Примеры программ для работы со стеком, очередью и списком

Задание: Создать стек, состоящий из чисел. В новом стеке после каждого максимального элемента вставить новый элемент. Вывести элементы стека в том же порядке, что и вводили

```
//result1 - сначала отрицательные, потом положительные
#include<iostream>
using namespace std;
//описание структуры стек
struct stack {
        int inf;
        stack *next;
};

//функция вставки элемента x в стек h
 void push(stack *&h, int x) {
        stack *r = new stack;
        r->inf = x;
        r->next = h;
        h = r;
       }
 //функция извлекает из стека h первый элемент и возвращает его в качестве результата
int pop(stack *&h) {
       int i = h->inf;
       stack *r = h;
       h = h->next;
       delete r;
       return i;
}
//функция, которая меняет порядок элементов в стеке на обратный
//если в h были элементы 1, 2, 3
//то после выполнения функции в h будут элементы 3, 2, 1
void reverse(stack *h) {
        stack *head1 = NULL; //инициализация буферного стека
        while (h) //пока стек не пуст
               push(head1, pop(h)); //переписываем элементы из одного стека в другой
        h = head1; //переобозначаем указатели
}

//функция возвращает значение максимального элемента в стеке h
int stack_max(stack *h) {
       stack *temp = NULL;//инициализация вспомогательного стека
       int max = pop(h), x; //в max кладем первый элемент стека h
       push(temp, max); //первый элемент стека h перекладываем в стек temp
       while (h) {//пока стек не пуст
              x = pop(h);//извлекаем первый элемент из стека h в переменную h
              if (x > max) //если этот элемент x больше max,
                     max = x; // то запомним его в max
              push(temp, x);//кладем извлеченный элемент в temp
       }
       reverse(temp);//меняем порядок элементов в  стеке temp
       h = temp;//h будет указывать на temp
       return max;
}

//функция, которая будет выполнять задание
stack *result(stack *h)
// после максимального вставляет новый элемент
{
       int max = stack_max(h);//ищем максимальный элемент
       stack *temp = NULL;//инициализация вспомогательного стека
       int new_el, x;
       cout << "new element = ";
```

```
	cin >> new_el;//вводим новый элемент, который будем вставлять
	//т.к. в h элементы будут записаны в обратном порядке
	//ввели 1, 2, 3
	//в стеке h будут записаны элементы в порядке 3 2 1
	//для того чтобы в temp новый элемент стоял после максимального,
	//его в temp надо добавлять перед максимальным
	while (h) {//пока стек не пуст
		x = pop(h);//извлекаем первый элемент из стека h в переменную h
		if (x == max) //если этот элемент x равен max
			push(temp, new_el);//то добавляем новый элемент new_el в стек temp
		push(temp, x);//добавляем x в стек temp
	}
	//после цикла в temp получится 1 2 3 <new_el>
	return (temp); //возвращаем temp как результат функции
}
int main() {
	stack *h = NULL;
	int n, x;
	cout << "n="; cin >> n;//вводим n количество элементов в стеке
	cout << "Input element\n";
	for (int i = 0; i < n; i++) {
		cin >> x;//вводим число
		push(h, x);//добавляем число в стек h
	}
	h = result(h); //вызываем функцию result
	while (h){
		cout << pop(h) << " ";//выводим элементы из стека на экран
	}
	return 0;
}
```

Задание: Создать очередь из целых чисел. В новую очередь вывести сначала нечетные, а потом четные числа

```
#include<iostream>
using namespace std;
//создаем структуру очередь
struct queue {
	int inf;
	queue *next;
};

//функция добавления элемента x в конец очереди,
//h - указатель на начало, t-указатель на конец
void push(queue *&h, queue*&t, int x) {
	queue *r = new queue; //создаём новый элемент
	r->inf = x;//заполняем его информационное поле
	r->next = NULL;//заполняем ссылочное поле
	if (!h && !t) { //если очередь пуста
		h = t = r; //это голова и хвост ссылаются на этот новый элемент r
	}
	else {
		t->next = r; //r будет следующим элементом после хвоста t
		t = r;//переносим указатель хвоста t на r
	}
}

//удаляет первый элемент из очереди и возвращает его значение
int pop(queue *&h, queue *&t) {
	int i = h->inf; //значение информационного поля головы h, т.е. первого элемента
	queue *r = h;//создаём указатель на голову
	h = h->next;//переносим указатель h на следующий элемент
	if (!h) //если h=NULL, т.е. в очереди был один элемент
```

```
			t = NULL;//то и хвост станет NULL
		delete r; //удаляем первый элемент
		return i;//возвращаем значение
}

//функция меняет очередь так, чтобы в ней были сначала нечетные, потом четные элементы
void result(queue *&h, queue *&t) {
		queue *h_nech = NULL, *t_nech = NULL; //инициализируем очередь,
//в которую потом запишем нечетные элементы
		queue *h_ch = NULL, *t_ch = NULL;//инициализируем очередь,
//в которую потом запишем четные элементы
		int x;
		while (h && t) {//пока исходная очередь не пуста
			x = pop(h, t);//извлекаем элемент из очереди
			if (x % 2 == 0)//если элемент четный,
				push(h_ch, t_ch, x);// то добавляем его в очередь для четных
			else push(h_nech, t_nech, x);//иначе добавляем его в очередь для нечетных
		}
//h и t делаем равными указателям на начало и конец очереди с нечетными
		h = h_nech; t = t_nech;
		while (h_ch && t_ch) {//пока очередь с четными не пуста
			push(h, t, pop(h_ch, t_ch));//переписываем четный элемент в очередь h, t
		}
}

int main(){
			queue *h = NULL, *t = NULL;//инициализируем очередь
			int n, x;
			cout << "n="; cin >> n;//n - количество элементов в очереди
			cout << "Input element\n";
			for (int i = 0; i < n; i++) {
				cin >> x;//вводим число
				push(h, t, x);//добавляем число в очередь
			}
			//вызываем функцию, которая меняет очередь в соответствии с заданием
			result(h, t);		cout << "New queue\n";
			while (h && t ) //пока очередь не пуста
				cout << pop(h, t) << " ";// выводим элемент очереди на экран
			return 0;
}
```

Задание: Создать двусвязный список, содержащий целые числа. Удалить все четные числа.

```
#include <iostream>
using namespace std;
struct list {
		int inf;
		list *next;
		list *prev;
};
//вставка элемента в конец списка
void push(list *&h, list *&t, int x) {
		list *r = new list; //создаём новый элемент
		r->inf = x;//заполняем информационное поле нового элемента
		r->next = NULL; //заполняем ссылочное поле нового элемента
		if (!h && !t) {  //если список пуст
			r->prev = NULL; //r будет первым элементом
			h = r;//голова p будет ссылаться на r
		}
		else {
			t->next = r; //следующий для хвоста для хвоста - это r
			r->prev = t;//предыдущим элементом для r будет t
		}
		t = r; //хвост t будет указывать на r
```

```
}

//функция удаляет из списка элемент, на который ссылается r
void del_node(list *&h, list *&t, list *r) {
	if (r == h && r == t) //если в списке один элемент
		h = t = NULL; //то список будет пустым
	else if (r == h) { //если надо удалить первый элемент в списке
		h = h->next; //то сдвигаем голову h на один элемент
		h->prev = NULL;
	}
	else if (r == t) { //если удаляем последний элемент
		t = t->prev; //сдвигаем хвост на один элемент влево
		t->next = NULL;
	}
	else {//случай когда удаляемый элемент в середине списка
		r->next->prev = r->prev;
		r->prev->next = r->next;
	}
	delete r; //удаляем r
}

//вывод элементов списка
void print(list *&h, list *&t) {
	list *p = h;//указатель на голову
	while (p) { //пока не дошли до конца
		cout << p->inf << " ";//выводим
		p = p->next;//переход к следующему
	}
}

//функция для изменения списка в соответствии с заданием
void result(list *&h, list *&t) {
	list *p = h;//указатель на голову
	while (p) {//пока не дошли до конца списка
		list *k = p->next; //сохраняем указатель на следующий элемент
		if (p->inf % 2 == 0) //если элемент чётный
			del_node(h, t, p); //удаляем его
		p = k;//присваиваем p следующий элемент
	}

}

int main() {
	int n, y;
	//инициализируем список
	list *h = NULL;
	list *t = NULL;
	cout << "n= ";
	cin >> n;//вод количества элементов
	for (int i = 0; i < n; i++) { //ввод самих элементов
		cin >> y;
		push(h, t, y); //добавление элемента в стек
	}
	result(h, t);
	print(h, t);//выводим полученный список
	return 0;
}
```