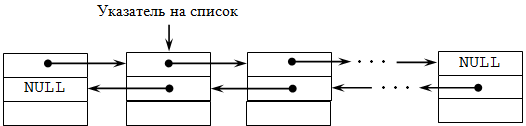
* **Списком** называется упорядоченное множество, состоящее из переменного числа элементов, к которым применимы **операции включения**, исключения. **Список**, отражающий отношения соседства между элементами, называется линейным.

**Однонаправленный (односвязный) список** – это **структура данных**, представляющая собой последовательность элементов, в каждом из которых хранится **значение** и **указатель** на следующий **элемент списка**

**Двунаправленный (двусвязный) список** – это **структура данных**, состоящая из последовательности элементов, каждый из которых содержит информационную часть и два указателя на соседние элементы



#include "pch.h"

#include <iostream>

//структура узлов

typedef struct Node

{

int value; //значение в узле

struct Node \*next, \*pred; //указатели на элементы структуры

} NODE, \*pNODE;

Здесь мы задаем структуру узлов и значения

pLIST createList() {

pLIST pL;

pL = (pLIST)malloc(sizeof(LIST));

if (!pL) return NULL;

pL->top = NULL;

pL->len = 0;

return pL;

В этой части кода мы выделяем динамическую память для хранения списка

int isEmpty(pLIST pL) {

if (pL->top&&pL->len) return 0;

return 1;

}

проверка списка на наличие элементов

pNODE getPointer(pLIST pL, int date) {

pNODE temp;

if (isEmpty(pL)) return NULL;

else {

temp = pL->top;

if (date >= temp->value) return temp;

while (temp->next && temp->next->value > date) temp = temp->next;

return temp;

}

}

определение места вставки следующего узла

int addNodeAfter(pLIST pL, pNODE pN, int newdate) {

pNODE pnew = (pNODE)malloc(sizeof(NODE));

if (pnew) {

pL->len++;

pnew->value = newdate;

if (isEmpty(pL) || newdate >= pL->top->value)

{

pnew->next = pL->top;

pnew->pred = NULL;

if (!isEmpty(pL)) pL->top->pred = pnew;

pL->top = pnew;

return 1;

}

if (!(pN->next)) {

pnew->next = pN->next;

pnew->pred = pN;

pN->next = pnew;

return 1;

}

pN->next->pred = pnew;

pnew->next = pN->next;

pnew->pred = pN;

pN->next = pnew;

return 1;

}

return 0;

}

добавление узла

pNODE findNode(pLIST pL, int date) {

pNODE p = pL->top, temp = p;

//if (isempty(pL)) return NULL;

while (temp&&temp->value != date) {

p = temp;

temp = temp->next;

}

if (temp) return p;

return NULL;

}

нахождение узла в списке

int delNode(pLIST pL, pNODE pN) {

if (!pN) return 0;

pNODE temp = pN->next;

pN->next = temp->next;

free(temp);

pL->len--;

return 1;

}

удаление узла

void clearList(pLIST pL) {

if (!isEmpty(pL)) {

while (pL->top->next)

{

delNode(pL, pL->top);

}

pL->len--;

free(pL->top);

}

}

очищение памяти отведенной для списка

void deleteList(pLIST pL) {

clearList(pL);

free(pL);

printf("List delete!\n");

}

Удаление

void showList(pLIST pL) {

pNODE temp = pL->top;

if (isEmpty(pL)) printf("List is empty!\n");

else

{

printf("Length list:%d\n", pL->len);

while (temp) {

printf\_s("adres: %p \t element: %d \t next:%p\n", temp, temp->value, temp->next);

temp = temp->next;

}

}

}

вывод списка