

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО МГТУ «СТАНКИН»)**

­­Институт информационных систем и технологий

Кафедра информационных технологий и вычислительных систем

**Курсовая работа**

По дисциплине

«Структуры и алгоритмы обработки данных.»

На тему: «Стек множеств целого типа»

Направление 09.03.01 “Информатика и вычислительная техника”

Направленность “Программное обеспечение средств вычислительной техники и автоматизированных систем”

Руководитель: ст. преп. Лакунина О.Н ФИО

Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_\_\_

Студент, гр. ИДБ-17-02 Кучма Д.И. ФИО

Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_\_\_

Москва 2019г.

**Содержание**

[Задание на курсовую работу 3](#_Toc532422093)

[Описание структуры данных 4](#_Toc532422094)

[Конечная схема реализуемой структуры данных 5](#_Toc532422095)

[Описание структур на языке C 6](#_Toc532422096)

[Схема вызова функций 7](#_Toc532422097)

[Список функций и их назначение 8](#_Toc532422098)

[Функции для работы с динамической матрицей 8](#_Toc532422099)

[Исходный код программы с комментариями 11](#_Toc532422100)

[Файл main.cpp 11](#_Toc532422101)

Задание на курсовую работу

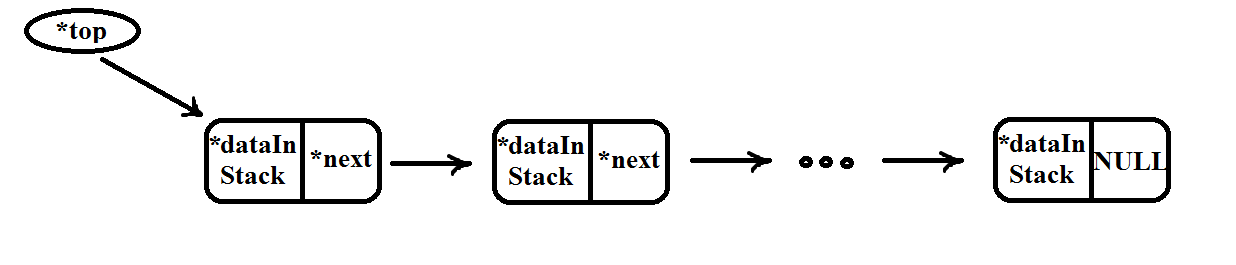
Написать программу реализующую логическую структуру данных – стек множеств целого типа. Программа должна работать в диалоговом режиме. Каждая операция должна быть реализована в отдельной функции. Сделать отчет.

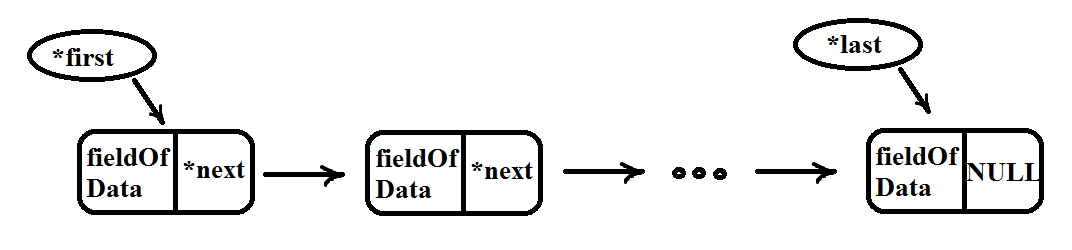
Описание структуры данных

**Стек *–*** логическая структура данных, организованная на базе односвязного списка. Стек, обладает следующими свойствами: работает по принципу LIFO – «последним пришел – первым вышел», имеет ограничение - 5 элементов. **Множество** – логическая структуру данных, организованная на базе односвязного списка. Множество обладает следующими свойствами: элементы множества не повторяются, не упорядочены.

|  |  |
| --- | --- |
| **Название** | **Описание** |
| \*top | Указатель очереди |
| \*dataInStack | Указатель на поле данных множества в стеке |
| \*next | Указатель на следующий элемент стека |
| \*first | Указатель на первый (головной) элемент множества |
| \*last | Указатель на последний элемент множества |
| fieldOfData | Поле данных множества типа int |

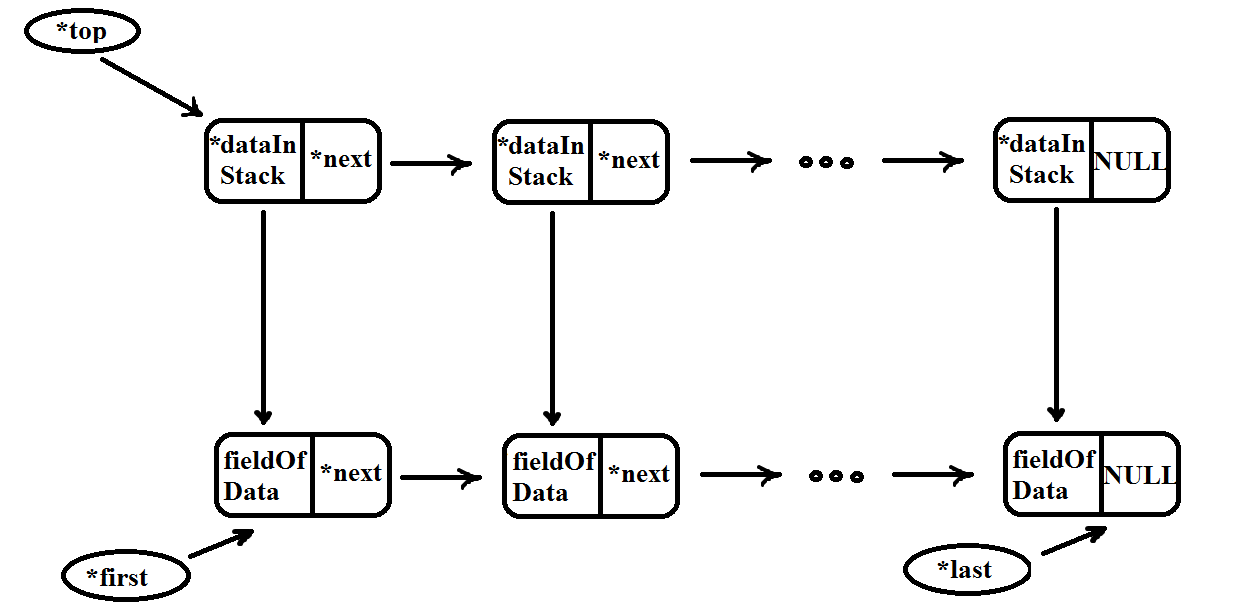
**Односвязный список-стек**

**Односвязный список-множество**

****

Конечная схема реализуемой структуры данных

***Стек множеств целого типа***



Описание структур на языке C

Структура myStack. На физическом уровне данная структура данных является односвязным списком. Содержит указатель на структуру mySet, которая является первым элементом другого односвязного списка, а также указатель на структуру типа myStack. Для пользователя список, состоящий из объектов этой структуры представляет собой элементы стека.

struct myStack{

struct mySet \*dataInStack;

struct myStack \*next;

};

Структура mySet. На физическом уровне данная структура данных является односвязным списком. Содержит переменную типа int, в которой хранится значение элемента в списке, а также указатель на структуру типа mySet. Для пользователя список, состоящий из объектов этой структуры представляет собой множество целого типа.

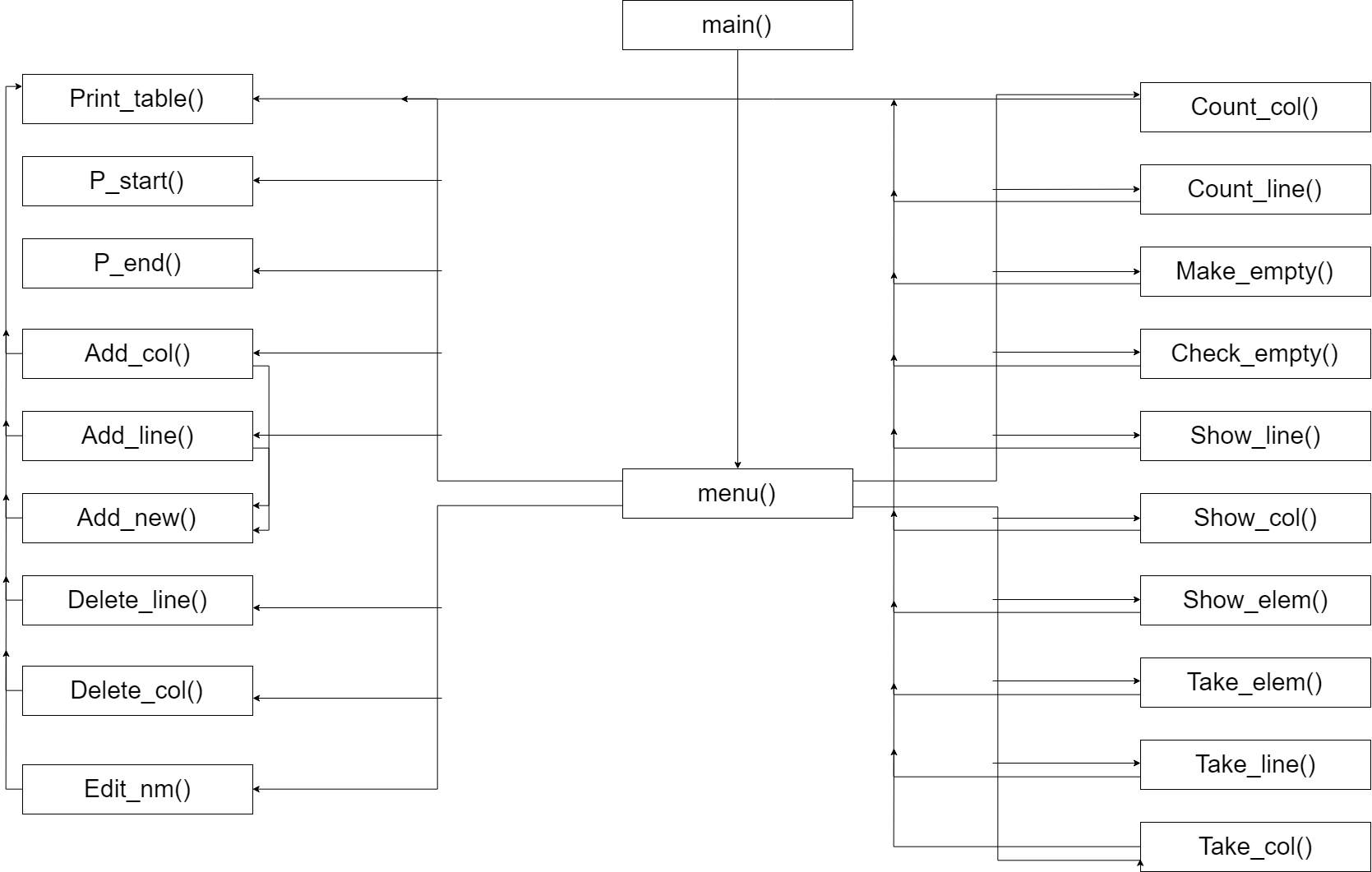
struct mySet{

int fieldOfData;

struct mySet \*next;

};

Схема вызова функций



**Список функций и их назначение****. Функции для работы со стеком:**

struct myStack \*PUSH(struct myStack \*top, struct mySet \*first);

Функция, добавляющая в стек новый элемент (наверх).

struct myStack \*POP(struct myStack \*top);

Функция, удаляющая элемент стека (верхний элемент).

struct myStack \*TakeTop(struct myStack \*top);

Функция взятия верхнего элемента стека. Выводит элемент на экран и удаляет его.

struct myStack \*changeTop(struct myStack \*top, struct mySet \*first);

Функция для изменения значения верхушки стека.

struct myStack \*clearStack(struct myStack \*top);

Функция очистки стека, удаляет все элементы.

bool checkStack(struct myStack \*top);

Функция проверки стека на пустоту.

void displayTop(struct myStack \*top);

Функция вывода верхнего (последнего) элемента стека на экран.

void displayStack(struct myStack \*top);

Функция вывода всех элементов стека на экран. **Для работы с множеством:**

struct mySet \*AddElem(struct mySet \*first, struct mySet \*last, int data);

Функция добавления элемента в множество.

struct mySet \*DelElem(struct mySet \*first, int data);

Функция удаления элемента из множества.

struct mySet \*TakeEl(struct mySet \*first, int data);

Функция взятия элемента из множества. Выводит элемент на экран и удаляет его.

bool checkElem(struct mySet \*first, int data);

Функция проверки на наличие элемента в множестве.

bool checkSet(struct mySet \*first);

Функция проверки множества на пустоту.

void clearSet(struct mySet \*first);

Функция очистки множества, удаляет все элементы.

void display(struct mySet \*first); - Функция вывода элементов множества на экран.

Исходный код программы с комментариями

Файл main.cpp

#include <iostream>  
#include <stdlib.h>  
#include <stdio.h>  
#include <conio.h>  
  
**using namespace** std;  
//Прототипы ф-ий для множества  
**struct** mySet \*AddElem(**struct** mySet \*first, **struct** mySet \*last, **int** data);  
**struct** mySet \*DelElem(**struct** mySet \*first, **int** data);  
**struct** mySet \*TakeEl(**struct** mySet \*first, **int** data);  
**bool** checkElem(**struct** mySet \*first, **int** data);  
**bool** checkSet(**struct** mySet \*first);  
**void** clearSet(**struct** mySet \*first);  
**void** display(**struct** mySet \*first);  
//Прототипы ф-ий для стека  
**struct** myStack \*PUSH(**struct** myStack \*top, **struct** mySet \*first);  
**struct** myStack \*POP(**struct** myStack \*top);  
**struct** myStack \*TakeTop(**struct** myStack \*top);  
**struct** myStack \*clearStack(**struct** myStack \*top);  
**bool** checkStack(**struct** myStack \*top);  
**struct** myStack \*changeTop(**struct** myStack \*top, **struct** mySet \*first);  
**void** displayTop(**struct** myStack \*top);  
**void** displayStack(**struct** myStack \*top);  
//Прототипы ф-ий для меню  
**int** stackMenu(**struct** myStack \*top, **struct** mySet \*first, **struct** mySet \*last);  
**struct** myStack \*setMenu(**struct** myStack \*top, **struct** mySet \*first, **struct** mySet \*last);  
  
**struct** mySet{  
 **int** fieldOfData;//поле данных типа int  
 **struct** mySet \*next;//указатель на следующий элемент множества  
};//структура множество  
**struct** mySet \*AddElem(**struct** mySet \*first, **struct** mySet \*last, **int** data){//функция добавления элемента в множество  
 **struct** mySet \*newElement = NULL, \*result = first;  
 **if** (!checkElem(first, data)) { //Проверка на наличие элемента в множестве  
 newElement = (**struct** mySet \*) malloc(**sizeof**(**struct** mySet)); //Выделение памяти  
 **if** (newElement != NULL) { //Проверка: выделилась ли память  
 **if** (last == NULL) { // Если список пуст  
 newElement->fieldOfData = data;  
 newElement->next = NULL;  
 result = newElement;  
 } **else** {  
 last->next = newElement;  
 newElement->next = NULL;  
 newElement->fieldOfData = data;  
 result = newElement;  
 }  
 }  
 **return** result; // Возвращает указатель на головной элемент нового множества  
 }**else** {  
 cout << "Element is already in set:" <<endl; //Если элемент уже есть, выводит сообщение об ошибке  
 **return** last; // Возвращает указатель на последний элемент  
 }  
  
}  
**struct** mySet \*DelElem(**struct** mySet \*first, **int** data){ //Функция удаления элемента из множества  
 **struct** mySet \*result = first, \*savePtr = NULL, \*currentElement = first;  
 **if** (!checkSet(first)) { //Если множество не пусто  
 **if** (checkElem(first, data)) { //Если такой элемент есть в множестве  
 **while** (currentElement->next != NULL) { //Цикл до тех пор, пока указатель на след элемент!=NULL  
 **if** (currentElement->fieldOfData == data) {//Если поле данных текущего элемента списка==искомому значению  
 **if** (savePtr == NULL) {  
 result = currentElement->next;  
 free(currentElement); //Очищает текущий элемент  
 **return** result; //Возвращает указатель на след. элемент  
 } **else** {  
 savePtr->next = currentElement->next;  
 free(currentElement);  
 **return** result;  
 }  
 }  
 savePtr = currentElement;//Сохраняем указатель на текущий элемент  
 currentElement = currentElement->next;  
 }  
 **if** (currentElement->fieldOfData == data) {//Если поле данных текущего элемента списка==искомому значению  
 **if** (savePtr == NULL) {  
 result = currentElement->next;  
 free(currentElement);  
 **return** result;  
 } **else** {  
 savePtr->next = currentElement->next;  
 free(currentElement);  
 **return** result;  
 }  
 }  
 }**else** cout << "Element is not found" <<endl; //Если такого элемента нет в множестве, выводится ошибка  
 } **else** cout << "Set is empty" <<endl; //Если множество пусто, выводится ошибка  
 **return** result;  
}  
**struct** mySet \*TakeEl(**struct** mySet \*first, **int** data){ //Функция взятия элемента множества  
 **struct** mySet \*result = first, \*savePtr = NULL, \*currentElement = first;  
 **if** (!checkSet(first)) {//Если множество не пусто  
 **while** (currentElement->next != NULL) {//Цикл до тех пор, пока указатель на след элемент!=NULL  
 **if** (currentElement->fieldOfData == data) { //Если поле данных текущего элемента списка==искомому значению  
 **if** (savePtr == NULL) {  
 result = currentElement->next;  
 cout << currentElement->fieldOfData << endl;  
 free(currentElement);//Очищаем текущий элемент  
 **return** result; //Возвращает указатель на след. за текущим элемент  
 } **else** {  
 savePtr->next = currentElement->next;  
 cout << currentElement->fieldOfData << endl;  
 free(currentElement);  
 **return** result;  
 }  
 }  
 savePtr = currentElement; //Сохраняем текущий элемент  
 currentElement = currentElement->next;  
 }  
 **if** (currentElement->fieldOfData == data) {//Если поле данных текущего элемента списка==искомому значению  
 **if** (savePtr == NULL) {  
 result = currentElement->next;  
 cout << currentElement->fieldOfData << endl;  
 free(currentElement);  
 **return** result;  
 } **else** {  
 savePtr->next = currentElement->next;  
 cout << currentElement->fieldOfData << endl;  
 free(currentElement);  
 **return** result;  
 }  
 }  
 } **else** cout << "Set is empty" <<endl;  
 **return** result;  
}  
**bool** checkElem(**struct** mySet \*first, **int** data){//Функция проверки наличия элемента в множестве  
 **if** (!checkSet(first)) {//Если множество не пусто  
 **struct** mySet \*currentElement = first;  
 **if** (first != NULL) {//Если первый элемент!=NULL  
 **while** (currentElement->next != NULL) {//Цикл до тех пор, пока указатель на след элемент!=NULL  
 **if** (currentElement->fieldOfData == data)//Если поле данных текущего элемента списка==искомому значению  
 **return true**;//возвращает true  
 currentElement = currentElement->next;//Иначе переходит на следующий элемент  
 }  
 **if** (currentElement->fieldOfData == data)  
 **return true**;  
 }  
 }  
 **return false**;  
}  
**bool** checkSet(**struct** mySet \*first){//Функция проверки: множество пусто или нет  
 **if** (first == NULL)//Если головной элемент множества указывает на NULL  
 **return true**;//Вернуть true  
 **else  
 return false**;  
}  
**void** clearSet(**struct** mySet \*first){//функция очистки множества  
 **struct** mySet \*savePtr = NULL, \*currentElement = first;  
 **if**(!checkSet(first)) {//Если множество не пусто  
 **while** (currentElement->next != NULL) {//Цикл до тех пор, пока указатель на след элемент!=NULL  
 savePtr = currentElement->next;//Сохраняем указатель на следующий за текущим элемент  
 free(currentElement);//Очищаем текущий  
 currentElement = savePtr;//Присваиваем текущему сохраненный указатель на следующий  
 }  
 free(currentElement);  
 cout << "Set is cleared" <<endl;//Выводим "множество очищено"  
 }**else** cout << "Set is already empty"<<endl;//Выводим "множество уже пусто"  
  
}  
**void** display(**struct** mySet \*first){//Функция вывода множества на экран  
 **struct** mySet \*CurrentElem = first;  
 **if** (!checkSet(first)) {//Если множество не пусто  
 **if** (CurrentElem != NULL) {//Если текущий элемент!=NULL  
 **while** (CurrentElem->next != NULL) {//Цикл до тех пор, пока указатель на след элемент!=NULL  
 cout << CurrentElem->fieldOfData << " ";//Выводим поле данных текущего элемента множества  
 CurrentElem = CurrentElem->next;//Присваиваем указателю на текущий указатель на следующий  
 }  
 cout << CurrentElem->fieldOfData << " " << endl;//Выводим последний элемент, находящийся вне цикла  
 }  
 }**else** cout<<"Set is empty"<<endl;//Иначе распечатать "Множество пусто"  
}  
  
**struct** myStack{  
 **struct** mySet \*dataInStack;//указатель на поле данных структуры множество  
 **struct** myStack \*next;//указатель на следующий элемент стека  
};//структура стек  
**int** counter = 0;//счетчик элементов стека  
**struct** myStack \*PUSH(**struct** myStack \*top, **struct** mySet \*first){//Функция добавления элемента на верхушку стека  
 **struct** myStack \*newElement = NULL, \*result = top;  
 newElement = (**struct** myStack \*)malloc(**sizeof**(**struct** myStack));//Выделение памяти  
 **if** (newElement != NULL){//Если память выделилась  
 **if** (top == NULL){//Если верхний элемент пуст  
 newElement->dataInStack = first;  
 newElement->next = NULL;  
 result = newElement;//Возвращаем новый указатель на верхушку стека  
 }**else**{//Если он уже существует  
 newElement->dataInStack = first;  
 newElement->next = result;  
 result = newElement;  
 }  
 }  
 counter++;//Увеличить значение счетчика  
 **return** result;  
}  
**struct** myStack \*POP(**struct** myStack \*top){//Функция удаления верхушки стека  
 **struct** myStack \*result = NULL;  
 **if** (!checkStack(top)) {//Если стек не пуст  
 **struct** myStack \*currentElement = top;  
 **if**(top->next != NULL) {//Если следующий за верхушкой элемент существует  
 result = currentElement->next;//Присваиваем указатель на следующий  
 free(currentElement);//Очищаем текущий  
 counter--;//Уменьшаем значение счетчика  
 **return** result;//Возвращаем указатель на следующий  
 }**else**{//Если не существует  
 free(currentElement);//Очищаем текущий  
 counter--;//Уменьшаем значение счетчика  
 **return** result;//Возвращаем NULL  
 }  
 }  
 **return** result;//Возвращаем NULL  
}  
**struct** myStack \*TakeTop(**struct** myStack \*top) {//Функция взятия элемента из множества  
 **struct** myStack \*result = NULL;  
 **if** (!checkStack(top)) {//Если стек не пуст  
 **struct** myStack \*currentElement = top;//Указатель на текущий = указатель на верхушку  
 **if**(top->next != NULL) {//Если верхушка не единственна  
 result = currentElement->next;//Присваиваем следующий  
 displayTop(currentElement);//Выводим текущий  
 free(currentElement);//Очищаем текущий  
 counter--;//Уменьшаем значение счетчика  
 **return** result;//Возвращаем указатель на следующий элемент  
 }**else**{//Если верхний элемент последний  
 free(currentElement);//Очищаем текущий  
 counter--;//Уменьшаем значение счетчика  
 cout << "Stack is cleared"<<endl;//Выводим сообщение о полной очистке стека  
 **return** result;//Возвращаем NULL  
 }  
 }  
 **return** result;//Возвращаем NULL  
}  
**struct** myStack \*changeTop(**struct** myStack \*top, **struct** mySet \*first){//Функция для замены элемента стека  
 **struct** myStack \*result = NULL;  
 **if** (!checkStack(top)) {//Если стек не пуст  
 **struct** myStack \*currentElement = top;  
 **if**(top->next != NULL) {//Если верхушка не единственный элемент в стеке  
 result = currentElement->next;  
 free(currentElement);  
 counter--;  
 **return** result;  
 }**else**{  
 free(currentElement);  
 counter--;  
 **return** result;  
 }  
 }  
 **return** result;  
}  
**struct** myStack \*clearStack(**struct** myStack \*top){//Функция очистки стека  
 **struct** myStack \*currentElement = top, \*savePtr = NULL;  
 **if** (!checkStack(top)) {//Если стек не пуст  
 **while** (currentElement->next != NULL) {//Цикл до тех пор, пока указатель на след элемент!=NULL  
 savePtr = currentElement->next;//Сохраняем указатель на след за текущим  
 free(currentElement);//Очищаем текущий  
 currentElement = savePtr;//Присваиваем текущему сохраненный  
 }  
 free(currentElement);//Очищаем текущий  
 currentElement = NULL;//Присваиваем ему NULL  
 cout << "Stack is cleared" <<endl;//Выводим сообщение о полной очистке стека  
 }**else** cout <<"Stack is already empty"<<endl;//Ошибка: стек уже пуст  
 **return** currentElement;//возвращает указатель на top  
}  
**bool** checkStack(**struct** myStack \*top){//Функция проверки стека на пустоту  
 **if** (top == NULL)//Если верхушка стека пуста  
 **return true**;//Возвращает true  
 **else  
 return false**;  
}  
**void** displayTop(**struct** myStack \*top) {//Функция вывода верха стека  
 **if** (!checkStack(top)) {//Если стек не пуст  
 **struct** myStack \*stackElement = top; //присваиваем указатель на верх стека  
 **struct** mySet \*setElement = top->dataInStack;//присваиваем указатель на поле данных  
 **while** (setElement->next != NULL) {//Цикл до тех пор, пока указатель на след элемент!=NULL  
 cout << setElement->fieldOfData << " ";//Вывод поля данных текущего элемента  
 setElement = setElement->next;//Указатель на текущий = указатель на следующий  
 }  
 cout << setElement->fieldOfData << endl; //Вывод поля данных последнего  
 }**else** cout <<"Stack is empty"<<endl; // Иначе ошибка - стек пуст  
}  
  
  
**void** displayStack(**struct** myStack \*top){ //Функция вывода всего стека на экран  
 **if**(!checkStack(top)){//Если стек не пуст  
 **struct** myStack \*stackElement = top; //Указатель на верх стека  
 **struct** mySet \*setElement = top->dataInStack; //\*верхстека->полеДанных  
 cout << "Stack:" <<endl;  
 **while** (stackElement->next != NULL){//Цикл до тех пор, пока указатель на след элемент!=NULL  
 **while** (setElement->next != NULL){//Цикл до тех пор, пока указатель на след элемент!=NULL  
 cout << setElement->fieldOfData << " "; //Вывод поля данных текущего элемента  
 setElement = setElement->next; //Текущий = след  
 }  
 cout << setElement->fieldOfData <<endl; // Вывод поля данных текущего  
 stackElement = stackElement->next; // Текущий = след  
 setElement = stackElement->dataInStack;  
 }  
 **while** (setElement->next != NULL){  
 cout << setElement->fieldOfData << " ";  
 setElement = setElement->next;  
 }  
 cout << setElement->fieldOfData <<endl;  
 }**else** cout <<"Stack is empty"<<endl; // Иначе вывести ошибку - стек пуст  
}  
  
**int** main() {  
 **struct** mySet \*first = NULL, \*last = NULL; // Создаем указатель на первый(головной) и последний элементы множества  
 **struct** myStack \*top = NULL; //Создаем указатель на верхний элемент стека. Все вышеперечисленные = NULL  
 stackMenu(top, first, last); //Вызов функции меню стека  
 **return** 0;  
}  
**int** stackMenu(**struct** myStack \*top, **struct** mySet \*first, **struct** mySet \*last){ //Функция вызова меню стека  
 **int** input; //Вводимый пользователем номер строки  
 **bool** flag = **false**; //Флаг для запуска/остановки программы  
 **do** {  
 cout << " 1.Start (stack)" << endl;  
 cout << " 2.Clear stack" << endl;  
 cout << " 3.Check stack for emptiness" << endl;  
 cout << " 4.Display top of stack" << endl;  
 cout << " 5.Remove top of stack" << endl;  
 cout << " 6.Take an element from stack" << endl;  
 cout << " 7.Change top of stack" << endl;  
 cout << " 8.Add an element into stack" << endl;  
 cout << " 9.Display stack" << endl;  
 cout << " 10.Finish (stack)" << endl;  
 cin >> input;  
 **switch** (input) {  
 **case** 1:  
 flag = **true**;  
 cout << "Stack is started" << endl;  
 **break**;  
 **case** 2:  
 **if** (flag) {  
 top = clearStack(top);  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 3:  
 **if** (flag) {  
 **if** (checkStack(top))  
 cout << "Stack is empty" << endl;  
 **else** cout << "Stack is not empty" << endl;  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 4:  
 **if** (flag) {  
 displayTop(top);  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 5:  
 **if** (flag) {  
 **if** (!checkStack(top)) {  
 top = POP(top);  
 **if** (top != NULL)  
 first = top->dataInStack;  
 **else** first = NULL;  
 cout << " Top is removed from stack" << endl;  
 displayStack(top);  
 }**else** cout << "Stack is empty" << endl;  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 6:  
 **if** (flag) {  
 **if** (!checkStack(top)) {  
 top = TakeTop(top);  
 displayStack(top);  
 } **else** cout << "Stack is empty" <<endl;  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 7:  
 **if** (flag){  
 **if** (!checkStack(top)) {  
 top = changeTop(top,first); //вызов функции изменения верха  
 top = setMenu(top, first, last); // функция меню множества  
 first = NULL;  
 last = NULL;  
 }**else** cout << "Stack is empty" <<endl;  
 }**else** cout << "Start, please"<<endl;  
 **break**;  
 **case** 8:  
 **if**(flag){  
 **if** (counter<5) {//Если кол-во элементов удовлетворяет ограничению  
 top = setMenu(top, first, last); //функция меню множества  
 }**else** cout <<"Stack is full"<<endl;  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 9:  
 **if** (flag) {  
 displayStack(top);  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 10:  
 flag = **false**;  
 //Очистка всех указателей:  
 free(top);  
 free(first);  
 free(last);  
 **return** 0;  
 }  
 }  
 **while**(input != 20);  
 **return** 0;  
}  
**struct** myStack \*setMenu(**struct** myStack \*top, **struct** mySet \*first, **struct** mySet \*last){ //Функция меню множества  
 **int** input, data; //Значение пункта меню и данных  
 **bool** flag = **false**;  
 **do** {  
 cout << " 1.Start (set)" << endl;  
 cout << " 2.Clear set" << endl;  
 cout << " 3.Check set for emptiness" << endl;  
 cout << " 4.Remove an element from set" << endl;  
 cout << " 5.Take an element from set" << endl;  
 cout << " 6.Add an element <N> into set" << endl;  
 cout << " 7.Check an element" << endl;  
 cout << " 8.Display set" << endl;  
 cout << " 9.Done" << endl;  
 cin >> input;  
 **switch** (input) {  
 **case** 1:  
 flag = **true**;  
 cout << "Set is started" << endl;  
 **break**;  
 **case** 2:  
 **if** (flag) {  
 clearSet(first);  
 first = NULL;  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 3:  
 **if** (flag) {  
 **if**(checkSet(first))  
 cout << "Set is empty" <<endl;  
 **else** cout << "Set is not empty" <<endl;  
  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 4:  
 **if** (flag) {  
 **if**(!checkSet(first)) {  
 cout << "Enter an element, please" << endl;  
 cin >> data;  
 first = DelElem(first, data);  
 display(first);  
 }**else** cout << "Set is empty"<< endl;  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 5:  
 **if** (flag) {  
 **if** (!checkSet(first)){  
 cout << "Enter an element, please" << endl;  
 cin >> data;  
 first = TakeEl(first, data);  
 display(first);  
 } **else** cout << "Set is empty"<<endl;  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 6:  
 **if** (flag) {  
 cout << "Enter an element, please" << endl;  
 cin >> data;  
 **if** (first == NULL) {//Если первый элемент добавляем(инициализируем множество)  
 first = AddElem(first, last, data);  
 last = first;  
 } **else** last = AddElem(first, last, data);  
 display(first);  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 7:  
 **if** (flag) {  
 cout << "Enter an element, please" << endl;  
 **if**(!checkSet(first)) {  
 cin >> data;  
 **if** (checkElem(first, data))  
 cout << "Element is found" << endl;  
 **else** cout << "Element is not found" << endl;  
 display(first);  
 } **else** cout << "Set is empty" <<endl;  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 8:  
 **if** (flag) {  
 display(first);  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 **case** 9:  
 **if** (flag) {  
 **if** (!checkSet(first)) {  
 top = PUSH(top, first); //Отправка множества в стек  
 first = NULL;  
 last = NULL;  
 input = 10;  
 cout << "Set has been pushed into stack" << endl;  
 } **else** {  
 input = 10;  
 cout << "Set is empty" << endl;  
 }  
 } **else** cout << "Start, please" << endl;  
 **break**;  
 }  
 }**while**(input != 10);  
 **return** top;  
  
}