Практическое занятие 14
«Динамические структуры данных. Списки»

Краткие теоретические сведения

Абстрактные структуры данных предназначены для удобного хранения и доступа к информации. Они предоставляют удобный интерфейс для типичных операций с хранимыми объектами, скрывая детали реализации от пользователя. Это весьма удобно и позволяет добиться большей модульности программы. Абстрактные структуры данных иногда делят на две части: *интерфейс*, набор операций над объектами, который называют АТД (*абстрактный тип данных*) и *реализацию*. Языки программирования высокого уровня (предоставляют удобный интерфейс для чисел: операции +, *, = .. и т.п., но при этом скрывают саму реализацию этих операций, машинные команды.

Статические структуры относятся к разряду непримитивных структур, которые, фактически, представляют собой структурированное множество примитивных, базовых, структур. Например, вектор может быть представлен упорядоченным множеством чисел. Поскольку по определению статические структуры отличаются отсутствием изменчивости, память для них выделяется один раз и ее объем остается неизменным до уничтожения структуры.

Динамические структуры по определению характеризуются отсутствием физической смежности элементов структуры в памяти, непостоянством и непредсказуемостью размера (числа элементов) структуры в процессе ее обработки.

Поскольку элементы динамической структуры располагаются по непредсказуемым адресам памяти, адрес элемента такой структуры не может быть вычислен из адреса начального или предыдущего элемента. Для установления связи между элементами динамической структуры **используются указатели**, через которые устанавливаются явные связи между элементами. Такое представление данных в памяти называется связным. Элемент динамической структуры состоит из двух полей:

- *информационного поля* или *поля данных*, в котором содержатся те данные, ради которых и создается структура; в общем случае информационное поле само является интегрированной структурой - вектором, массивом, другой динамической структурой и т.п.;
- *поле связок*, в котором содержатся один или несколько указателей, связывающий данный элемент с другими элементами структуры.

Когда связное представление данных используется для решения прикладной задачи, для конечного пользователя «видимым» делается только содержимое информационного поля, а поле связок используется только программистом-разработчиком.

Достоинства связного представления данных − в возможности обеспечения значительной изменчивости структур;

- размер структуры ограничивается только доступным объемом машинной памяти;
- при изменении логической последовательности элементов структуры требуется не перемещение данных в памяти, а только коррекция указателей;
- большая гибкость структуры.

Вместе с тем связное представление не лишено и *недостатков*, основные из которых:

- на поля связок расходуется дополнительная память;
- доступ к элементам связной структуры может быть менее эффективным по времени.

Последний недостаток является наиболее серьезным и именно им ограничивается применимость связного представления данных. Поэтому связное представление практически никогда не применяется в задачах, где логическая структура данных имеет вид вектора или массива - с доступом по номеру элемента, но часто применяется в задачах, где логическая структура требует другой исходной информации доступа (таблицы, списки, деревья и т.д.).

Списком называется упорядоченное множество, состоящее из переменного числа элементов, к которым применимы операции включения, исключения. Список, отражающий отношения соседства между элементами, называется *линейным*. *Длина списка* равна числу элементов, содержащихся в списке, список нулевой длины называется пустым списком. *Линейные связные списки являются простейшими динамическими структурами данных.*

Графически связи в списках удобно изображать с помощью стрелок. Если компонента не связана ни с какой другой, то в поле указателя записывают значение, не указывающее ни на какой элемент. Такая ссылка обозначается специальным именем - nil.

На рис. 1 приведена структура *односвязного списка*. На нем поле INF - информационное поле, данные, NEXT - указатель на следующий элемент списка. Каждый список должен иметь особый элемент, называемый указателем начала списка или головой списка, который обычно по формату отличен от остальных элементов. В поле указателя последнего элемента списка находится специальный признак nil, свидетельствующий о конце списка.

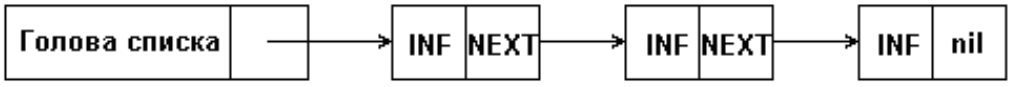

Рис. 1. Представление односвязного списка в памяти

Двусвязный список характеризуется наличием пары указателей в каждом элементе: на предыдущий элемент и на следующий (рис. 2).

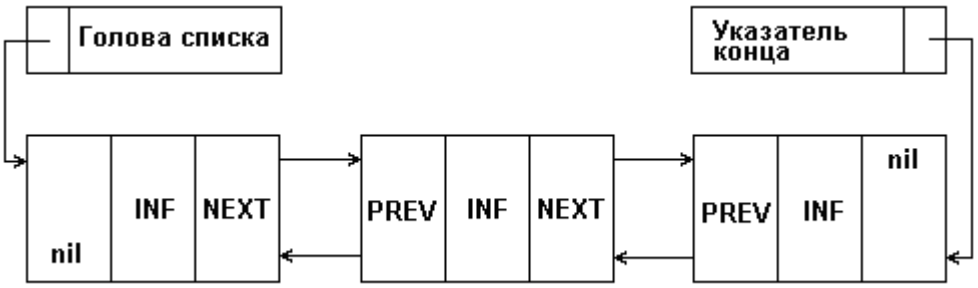

Рис. 2. Представление двусвязного списка в памяти

Очевидный плюс в том, что от данного элемента структуры мы можем пойти в обе стороны. Таким образом упрощаются многие операции. Однако на указатели тратится дополнительная память.

Выделим *типовые операции* над списками:

- добавление узла в список;
- удаление узла из списка;
- проверка, пуст ли список;
- очистка списка;
- печать списка.

Программная реализация списка.

На примере двусвязного списка, разберем принцип работы этой структуры данных. При реализации списка удобно использовать структуры (рис. 3).

```
struct DoubleList //описание узла списка
{
    int data; //информационное поле
    DoubleList *next; //указатель на следующий элемент
    DoubleList *prev; //указатель на предыдущий элемент
};
DoubleList *head; //указатель на первый элемент списка
```

Рисунок 3. Описание узла двунаправленного связного списка и указателя на первый элемент списка

Функции обработки связного списка.

Добавление элемента.

Опишем функцию AddList, которая в качестве параметров принимает значение и адрес будущего узла, после чего создает его в списке:

Вывод списка.

Функция PrintList выводит на экран все элементы списка:

```
#include "pch.h"
#include <iostream>
using namespace std;
struct DoubleList //описание узла списка
{
	int data; //информационное поле
	DoubleList *next; //указатель на следующий элемент
	DoubleList *prev; //указатель на предыдущий элемент
};
DoubleList *head; //указатель на первый элемент списка
//**********************ДОБАВЛЕНИЕ ЭЛЕМЕНТА**********************
void AddList(int value, int position)
{
	DoubleList *node = new DoubleList; //создание нового элемента
	node->data = value; //присвоение элементу значения
	if (head == NULL) //если список пуст
	{
		node->next = node; //установка указателя next
		node->prev = node; //установка указателя prev
		head = node; //определяется голова списка
	}
	else
	{
		DoubleList *p = head;
		for (int i = position; i > 1; i--) p = p->next;
		p->prev->next = node;
		node->prev = p->prev;
		node->next = p;
		p->prev = node;
	}
	cout << "\nЭлемент добавлен...\n\n";
}
//**********************УДАЛЕНИЕ ЭЛЕМЕНТА**********************
int DeleteList(int position)
{
	if (head == NULL) { cout << "\nСписок пуст\n\n"; return 0; }
	if (head == head->next)
	{
		delete head;
		head = NULL;
	}
	else
	{
		DoubleList *a = head;
		for (int i = position; i > 1; i--) a = a->next;
		if (a == head) head = a->next;
		a->prev->next = a->next;
		a->next->prev = a->prev;
```

```
			delete a;
		}
		cout << "\nЭлемент удален...\n\n";
}
//************************ВЫВОД СПИСКА************************
void PrintList()
{
		if (head == NULL) cout << "\nСписок пуст\n\n";
		else
		{
			DoubleList *a = head;
			cout << "\nЭлементы списка: ";
			do
			{
				cout << a->data << " ";
				a = a->next;
			} while (a != head); cout << "\n\n";
		}
}
//************************ГЛАВНАЯ ФУНКЦИЯ************************
void main()
{
		setlocale(LC_ALL, "Rus");
		int value, position, x;
		do
		{
			cout << "1. Добавить элемент" << endl;
			cout << "2. Удалить элемент" << endl;
			cout << "3. Вывести список" << endl;
			cout << "0. Выйти" << endl;
			cout << "\nНомер операции > "; cin >> x;
			switch (x)
			{
			case 1:
				cout << "Значение > "; cin >> value;
				cout << "Позиция > "; cin >> position;
				AddList(value, position); break;
			case 2:
				cout << "Позиция > "; cin >> position;
				DeleteList(position); break;
			case 3: PrintList(); break;
			}
		} while (x != 0);
		}
```

Индивидуальное задание

В соответствии с Вашим вариантом напишите и отладьте программу для формирования обработки динамического списка, считая, что длина списка задана.

Номер варианта	Задание
1.	Используя динамическую структуру список, подсчитать количество цифр в заданном наборе символов. Записать их в отдельный динамический массив. Полученный массив вывести на экран.

2.	Найти сумму четных элементов списка, состоящего из не менее чем двух элементов. Записать их в отдельный динамический массив. Полученный массив вывести на экран.
3.	Используя динамическую структуру список, посчитать количество чисел кратных 10 в списке. Записать их в отдельный динамический массив. Полученный массив вывести на экран.
4.	Используя динамическую структуру список, подсчитать количество четных чисел в заданном наборе символов. Записать их в отдельный динамический массив. Полученный массив вывести на экран.
5.	Используя динамическую структуру список, подсчитать произведение отрицательных чисел в списке. Записать их в отдельный динамический массив. Полученный массив вывести на экран.
6.	Сформировать список из элементов целого типа. Четные элементы возвести в квадрат. Записать их в отдельный динамический массив. Полученный массив вывести на экран.
7.	Используя динамическую структуру список, подсчитать количество элементов лежащих в диапазоне от 20 до 50. Записать их в отдельный динамический массив. Полученный массив вывести на экран.
8.	Используя динамическую структуру список, увеличить в 2 раза нечетные числа. Записать их в отдельный динамический массив. Полученный массив вывести на экран.
9.	Используя динамическую структуру список, подсчитать произведение элементов массива с номерами 3, 6, 9, 12 и т.д. Записать их в отдельный динамический массив. Полученный массив вывести на экран.
10.	Используя динамическую структуру список, проверить является ли он упорядоченным по возрастанию набором чисел. Если нет, то сформировать динамический массив и записать в него начало списка, которое является упорядоченной по возрастанию. Например, список: 1, 2, 3, 4, -8, 6, 7, 35, динамический массив: 1, 2, 3, 4.

Пример выполнения задания.

Создать список из 50 элементов и заполнить его случайными символами. Узнать сколько в полученном списке английских строчных букв и записать их в отдельный динамический массив. Вывести полученный массив на экран.

Листинг 1.

```
#include "stdafx.h"
```

```
#include <iostream>
//#include <stdlib.h>  RAND_MAX (32767)
// для изменения диапазона изменения случайных чисел
#include <ctime>
using namespace std;
int main()
{
     struct node
     { unsigned char info;
     struct node *next;
     };
     srand(time(0));// без этого числа будут одинаковые
     //srand - Sets a random starting point.
typedef node *NodePtr; // указатель на тип node - шаблон
NodePtr head = NULL;
NodePtr tek; // указатель на текущий элемент
NodePtr hvost; // указатель на "хвост" списка
int N = 50; // количество элементов в списка
int cnt = 1; // счетчик элементов в списка
if (head == NULL)
{ head = new node;
head->info = char(rand()%255); // случайный символ
head->next = NULL;
hvost = head;
}
for (int i = 2; i<=N; i++)
{ tek = new node;
++cnt;
tek->info =char(rand()%255);
tek->next=NULL;
hvost->next=tek;   // в данном случае - NULL
 hvost = tek;
}
// Вывод списка на экран
cout<<"Spisok simvolov"<<endl;
tek = head;
int *bukva=0;
int *temp=0;
int k=0;
for (int i = 1; i<=N; i++)
{
     cout << tek->info << ' ';
if ((tek->info>=97) && (tek->info<=122))
     {
          k++;
          if (k==1)
          {
          bukva = new int [k];
          bukva[0]=tek->info;
          //создаем дополнительный массив
          temp = new int [k];// на кол-во элементов в массиве с
данными
          temp[0] = bukva[0];
```

```
            }
            else
            {
            delete [] bukva; // удаляем массив
            bukva= new int[k];
// создем новый массив на 1 больше
            for(int i=0;i<k-1;i++)
            bukva[i]=temp[i];
            bukva[k-1] = tek->info;// добавляем элемент в конец
            delete [] temp;
            temp = new int [k];
      //копируем все данные из массива bukva в массив temp
      for(int i=0;i<k;i++)
      temp[i] = bukva[i];
            }
      }
tek = tek->next;
}
cout<<endl<<"Kol-vo bukv="<<k<<endl;
if (k!=0)
      {
      cout<<"Spisok strochnyh angliyskih bukv"<<endl;
      for(int i=0;i<k;i++)
      cout << char(bukva[i])<<'\t';
      cout<<endl;
      }
delete [] bukva;
delete [] temp;
return 0;
}
```

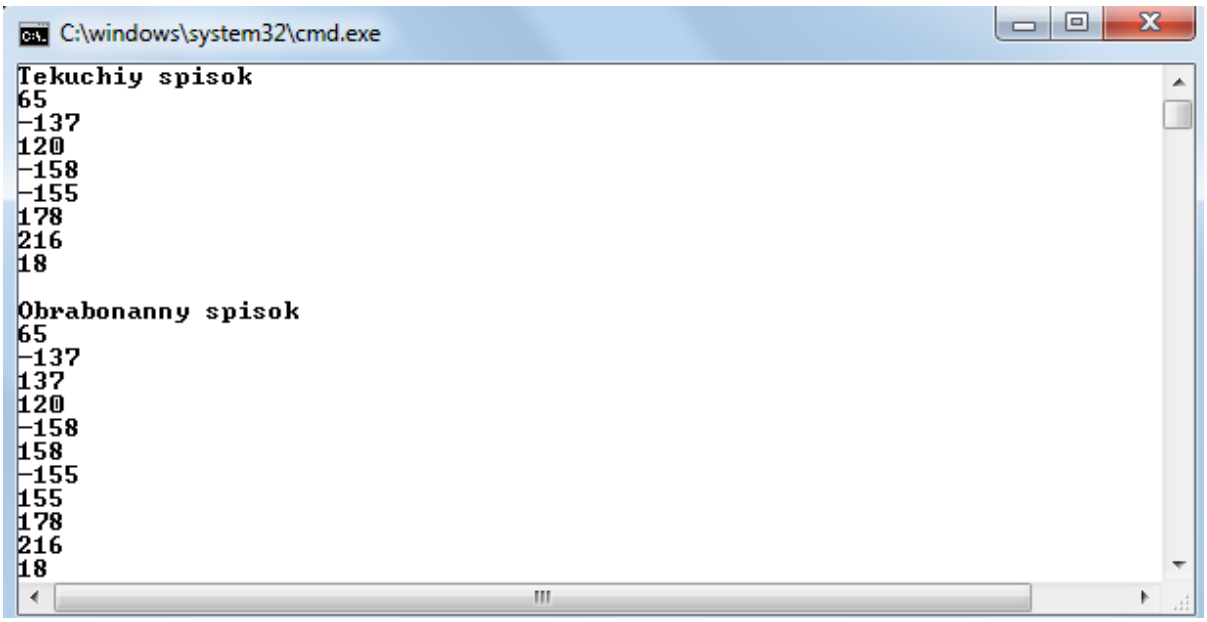

Рис. 9. Результаты работы программы из задания 1