**Задание № 1 "Динамические массивы"**

**Цель:** Организация динамических массивов.

**1. Краткие теоретические сведения**

При традиционном определении массива:

тип имя\_массива [количество\_элементов];

общее количество памяти, выделяемой под массив, задается определением и равно количество\_элементов \* sizeof.

Но иногда бывает нужно чтобы память под массив выделялась для решения конкретной задачи, причем ее размеры заранее не известны и не могут быть фиксированы.

Формирование массивов с переменными размерами можно организовать с помощью указателей и средств динамического распределения памяти двумя способами:

* 1. с использованием библиотечных функций, описанных в за­головочных файлах alloc.h и stdlib.h (стандартный Си);
  2. с использованием операций new и delete (Си++).

**1.1. Формирование динамических массивов с использованием** **библиотечных функций**

Для выделения и освобождения динамической памяти исполь­зуются функции

|  |  |
| --- | --- |
| Функция | Прототип и краткое описание |
| malloc | void \* malloc(unsigned s) Возвращает указатель на начало области динамической памяти длиной в s байт, при неудачном завершении возвращает NULL |
| calloc | void \* calloc(unsigned n, unsigned m) Возвращает указатель на начало области динамической памяти для размещения n эле­ментов длиной по m байт каждый, при не­удачном завершении возвращает NULL |
| realloc | void \* realloc(void \* p, unsigned s) Изменяет размер блока ранее выделенной динамической памяти до размера s байт, р- адрес начала изменяемого блока, при не­удачном завершении возвращает NULL |
| free | void \*free(void p)  Освобождает ранее выделенный участок ди­намической памяти, р - адрес первого бай­та |

Пример:

Функция для формирования одномерного динамического масси­ва

int \* make\_mas(int n) (

int \*mas;

mas=(int\*)malloc(n\*sizeof(int)); for(int i=0;i<n;i++) mas[i]=random(10);

return mas; }

Для выделения памяти используется функция malloc, параметром которой является размер выделяемого участка памяти равный n\*sizeof(int). Так как функция malloc воз­вращает нетипизированный указатель void\*, то необходимо выполнить преобразование полученного нетипизированного указателя в указатель int\*.

Освободить выделенную память можно функцией free(mas).

**1.2. Формирование динамических массивов с использованием** **операций new и delete**

Для динамического распределения памяти используются операции new и delete. Операция new имя\_типа

или

new имя\_типа инициализатор позволяет выделить и сделать доступны свободный участок памяти, размеры которого соответствуют типу данных, опре­деляемому именем типа. В выделенный участок заносится значение определяемое инициализатором, который не являет­ся обязательным параметром. В случае успешного выделения памяти операция возвращает адрес начала выделенного уча­стка памяти, если участок не может быть выделен, то воз­вращается NULL. Примеры:

* + 1. int \*i; i=new int(10);
    2. float \*f; f=new float;
    3. int \*mas=new[5];

В примерах 1, 2 показано как выделить память под скаляр­ные переменные, пример 3 показывает выделение памяти под массив переменных.

Операция delete указатель освобождает участок памяти ранее выделенный операцией new. Пример:

Функция для формирования двумерного динамического массива

int \*\* make\_matr(int n) {

int \*\*matr; int i,j;

matr=new int\*[n];

for (i=0;i<n;i++) {

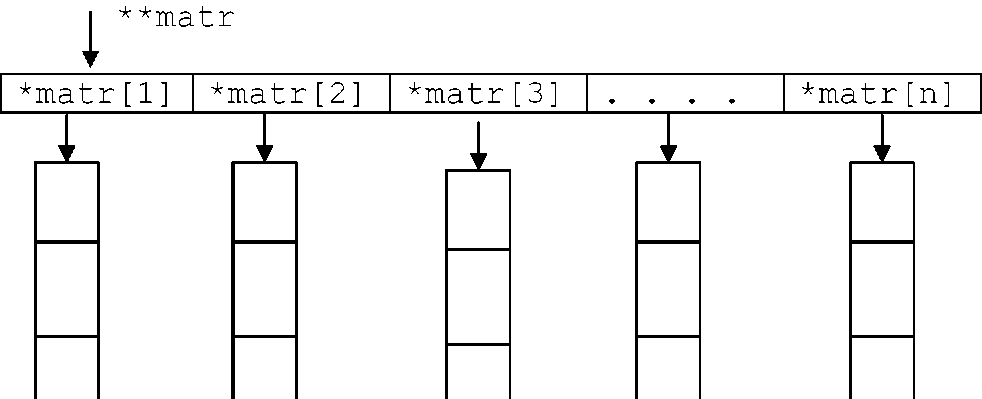
matr[i]=new int[n]; for (j=0;j<n;j++)

matr[i][j]=random(10); }

return matr; }

При формировании матрицы сначала выделяется памяти для массива указателей на одномерные массивы, а затем в цикле с параметром выделяется память под n одномерных массивов.

I \*\*matr



Чтобы освободить память необходимо выполнить цикл для освобождения одномерных массивов for(int i=0;i<n;i++) delete matr[i];

После этого освобождаем память на которую указывает указатель matr

delete [] matr;

**2. Постановка задачи**

Написать программу, в которой создаются динамические массивы и выполнить их обработку в соответствии со своим вариантом.

**3. Порядок выполнения работы**

* 1. Ввести размер массива;
  2. Сформировать массив с помощью операции new или биб­лиотечных функций malloc (calloc);
  3. Заполнить массив (можно с помощью датчика случайных чисел);
  4. Выполнить задание варианта, сформировать новый мас­сив (ы)-результат(ы);
  5. Напечатать массив(ы)-результат(ы);
  6. Удалить динамические массивы с помощью операции de­lete или библиотечной функции free.

**Вариант задания**

14. Сформировать двумерный массив. Удалить из него стро­ку и столбец, на пересечении которых находится мини­мальный элемент.

**Задание №2 "Информационные динамические структуры"**

**Цель:** Знакомство с динамическими информационными структу­рами на примере одно- и двунаправленных списков.

**1. Краткие теоретические сведения**

Во многих задачах требуется использовать данные у кото­рых конфигурация, размеры, состав могут изменяться в про­цессе выполнения программы. Для их представления исполь­зуют динамические информационные структуры.

Наиболее простая информационная структура - это од- носвязный список, элементами которого служат объкты структурного типа. Например

struct имя\_структурного\_типа

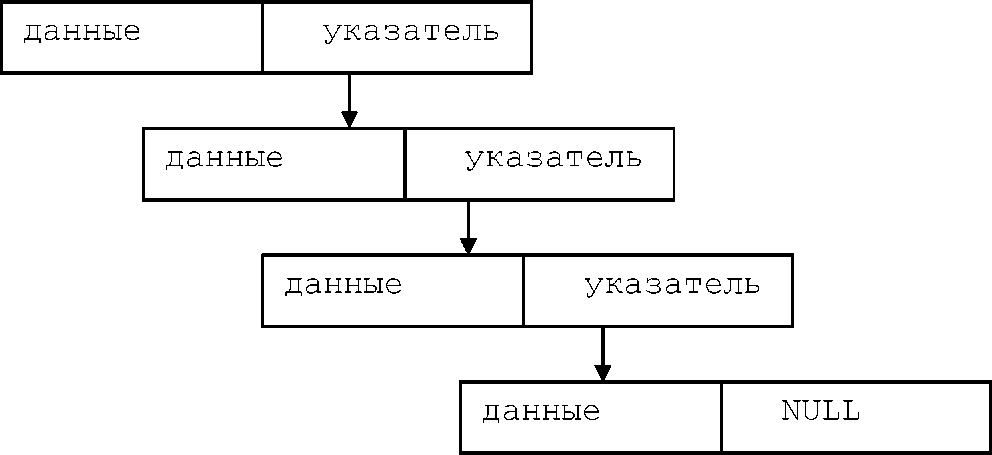
{

элементы\_структуры;

struct имя\_структурного\_типа \*указатель;

}

В каждую структуру такого типа входит указатель на объ­ект того же типа, что и определяемая структура.



beg

Примеры:

Описание структуры struct point

{int key;

point\* next; };

Поле key содержит информационную часть структуры point, а поле next содержит адрес следующего элемента списка.

Функция для формирования однонаправленного списка

point\* make\_point( int n) {

point \*first, \*p; first=NULL;

for (int i=n;i>0;i--) { p=new(point);

p->key=i;

p->next=first;

first=p; }

return first; }

В качестве параметра в функцию передается количество элементов в списке, а результатом является указатель на первый элемент этого списка. Указатель р указывает на вновь создаваемый элемент. Для обращения к полям исполь­зуется операция доступа к элементу структуры, с которой связан указатель -> . Существует вторая возможность обра­щения к полю динамической структуры: (\*p) .key или (\*p).next. В информационное поле key заносится порядковый номер элемента в списке. Добавление новых элементов осу­ществляется в начало списка.

3. Функция для печати однонаправленного списка

point\* print\_point(point\*first) {

if (first==NULL)return NULL; point\*p=first;

while(p!=NULL) {

cout<<p->key<<" ";

p=p->next; }

return first; }

При печати сформированного списка осуществляется проход по списку с помощью вспомогательной переменной р до тех пор, пока она не станет равна NULL.

**2. Постановка задачи**

Написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом.

**Для каждого вариант разработать следующие функции:**

Создание списка.

Добавление элемента в список (в соответствии со своим вариантом).

Удаление элемента из списка (в соответствии со сво­им вариантом).

Печать списка.

Запись списка в файл.

Уничтожение списка.

Восстановление списка из файла.

**3. Порядок выполнения работы**

Написать функцию для создания списка. Функция может создавать пустой список, а затем добавлять в него эле­менты.

Написать функцию для печати списка. Функция должна пре­дусматривать вывод сообщения, если список пустой.

Написать функции для удаления и добавления элементов списка в соответствии со своим вариантом.

Выполнить изменения в списке и печать списка после каж­дого изменения.

Написать функцию для записи списка в файл.

Написать функцию для уничтожения списка.

Записать список в файл, уничтожить его и выполнить пе­чать (при печати должно быть выдано сообщение "Список пустой").

Написать функцию для восстановления списка из файла.

Восстановить список и распечатать его.

Уничтожить список.

**4. Варианты заданий**

14. Записи в линейном списке содержат ключевое поле типа \*char(строка символов). Сформировать двунаправленный список. Удалить из него К элементов с указанными номе­рами. Добавить К элементов с указанными номерами.