина отличается не более, чем на единицу (для четных ***N***). В корень создаваемого дерева помещаем элемент *a[N/2]+1*, а левое и правое поддеревья строим таким же алгоритмом, соответственно, для последовательностей *{ai}i=1,…,[N/2]*и *{ai}i=[N/2]+2,…,N*.

Операции – поиск по дереву. В этом случае, согласно построению идеально сбалансированного дерева’, поиск по дереву будет в точности совпадать с поиском по массиву, что делает само дерево ненужным.

Связь: *Идеально сбалансированное’ дерево является идеально сбалансированным.*

1. ***Какая комбинация знаков соответствует операции декрементирования?* х--**
2. ***Какая комбинация знаков соответствует операции присваивания с суммированием?* Х+=**
3. ***Какие объекты передаются в функции в языке С по ссылке?***

которые могут изменяться. Передача по ссылке позволяет возвратить из функции сразу несколько значений. Также передача параметров по ссылке является более эффективной при передаче очень больших объектов, поскольку в этом случае не происходит копирования значений, а функция использует сам объект, а не его значение.

1. ***Какие способы передачи параметров в функции существуют в различных языках программирования?***

ЛИБО

В С++ *передача параметров* осуществляется тремя способами:

· по копии (или по значению);

· по адресу (или через указатель);

· по ссылке.

В Си – по значению и указателю.

ЛИБО

Существует два метода, с помощью которых машинный язык может передавать аргумент подпрограмме (функции). Первый способ – передача параметра по значению (сюда же относятся указатели). Этот метод копирует значение [аргумента](https://docs.mql4.com/ru/basis/function/call#argument) в формальный параметр функции. Поэтому любые изменения этого параметра внутри функции не имеют никакого влияния на соответствующий аргумент вызова.

Второй способ – передача аргумента по ссылке. В этом случае ссылка на параметр (а не его значение) передается параметру функции. Внутри функции она используется для того, чтобы обратиться к фактическому параметру, указанному в вызове. Это означает, что изменения параметра будут влиять на аргумент, использованный для вызова функции.

1. ***Какое значение будет иметь переменная b после выполнения оператора:***

***int i, b, r, x;***

***i=2; b=i; r=b; x=r;***

***do {               b=(b+r)%2;        r=x/(i+1);            i++; }while (r>=3 || r<=-3);***

b=0 (выполнится 1 раз, тк условие проверяется после; r=0, i=3, x=2)

1. ***Какое значение будет иметь переменная i после выполнения оператора (переменные i, b, r, x имеют тип int):***

***for (i=2, b=1, r=0, x=5; r>3 || r<-3; ) {b+=r; r=r\*x-i; i-=(r-b); }***

i=2 (сначала проверится условие, поэтому тело цикла не выполнится; b=1, r=0, x=5)

1. ***Какое значение будет иметь переменная i после выполнения оператора:***

***int i, b, r, x;***

***for (i=5, b=2, r=4, x=2; r>3 || r<-3; ) {         b+=r;     r=r\*x-i; i-=(r-b); }***

i=8 (выполнится 1 раз; x=2, b=6, r=3)

1. ***Какое значение будет иметь переменная p после выполнения операций:***

***int p=3, x=6;  
if ((p%x)>2) p=p+x;else {      if (p%2==0) p=p-x;         else p=-p-x; }***

р=9 (3%6=3)

1. ***Какое значение будет иметь переменная p после выполнения операций:***

***int p=3;***

***switch (p<<1) {               case 2: break;    case 3: p+=1; break;       case 4: p=p+2;  default: p=0; }***

р=0 (0011 << 1 = 0110; это 6 => выполняется default)

1. ***Какое служебное слово используется для завершения текущей итерации цикла (продолжение цикла при помощи перехода к следующей итерации)?***

Continue

 все последующие за ним операторы пропускаются, осуществляя переход к концу цикла. Далее все зависит от вида цикла – с параметром или с условием. Если цикл с параметром (цикл for) или цикл с предусловием (while), то сразу переходим в начало и проверяем условие входа в цикл; если цикл с послеусловием (repeat), то прорверяется условие выхода из цикла.

1. ***Какое служебное слово является заголовком оператора цикла с постусловием?***

Do … while (c++)

     do  
     {  
         Блок инструкций  
     }  
     while (условие);

Cначала выполняется блок цикла, а потом проверяется условие. Если условие истинно, то цикл будет выполнен еще раз, и так до тех пор, пока условие будет истинно. Поскольку условие проверяется после выполнения тела цикла, то блок цикла с постусловием всегда будет выполнен хотя бы один раз, независимо от истинности условия.

1. ***Какое служебное слово является заголовком оператора цикла с постусловием?***

Do … while (c++)

     do  
     {  
         Блок инструкций  
     }  
     while (условие);

Cначала выполняется блок цикла, а потом проверяется условие. Если условие истинно, то цикл будет выполнен еще раз, и так до тех пор, пока условие будет истинно. Поскольку условие проверяется после выполнения тела цикла, то блок цикла с постусловием всегда будет выполнен хотя бы один раз, независимо от истинности условия.

1. ***Какой знак соответствует операции остаток от деления целых чисел***

**%**

1. ***Какой знак соответствует операции получения адреса операнда?***

**&**

1. ***Какой знак соответствует операции получения остатка от деления целочисленных операндов?***

**%**

1. ***Какой знак соответствует операции поразрядного инвертирования внутреннего двоичного кода целочисленного аргумента (побитовое отрицание)?***

**~**

1. ***Какой синтаксис (функциональная схема) соответствует конструкции if-else?***

if(условие)

{

Operations;

}

else if (условие)

{

Operations;

}

else

{

Operations;

}

1. ***Какой синтаксис соответствует конструкции do-while?***

do

{

Operations;

}

while (условие);

1. ***Какой тип данных имеет диапазон значений 0...+255?***

unsigned char

1. ***Логические операции (7шт). Выражения, воспринимаемые как логические.***

-Конъюнкция(&&, логическое умножение, Возвращает 1, если оба операнда не равны 0. Возвращает 0, если хотя бы один операнд равен 0.)

-дизъюнкция (||, логическое сложение, Возвращает 1, если хотя бы один операнд не равен 0. Возвращает 0, если оба операнда равны 0.)

-инверсия (!, отрицание, Если операнд равен 0, то возвращает 1, иначе возвращает 0.)

-импликация,(-> (!a||b), логическое следование, ложна, если первый операнд равен 1, а второй – 0)

-эквивалентность (≡, логическая равнозначность; истинна, если значения операндов равны)

-строгая дизъюнкция (^,сложение по модулю два, истинна, если значения операндов не равны)

-стрелка Пирса (истинна, только если оба операнда ложны)

-штрих Шеффера (ложна, только если оба операнда истинны).

Выражения воспринимаются, как логические, если в них есть операторы сравнения и логические операции.

1. ***Математическая библиотека. Основные функции. Правила использования функций pow(), округления.***

abs(a); - модуль или абсолютное значение a

sqrt(a); - корень квадратный из а, причем а не отрицательно

ceil(a);- округление а до наименьшего целого, но не меньше чем а

ceil(2.3)=3.0; ceil(-2.3)=-2.0

floor(a);- округление а до наибольшего целого, но не больше чем а

floor(2.3)=2.0;floor(-2.3)=-3.0

fmod(a,b); - вычисление остатка от целочисленного деления a на b

exp(a);- вычисление экспоненты e в степени a

sin(a); - a задается в радианах

cos(a); - а задается радианах

log(a); - натуральный логарифм

log10(a); - десятичный логарифм

asin(a) – арксинус а, -1.0 < a < 1.0

pow(a,b) - Возведение значения а в степень b

Если величина результата настолько велика, что она не может быть представлена в объекте возвращаемого типа, возникает ошибка области допустимых значений, и функция возвращает значение HUGE\_VAL с соответствующим ему знаком.

Если a  отрицательное и b не целое значение, или если а равна нулю, а b — отрицательное, возникает ошибка области допустимых значений.

1. ***Может ли прототип double sign(double); являться прототипом функции с заголовком double sign(double k)?***

Да, может в С++. В С необходимо прописывать переменную k, также это удобнее, потому что можно копировать объявление функции в определение.

1. ***Можно ли использовать "continue" внутри оператора switch?***

Так не нужно делать, так как он будет относиться к внешнему циклу для switch, если таковой имеется.

1. ***Можно ли использовать "return" внутри оператора switch?***

Да, можно. Пример:

[#include](https://vk.com/im?sel=324817604&st=%23include) <iostream>  
using namespace std;  
int main()  
{  
int i;  
cout«"input number from 0 to 2:";  
cin»i;  
switch(i)  
{  
case 0: return 0;  
case 1: return 1;  
case 2: return 2;  
default: return 0;  
}  
}

1. ***Написать функцию, рекурсивно (!) определяющую является ли целое число N точной степенью двойки. Привести пример вызова такой функции.***

#include <iostream>

#include <math.h>

using namespace std;

void func(double N)

{

 double c,d;

 c=modf(N,&d);

 if (c!=0) {cout<<"NO"<<endl;return;}

 if (N==1) {cout<<"YES"<<endl;return;}

 return func((double)N/2);

}

int main()

{

 int N;

 cin>>N;

 func(N);

}

1. ***Напишите выводимое значение. int mass[]={1, 6, 9, 2, 5};int \*mass2 = &mass[3];printf("%d", mass2[-1]);***

9.

1. ***Напишите десятичное значение «-16» в прямом, обратном и дополнительном двоичном коде (ответ дать в 8битном представлении).***

1001 0000 //старший бит под минус, затем значение 1110 1111 //старший бит под минус, затем инверсия  
1111 0000 //инверсия +1

1. ***Напишите значение выражения: (!1)+(!0)+(!(-5)).***

1110 + 1111 + !(1010)= 14+15+5= 34  
для си имеем 0+1+0=1.

1. ***Напишите значение выражения: (!18)-(5<<2)+(6^5).***

(!18)-(5<<2)+(6^5)= -19-20+3=-36

1. ***Напишите значение выражения: (3==5)? 3 : 5.***

5

1. ***Напишите значение выражения: (4<<2)+(6&5)-(6^5);***

(4<<2)+(6&5)-(6^5) = 16+4-3=17

1. ***Напишите значение выражения: (5<<1)+(3!=5)+(13%(-4))+((-17)%4)***

(5<<1)+(3!=5)+(13%(-4))+((-17)%4)=10+6+1+3=20

1. ***Напишите значение выражения: (7>>1)+((4^7)%5);***

(7>>1)+((4^7)%5)=3+3%5=3+3=6

1. ***Напишите значение выражения: 3+4>5 && 3+5>4 && 4+5>3;***

1.

1. ***Напишите идентификатор, определяющий действия при отсутствии нужного варианта в операторе switch?***

default

1. ***Напишите идентификатор, соответствующий операции определения размера операнда?***

sizeof

1. ***Напишите название базового типа данных, который указывает на целое число увеличенной длины (длинное целое).***

long int

1. ***Напишите название директивы макроподставноки.***

#define

1. ***Напишите название директивы препроцессора определения макроса или препроцессорного идентификатора.***

#define

1. ***Напишите название директивы препроцессора проверки неопределённости идентификатора.***

#ifndef

1. ***Напишите название квалификатора, который можно использовать с базовыми типами данных для установления «короткого» целого числа.(некорректный вопрос – это модификатор)***

short

1. ***Напишите название квалификатора, который можно использовать с базовыми типами данных для установления значений со знаком. (некорректный вопрос – это модификатор)***

signed

1. ***Напишите название оператора, который возвращает размер в байтах переменной или типа?***

sizeof(object); sizeof(type name)

1. ***Напишите название функции арксинуса, принадлежащей математической библиотеке.***

asin(object)

1. ***Напишите название функции записи символа в файл.***

fprintf(FILE \*stream, const char \*format, …)

1. ***Напишите название функции установки произвольной позиции в файле.***

fseek (FILE \*stream, long offset, int origin). /\*Устанавливает указатель положения в файле, связанном со stream, в соответ­ствии со значениями offset и origin. Аргумент offset — это выраженный в байтах сдвиг от позиции, определяемой origin, до новой позиции. Аргумент origin может принимать значения 0, 1 или 2, причем 0 означает начало файла, 1 — текущую позицию, а 2 — конец файла.\*/

1. ***Напишите название функции форматированного чтения из файла.***

fscanf(FILE \*stream, const char \*format, …)

1. ***Напишите название функции, возвращающей значение натурального логарифма***

log(a)

1. ***Напишите название функции, возвращающей значение первого числа в степени второго***

pow(a,b)

1. ***Напишите название функции, возвращающей наибольшее целое, не превышающее данное значение.***

floor(x)

1. ***Напишите название функции, выполняющей выделение памяти и перенос в неё копию заданной строки.***

strcpy(char \*str1, const char \*str2)

1. ***Напишите название функции, выполняющей поиск второй заданной строки в первой.***

strstr(const char \*str1, const char \*str2)

1. ***Напишите название функции, выполняющей сравнение заданных строк.***

strcmp(const char \*str1, const char \*str2)

1. ***Напишите название функции, выполняющей сцепление строк символов.***

strcat(char \*str1, const char \*str2)

1. ***Напишите название функции, вычисляющей ближайшее целое, не меньшее, чем данный вещественный аргумент***

# ceil (double num)

1. ***Напишите название функции, считывающей один символ из файла.***

fgetc(FILE \*stream)

1. ***Напишите название функции, считывающей очередной символ из стандартного входного потока (stdin).***

getc(FILE \*stream), getchar(void)

1. ***Напишите последовательность символов, выводимую при выполнении кода:***

***… int main ()***

***{char str[] = "12345678910";***

***char \* pch;pch=strchr(str,'5');***

***printf ("%s\n", pch);***

***return 0;}***

567890

1. ***Напишите последовательность символов, выводимую при выполнении кода:***

***… int main(void)***

***{char s1[80], s2[80];***

***strcpy(s1, "A1");***

***strcpy(s2, "B2");***

***strcat(s1, s2);***

***printf("%s", s1);***

***return 0; }***

A1B2

1. ***Напишите последовательность символов, выводимую при выполнении кода:***

***… int main(void)***

***{int a=27;char str[] = "12345678910";***

***printf("%2s\n", str);***

***return 0;}***

1234567890

1. ***Напишите последовательность символов, используемую внутри printf для выведения вывода вещественного числа в поле из 7 позиций.***

%7d

1. ***Напишите последовательность символов, используемую внутри printf для выведения двойных кавычек.***

\»

1. ***Напишите последовательность символов, используемую внутри printf для выведения строки символов.***

%s

1. ***Напишите последовательность символов, используемую внутри printf для выведения табуляции.***

\t

1. ***Напишите последовательность символов, используемую внутри printf для выведения указателя.***

%p или %n

1. ***Напишите последовательность символов, используемую внутри printf для выведения целого десятичного числа.***

%d или %i

1. ***Напишите последовательность символов, используемую внутри printf для выведения числа в экспоненциальной форме.***

%e или %E

1. ***Напишите последовательность символов, используемую внутри printf для выведения числа с двойной точностью***.

%2

1. ***Напишите последовательность символов, используемую внутри printf для выведения шестнадцатеричного числа.***

%x или %X

1. ***Напишите последовательность символов, используемую внутри printf для печати в выводимой строке не менее 15 символов.***

%.15s

1. ***Общая структура С-программы. Оформление include-файлов. Что может содержаться в include-файлах и что в C-файлах.***

Программа на языке С **состоит из файлов**. Файлы бывают обычными С-файлами и include-файлами. Include-файлы существуют только для того, чтобы включаться в другие файлы программы (в C-файлы и include-файлы). Компилятор языка С include-файлы не видит. Их вставка осуществляется препроцессором, который запускается перед компилятором. На самом деле, запуск, компилятора может тоже состоять из запуска отдельных программ.

Что есть в файлах:

·         Директивы препроцессора

·         Директивы *#pragma* управления процессом компиляции

·         Описания и определения функций

·         Описания и определения глобальных переменных

·         Определения новых типов

Вообще говоря, функции можно пописывать в include-файлах, но если функции занимает более одной строки, то необходимо выносить ее описание в С-файл. А так все остальное пишется в include-файле.

1. ***Однонаправленные списки. Описание типов, необходимых для реализации (два типа). Определение одной из функций списка.***

Создание исполнителя *односвязный список* (*L1-список*) предполагает наличие следующих функций

Ø  инициализация

Ø  установка текущего элемента в начало списка

Ø  перемещение текущего элемента к следующему элементу списка

Ø  взятие значения текущего элемента

Ø  уничтожение текущего элемента с автоматическим перемещением текущего элемента к следующему элементу списка

Ø  вставка нового элемента после текущего

Ø  проверка: пуст ли список?

Ø  проверка: текущий элемент в конце списка?

Ø  очистка списка

template <class T> class CListNode

{public:

CListNode<T> \*next;

T v;

CListNode(){next=NULL;}

}

template<class T> class CList1

{CListNode<T> t, \*cur;

public:

void AddAfter(const T&x) {CListNode<T> \*p=new CListNode<T>; p->v=x; p->next=cur->next; cur->next=p; }

}

1. ***Операторы перехода (break, continue, goto). Описать случаи, когда использование оператора goto является дурным тоном и когда оно необходимо.***

Break – завершает выполнение цикла.

Continue - выполняетпереход в конец тела цикла.

Goto – выполняет переход из одного участка цикла в другой, определяемый лэйблом.

Goto является дурным тоном,если он ведет к верхней части программы и может при измерении программы случайно привести к бесконечной рекурсии.

- Если некоторая переменная инициализируется (получает значение) в одном месте и потом используется далее, то переход в точку после инициализации, но до использования, приведёт к тому, что будет использовано значение, которое находилось в памяти, выделенной под переменную, до момента выделения (и которое, как правило, является произвольным и случайным).

- Передача управления внутрь тела цикла приводит к пропуску кода инициализации цикла или первоначальной проверки условия. Аналогично, передача управления внутрь процедуры или функции приводит к пропуску её предисловия (пролога), в котором производится инициализация (выделение памяти под локальные переменные и т. п.).

Использование goto для выхода из нескольких вложенных циклов сразу в этом случае значительно упрощает код программы, избавляя от необходимости применения вспомогательных переменных-флагов и условных операторов. Если в языке нет средств обработки исключений, то оператор goto может использоваться для прерывания «нормального» выполнения кода и перехода к завершающему коду для освобождения занятой памяти и прочих финальных действий.

1. ***Операторы условия и выбора (if, switch). Объяснить принцип реализации оператора switch на уровне машинных команд.***

if – условный оператор, который может быть в форме полной(if (условие){блок операций1;}else{блок операций2;} ) и неполной(if (условие){блок операций1;}) развилки.

switch – оператор множественного выбора.

switch (ЦелоеВыражение)

{

case Константа1: БлокОпераций1 ;

break;

case Константа2: БлокОпераций2 ;

break;

…

case КонстантаN: БлокОперацийN ;

break;

default: БлокОперацийПоУмолчанию ;

break;

}

Реализация: выбор случая значения переменной m может выполняться за константу операций независимо от количества описываемых случаев. Каждый случай занимает несколько байт. Switch(m) - перейти на оператор, который имеет следующий адрес, по сравнению с адресом, в котором программа сейчас находится: m – N\*(размер одного кусочка).

1. ***Операторы цикла (for, while, do-while). Оператор запятая.***

* for ( /\*инициализация переменной; условие; изменение значения переменной\*/ )

{ // тело цикла (тут находится код который будет повторяться)}

* while(/\*условие\*/)

{ // тело цикла (тут находится код который будет повторяться)}

* do{//тело цикла} while(/\*условие\*/);
* Оператор «запятая» используется для связки нескольких выражений. Левая сторона оператора «запятая» всегда вычисляется как void (то есть не выдающее значения). Это означает, что значение выражения, находящегося с правой стороны, станет значением разделенного запятыми выражения. Например:  
    
  х = (у = 3, у + 1);  
    
  Сначала присваивается 3 переменной у, а затем 4. переменной х.

1. ***Операция ? : . Рекурсивное использование данного оператора.***

Тернарный оператор: условие? команда1 : команда2. Команда1 выполняется, если условие истинно,команда2 – если оно ложно.

Пример: максимум из двух чисел c = (a > b) ? a : b;

x<0?0:x<1?1:x<2?2:3

1. ***Опишите ситуацию, в которой sizeof(x) возвращает значение, отличное от 10, в случае если x описывается как char x[10].***

Если мы имеем в виду размер указателя, а не массива. Sizeof это оператор языка, он возвращает размер объекта, который ему передан, в байтах. Например, в данном случае под "объектом" может пониматься указатель на строку, а не сама строка.

1. ***Основные скалярные типы (различные целые, вещественные). Формат представления констант (различных целых, различных вещественных, символьных, строковых).***

Ответ: Целые типы данных могут быть в знаковой и беззнаковой форме. Беззнаковые целые числа – последовательность битов в диапазоне от 0 до 2^n-1, n-количество занимаемых битов. Знаковые целые числа представляются в диапазоне -2^(n-1)…+2^(n-1)-1. При этом старший бит данного отводится под знак числа (0 – положительное, 1 – отрицательное).  
Основные типы и размеры целочисленных данных:  
беззнаковый unsigned char, знаковый char – 8 бит  
беззнаковый unsigned short, знаковый short – 16 бит  
беззнаковый unsigned int, знаковый int – 32 бита  
беззнаковый unsigned long int, знаковый long int – 64 бита

Вещественный тип предназначен для обозначения действительных чисел. Вещественные числа представляются в виде: x = s \* m \* 2d, где s – знак (отводится один бит), m – мантисса в виде 1.xxxxx (где x – двоичная цифра; 1 не хранится), d – степень (оставшиеся биты). Различают три основных типа: float (32 бита), float (64 бита), long double (80 бит).  
  
*Константа* — это фиксированное значение, которое не может быть изменено программой. Используя суффикс, можно явно указать тип числовой константы. Если после числа в плавающем формате стоит суффикс F, то считается, что константа имеет тип float, а если L, то long double. Для целых типов суффикс U означает unsigned, a L — long.

1. ***Очередь. Различные реализации. Описания типов для различных реализаций. Реализация одной из функций очереди.***

Ответ: Очередью называется структура данных, организованная по принципу FIFO – first-in, first-out , т.е. элемент, попавшим в множество первым, должен первым его и покинуть. При практическом использовании часто налагается ограничение на длину очереди, т.е. требуется, чтобы количество элементов не превосходило N для некоторого целого N.

Создание исполнителя очередь предполагает наличие следующих функций

-инициализация

-добавление элемента в конец очереди

-взятие/извлечение элемента с начала очереди

-проверка: пуста ли очередь?

-очистка очереди

struct SQueue

{

int Array[N];

int Begin, End;

};

Функция инициализации очереди может выглядеть

void Init(struct SQueue \*queue){ queue->Begin=queue->End=N-1;}

Функция добавления элемента может выглядеть следующим образом

int Add(struct SQueue \*queue, int value)

{

if(queue->End - queue->Begin == 1 ||

queue->Begin - queue->End == N-1)return -1;

queue->Array[queue->End-- ]=value;

if(queue->End<0) queue->End=N-1;

return 0;

}

1. ***Почему при выполнении функции printf(“1”) на экране может ничего не появиться. Как исправить эту ситуацию?***

Ответ: в си необходимо указывать формат и подключать stdio.

1. ***Представление целых и вещественных чисел в ЭВМ. IEEE-стандарт представления вещественных чисел. Объяснить логику компилятора, не позволяющего сравнивать на равенство вещественные числа.***

Ответ: Беззнаковые целые используется двоичное представление чисел.

x = a{n-1}\*2^(n-1) + a{n-2}\*2(n-2) + … a{1}\*2^1 + a{0}\*2^0, где каждый член a{i} представляет собой 0 или 1 или, иначе говоря, одну цифру в двоичном представлении данных, а n задается количеством бит (в одном бите хранится одна двоичная цифра), отводящихся под данное число.  
  
Целых со знаком существует три основных формата:

– прямой (старший бит служит признаком знака: 1=’-‘; 0=’+’)

– обратный (для представления отрицательного числа берется представление его модуля, а потом все биты инвертируются)

– дополнительный (для представления отрицательного числа берется представление его модуля, все биты инвертируются, к результату прибавляется 1)

 Реально, в основном, используется только дополнительный код.

Вещественные числа с фиксированной запятой. VB тип Currency - восьмибайтовая переменная, которая содержит вещественное число в формате с фиксированной десятичной запятой (четыре десятичных знака после запятой).

C# тип decimal

Вещественные числа с плавающей запятой. Формат закреплен IEEE: Institute of Electrical and Electronics Engineers, где число представляется в виде x = s \* m \* 2d, где s – знак (отводится один бит), m – мантисса в виде 1.xxxxx (где x – двоичная цифра; 1 не хранится), d – степень (оставшиеся биты).

 Согласно формату IEEE вместо степени хранится характеристика = d-dmin , где dmin – минимально возможное значение мантиссы. Таким образом, минимально возможное значение характеристики = 0. На самом деле, по стандарту IEEE  представление более сложное. Пока характеристика положительна поддерживается вышеуказанное представление. Для нулевой характеристики число хранится уже как число с фиксированное запятой в виде x = p \* 2-dmin, где  p – число с фиксированной точкой в виде x.xxxxx (где x – двоичная цифра; хранятся все указанные биты).  
  
Числа с плавающей запятой не могут быть представлены точно, по-этому мы не можем полагаться на оператор сравнения.

1. ***Работа с бинарными файлами. В том числе, использование функций fseek()/ftell()***

Ответ: Можно хранить данные непосредственно в бинарном виде. Для этих целей используются бинарные файлы.

Запись в файл осуществляется с помощью функции

**size\_t fwrite ( const void \* ptr, size\_t size, size\_t count, FILE \* stream );**

Функция возвращает число удачно записанных элементов. В качестве аргументов принимает указатель на массив, размер одного элемента, число элементов и указатель на файловый поток.

Функция fseek:  
**int fseek ( FILE \* stream, long int offset, int origin )**

Эта функция устанавливает указатель позиции, ассоциированный с потоком, на новое положение. Индикатор позиции указывает, на каком месте в файле мы остановились. Когда мы открываем файл, позиция равна 0. Каждый раз, записывая байт данных, указатель позиции сдвигается на единицу вперёд.

fseek принимает в качестве аргументов указатель на поток и сдвиг в offset байт относительно origin. origin может принимать три значения

SEEK\_SET - начало файла, SEEK\_CUR - текущее положение файла, SEEK\_END - конец файла. К сожалению, стандартом не определено, что такое конец файла, поэтому полагаться на эту функцию нельзя.

В си определён специальный тип fpos\_t, который используется для хранения позиции индикатора позиции в файле.

Функция

**int fgetpos ( FILE \* stream, fpos\_t \* pos );**

используется для того, чтобы назначить переменной pos текущее положение. Функция

**int fsetpos ( FILE \* stream, const fpos\_t \* pos );**

используется для перевода указателя в позицию, которая хранится в переменной pos. Обе функции в случае удачного завершения возвращают ноль.

**long int ftell ( FILE \* stream );**

возвращает текущее положение индикатора относительно начала файла. Для бинарных файлов - это число байт, для текстовых не определено (если текстовый файл состоит из однобайтовых символов, то также число байт).

1. ***Работа с текстовыми файлами. Правила использования функции fgets(). Считывание строк неограниченной длины с помощью функции fgets().***

Ответ:

Функция fopen может открывать файл в текстовом или бинарном режиме. По умолчанию используется текстовый. Режим доступа может быть следующим:

r Чтение. Файл должен существовать.

w Запись нового файла. Если файл с таким именем уже существует, то его содержимое будет потеряно.

a Запись в конец файла. Операции позиционирования (fseek, fsetpos, frewind) игнорируются. Файл создаётся, если не существовал.

r+ Чтение и обновление. Можно как читать, так и писать. Файл должен существовать.

w+ Запись и обновление. Создаётся новый файл. Если файл с таким именем уже существует, то его содержимое будет потеряно. Можно как писать, так и читать.

a+ Запись в конец и обновление. Операции позиционирования работают только для чтения, для записи игнорируются. Если файл не существовал, то будет создан новый.

Если необходимо открыть файл в бинарном режиме, то в конец строки добавляется буква b, например “rb”, “wb”, “ab”, или, для смешанного режима “ab+”, “wb+”, “ab+”. Вместо b можно добавлять букву t, тогда файл будет открываться в текстовом режиме.

char \*fgets(char \*str, int num, FILE \*f)

Функция fgets() считывает до num-1 символов из файла f и помещает их в массив символов, на который указывает str. Символы считываются до тех пор, пока не встретится символ «новая строка», EOF или до достижения указанного предела.

По окончании считывания в массив str сразу после последнего считанного символа помещается нулевой символ.

Символ «новая строка» при считывании будет сохранен и станет частью массива str.

В случае удачи fgets() возвращает str, при неудаче возвращается NULL.

В случае ошибки по чтению содержимое массива, на который указывает str, не определено. Поскольку как в случае ошибки, так и при достижении конца файла возвращается null, для определения того, что имен­но произошло, необходимо использовать feof() или ferror().

Пример:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

void main() {

FILE \*f;

char m[128];

f = fopen("C:/c/test.txt", "w");

fprintf(f, "Hello, World!");

fclose(f);

f = fopen("C:/c/test.txt", "r");

fgets(m, 127, f);

printf("%s", m);

fclose(f);

return 0;

}

Считывание строк неограниченной длины с помощью функции fgets():

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

int main(void){

FILE in;

char str;

char tmp[100];

int k = 0;

in = fopen(1.txt, r);

while(fgets(tmp, 100, in) != NULL) {

k += strlen(tmp);

str = (char)realloc(str, k);

strcat(str, tmp);

}

printf(%s, str);

return 0;

}

1. *Реализация списков с самостоятельным выделением памяти. Описание типов, необходимых для реализации (два типа). Определение одной из функций списка.*

Ответ:

// тип значения в элементе списка

typedef int value\_t;

// тип элемента списка

typedef struct node\_t {

value\_t value;

struct node\_t \*next;

} node\_t;

void add\_to\_list(node\_t \*\*list, value\_t value) {

node\_t \*node, \*tmp;

node = (node\_t \*) malloc(sizeof(node\_t));

node->value = value;

node->next = NULL;

if (\*list == NULL) {

\*list = node;

}

else {

tmp = \*list;

while (tmp->next)

tmp = tmp->next;

tmp->next = node;

}

}

1. ***Стандартная библиотека ввода-вывода: семейство функций printf(). Основные спецификации формата. Объяснить путаницу в gcc с форматами %f и %lf***

**Ответ:**

int printf(const char \*управляющая\_строка, ...);

Функция printf() возвращает число выведенных символов или отрицательное значение в случае ошибки.

Управляющая\_строка состоит из элементов двух видов. Первый из них — это символы, которые предстоит вывести на экран; второй — это спецификаторы преобразования, которые определяют способ вывода стоящих за ними аргументов. Каждый такой спецификатор начинается со знака процента, за которым следует код формата.

Аргументов должно быть ровно столько, сколько и спецификаторов, причем спецификаторы преобразования и аргументы должны попарно соответствовать друг другу в направлении слева направо.

Спецификации:

%c Символ

%d Десятичное целое со знаком

%e Экспоненциальное представление

%f Десятичное с плавающей точкой

%s Строка символов

%n Аргумент, соответствующий этому спецификатору, должен быть указателем на целочисленную переменную.

Спецификатор позволяет сохранить в этой переменной количество записанных символов (записанных до того места, в котором находится код %n)

1. *Стандартная библиотека ввода-вывода: семейство функций scanf().Основные спецификации формата (в том числе форматы %n и [] с примерами использования).*

Ответ:

Функция scanf() представляет собой процедуру ввода общего назначения, которая читает поток stdin и сохраняет информацию в переменных, перечисленных в списке аргументов.

Она может читать все встроенные типы данных и автоматически преобразовывать их в соответствующий внутренний формат.

Основные спецификации формата:

%d Читает десятичное целое

%f Читает число с плавающей точкой

%g Читает число с плавающей точкой

%s Читает строку

%n Принимает целое значение, равное количеству прочитанных до сих пор символов

%[] Просматривает набор символов

По умолчанию спецификации a, f, e и g заставляют функцию scanf() присваивать данные переменным типа float.

Если перед одной из этих спецификаций поставить модификатор l, функция scanf() присвоит прочитанные данные переменной типа double.

Использование же модификатора L означает, что полученное значение присвоится переменной типа long double.

Функция scanf() поддерживает спецификатор формата общего назначения, называемый набором сканируемых символов (scanset). В этом случае определяется набор символов, которые могут быть прочитаны функцией scanf() и присвоены соответствующему массиву символов. Для определения такого набора символы, подлежащие сканированию, необходимо заключить в квадратные скобки. Открывающая квадратная скобка должна следовать сразу за знаком процента. Например, следующий набор сканируемых символов указывает на то, что необходимо читать только символы A, B и C.

%[ABC]

При использовании набора сканируемых символов функция scanf() продолжает читать символы и помещать их в соответствующий массив символов до тех пор, пока не встретится символ, отсутствующий в заданном наборе. Соответствующая набору переменная должна быть указателем на массив символов. При возврате из функции scanf() этот массив будет содержать строку из прочитанных символов, завершающуюся символом конца строки.

Если первый символ в наборе является знаком ^, то получаем обратный эффект: входное поле читается до тех пор, пока не встретится символ из заданного набора сканируемых символов, т.е. знак ^ заставляет функцию scanf() читать только те символы, которые отсутствуют в наборе сканируемых символов.

Во многих реализациях допускается задавать диапазон с помощью дефиса. Например, функция scanf(), встречая набор сканируемых символов в виде %[A-Z], будет читать символы, попадающие в диапазон от А до Z.

ЛИБО

Функция scanf(const char \*format, ...)

Позволяет считывать данные из стандартного потока stdin в заданном формате.

Управляющая строка, на которую указывает format, состоит из символов трех типов:

1. Спецификаторы формата

2. Специальные символы

3. Прочие символы (не специальные)

Основные спецификации формата:

%с Считать один символ

%d Считать десятичное число целого типа

%i Считать десятичное число целого типа

%е Считать число с плавающей запятой

%f Считать число с плавающей запятой

%g Считать число с плавающей запятой

%о Считать восьмеричное число

%s Считать строку

%х Считать шестнадцатиричное число

%р Считать указатель

%n Принимает целое значение, равное количеству считанных до текущего момента символов

%u Считывает беззнаковое целое

%[] Просматривает набор символов

%% Считывает символ %

scanf("%d:%d:%d", &year, &month, &day);

1. ***Стек. Различные реализации. Описания типов для различных реализаций. Реализация одной из функции стека.***

**Ответ:**

Стеком называется структура данных, организованная по принципу LIFO – last-in, first-out , т.е.

элемент, попавшим в множество последним, должен первым его покинуть. При практическом

использовании часто налагается ограничение на длину стека, т.е. требуется, чтобы количество

элементов не превосходило N для некоторого целого N.

Создание исполнителя стек предполагает наличие следующих функций

Ø инициализация

Ø добавление элемента на вершину стека

Ø взятие/извлечение элемента с вершины стека

Ø проверка: пуст ли стек?

Ø очистка стека

Стек можно реализовать на базе массива или (в языке С ) это можно сделать на базе указателей.

Стек. Реализация 1 (на основе массива).

Для реализации стека целых чисел, состоящего не более чем из 100 чисел, на базе массива в языке С

следует определить массив целых, состоящий из 100 чисел и целую переменную, указывающую на

вершину стека (ее значение будет также равно числу элементов в стеке)

int stack[100], i0=0;

Стек. Реализация 2 (на основе массива с использованием общей структуры).

struct SStack { int stack[100]; int i0 ; };

Инициализация: void InitStack(struct SStack \*ss){ ss->i0=0 ;}

Вызов функции осуществляется следующим способом : InitStack(&st2) ;

Стек. Реализация 3 (на основе указателей).

Если максимальный размер стека можно выяснить только после сборки программы, то память под

стек можно выделить динамически. При этом, вершину стека можно указывать не индексом в

массиве, а указателем на вершину (отметим, что оба способа указания вершины стека применимы в

обоих реализациях стека). Для этого следует определить следующие переменные

int \*stack, \*head;

Как и ранее, эти переменные можно объединить в структуру: struct SStack3{ int \*stack, \*head; };

Стек. Реализация 4 (на основе массива из двух указателей).

Однако, можно поступить и по-другому. Т.к. элементы stack и head имеют один тип, то их можно объединить в один массив объектов соответствующего типа (т.е. типа int\* ). Массив, естественно, должен быть длины 2:

int \*st4[2];

Здесь следует заметить, что при определении/описании переменных квадратные скобки имеют

приоритет больший, чем \*, поэтому переменная st4 имеет тип `массив указателей’, а не `указатель на

массив’.

Функция создания стека не более чем из n элементов может выглядеть, в простейшем случае,

следующим образом

void StackCreate4(int n, int \*st[2] ) {st[1]= st[0] = (int\*)malloc(n\*sizeof(int));}

а ее вызов будет выглядеть так: StackCreate4(n,st4);

Простейшая функция добавления элемента к стеку может выглядеть, в простейшем случае (без

проверки переполнения), следующим образом void StackAdd4(int v, int \* st[2] ) { (\*(st[1]++)) = v;}

1. ***Строки в языке С, принципы организации, основные функции (strlen, strcat, strcpy, strdup,strcmp, strstr, strchr)***

**Ответ:**

int strlen(const char \*);//длина строки

char \* strcpy(char \*, const char \*);//копирование второй строки в первую

char\* strdup (const char \*src);//указатель на копию строки. Если выделение памяти закончилось неудачей, то создания копии не происходит и функция возвращает NULL.

char \* strcat(char \*, const char \*);//подклеивание второй строки к первой

char \* strstr(char \*, const char \*);//поиск второй строки в первой

int strcmp(const char \*, const char \*);//лексикографич.сравнение строк

int strncmp(const char \*, const char \*,int n);//лекс.сравн. первых n байт строк

1. ***Указатели и массивы. Использование. Указать что общего, что различного и в каких случаях их использование дает различный результат.***

**Ответ:**

Фиксированный массив определяется следующим образом: int array [количество элементов]={элемент 1, элемент 2, ...};  
Адрес, хранящийся в переменной array является адресом первого элемента массива  
Переменная array и указатель на array являются разными элементами. Тип переменной array int [], а тип указателя на массив int \*

Основное различие между ними возникает при использовании оператора sizeof. При использовании в фиксированном массиве оператор sizeof возвращает размер всего массива\*размер элемента). При использвании с указателем оператор sizeof возвращает размер адреса памяти (в байтах). Второе различие возникает при использовании оператора адреса &. Используя адрес указателя, мы получаем адрес памяти переменной указателя. Используя адрес массива, возвращается указатель на целый массив. Этот указатель также указывает на первый элемент массива, но информация о типе различается.

1. *Функции и прототипы, оператор return. Объяснить логику компилятора, не позволяющего без описания определять функции перед использованием.*

Ответ:

1. ***Циклические списки. Описание типов, необходимых для реализации (два типа). Определение одной из функций списка.***

**Ответ:**

В них ссылки на концах списка циклически замыкаются. Хотя, формально, в данной структуре данных нет начала и конца, тем не менее, понятие первого элемента списка следует оставить, т.к. без него сложно, например,  реализовать перебор всех элементов списка. Список предписаний несколько модифицируется. Например, для двусвязного списка он может выглядеть следующим образом:

–  инициализация

–  установка текущего элемента в первый элемент списка

–  перемещение текущего элемента к следующему элементу списка

–  перемещение текущего элемента к предыдущему элементу списка

–  взятие значения текущего элемента

–  уничтожение текущего элемента с автоматическим перемещением текущего элемента к следующему элементу списка

–  вставка нового элемента после текущего

–  вставка нового элемента перед текущим

–  проверка: пуст ли список?

–  проверка: текущий элемент первый в списке?

–  очистка списка

**struct list \* init(int a)**// а- значение первого узла

**{**

**struct list \*lst;**

**// выделение памяти под корень списка**

**lst = (struct list\*)malloc(sizeof(struct list));**

**lst->field = a;**

**lst->ptr = lst; // указатель на сам корневой узел**

**return(lst);**

**}**

**struct list \* addelem(list \*lst, int number)**

**{**

**struct list \*temp, \*p;**

**temp = (struct list\*)malloc(sizeof(list));**

**p = lst->ptr; // сохранение указателя на следующий элемент**

**lst->ptr = temp; // предыдущий узел указывает на создаваемый**

**temp->field = number; // сохранение поля данных добавляемого узла**

**temp->ptr = p; // созданный узел указывает на следующий элемент**

**return(temp);**

**}**

1. ***Чем отличается предназначение конструкции if-else от конструкции #if-#else-#endif?***

Ответ: перед компиляцией есть возможность выбора. Во второй конструкции можно исходя из условий использовать или не использовать какой-либо код из основной компиляции. Порядок работы такой: сначала процессор анализирует код, потом что-то включается или исключается из него, а потом происходит компиляция.

ЛИБО

if-else - обычный условный оператор, исполняется программой.

#if-#else-#endif - команды препроцессора, они выполняются ещё на этапе компиляции

1. ***Чем отличается функция printf от функции fprintf.***

Ответ: fprintf записывает форматированный текст в указанный поток вывода.  
printf эквивалентно записи fprintf(stdout, ...) и записывает форматированный текст туда,куда указывает стандартный выходной поток.

1. ***Чем является "а" в выражении double \*a;?***

Ответ: указателем, так как имеется в виду конструкция <тип> \*<имя>;

1. ***Что может являться прототипом функции func?:  а) func(x); б) void func(double); в) double func(double x); г) int fun(double x);***

Ответ: б и в

В прототипе функции обязан содержаться тип функции, её название и типы переменных, поэтому подходят варинты б и в.

1. ***Что появится на экране при выполнении функции printf(“%d\n”,~-2);***

Ответ: 1