4.10. Динамические структуры данных

К таким структурам относят:
1) линейные списки;
2) стеки;
3) очереди;
4) бинарные деревья.

Они отличаются способом связи отдельных элементов и допустимыми операциями. Динамическая структура может занимать несмежные участки динамической памяти.

Наиболее простой динамической структурой является линейный однонаправленный список, элементами которого служат объекты структурного типа (рис. 25).

Рис. 25. Линейный однонаправленный список

4.10.1. Линейный однонаправленный список

Описание простейшего элемента такого списка выглядит следующим образом:

struct имя_типа
{
информационное поле;
адресное поле;
};

Информационное поле – это поле любого ранее объявленного или стандартного типа. Информационных полей может быть несколько.

Адресное поле – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента списка.

Пример 104

```
1.
struct List
{
int key;          //информационное поле
List *next;       //адресное поле
};

2.
struct point
{
char*name;     //информационное поле
int age;          //информационное поле
point*next;    //адресное поле
};
```

Каждый элемент списка содержит ключ, который идентифицирует этот элемент. Ключ обычно бывает либо целым числом (*пример 104 1.*), либо строкой (*пример 104 2.*).

Над списками можно выполнять следующие *операции:*

1) начальное формирование списка (создание первого элемента);

2) добавление элемента в конец списка;
3) добавление элемента в начало списка;
4) удаление элемента с заданным номером;
5) чтение элемента с заданным ключом;
6) вставка элемента в заданное место списка (до или после элемента с заданным ключом);
7) упорядочивание списка по ключу;
8) и др.

Пример. Функция формирования списка (*пример 104 1.*) из n элементов.

```
List * Init(List *beg)
{
	int n;
	cout << "ВВЕДИТЕ КОЛИЧЕСТВО ЭЛЕМЕНТОВ СПИСКА"<<endl;
	cin>>n;
	cout << "ВВОДИТЕ  " <<n <<"  ЭЛЕМЕНТОВ СПИСКА"<<endl;
	beg = new List;
	cin>>beg->key;
	beg->next = NULL;
	List *p = beg;
	for(int i=1; i<n; i++)
	{
		p->next = new List;
		p=p->next;
		cin>>p->key ;
		p->next=NULL;
	}
return beg;
}

/*печать списка, на который указывает указатель beg*/
void print_list(List*beg)
{
	List*p=beg;		//p присвоить адрес первого элемента списка
	while(p)		//пока нет конца списка
	{
		cout<<p-> key; /*печать элемента, на который указывает элемент p*/
		p=p->next;	//переход к следующему элементу
	}
}
```

Пример 106. Удаление из однонаправленного списка элемента с номером k (рис. 26.).

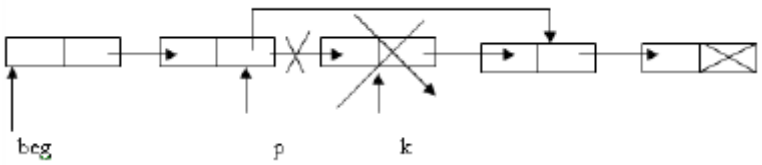

Рис. 26. Удаление элемента с номером k из однонаправленного списка

```
//удаление элемента с номером к

List*del_List(List*beg,int k)
{
	List*p=beg;	/*поставить	вспомогательную переменную на начало списка*/
	if(k==1)
	{	//удалить первый элемент
		beg=beg->next;
		delete[]p->name;	/*удалить	динамическое поле name*/
		delete[]p;		//удалить элемент из списка
	}
	 else
	{
		int i=0;		//счетчик элементов в списке
		List*r;		//вспомогательная переменная для удаления
		while(i<k-2)		//поставить	p на	элемент с номером k-1
		{
			p=p->next;
			i++;
		}
	r=p->next;	//поставить	r	на удаляемый элемент
	if(r)		//если r не последний элемент
	{
		p->next=r->next;	//исключить	r	из списка
		delete[]r->name;	//удалить динамическое поле name
		delete[]r;		//удалить	элемент	из списка
	}
	else
	{
		p->next=0;	//если r -последний элемент, то полю p->next присвоить NULL
		delete[]r;		//удалить	элемент	из списка
	}

}
return beg;		//вернуть адрес первого элемента
}
```

Вызов функции del_List

```
beg=del_List(beg,2);
print_list(beg);
```

Для **добавления элемента в начало списка** необходимо создать новый элемент (t), связать его с первым элементом и установить указатель beg на первый элемент списка.

Пример 8.2. Функция добавления в начало списка.

```
List * Add_begin(List *beg, const int a)	//добавление в начало списка
{
	List *t = new List;		//создание нового элемента списка
	t->a=a;				//занесение значения в новый элемент списка
```

```
        t->next = beg; //связывание нового элемента списка с первым элементом
        beg=t;                          //установка указателя beg на начало списка
        return beg;
}
```

Для **добавления элемента в конец списка** необходимо установить указатель (t) на последний элемент списка, создать новый элемент (p), связать последний элемент списка и новый элемента, занести в адресное поле нового элемента NULL.

Пример 8.3. Функция добавления в конец списка.

```
List * Add_end(List *beg,const int a) //добавление в конец списка
{
        List *t=beg;            //установка указателя t на начало списка

        while(t->next)          //установка указателя t на последний элемент списка
                t = t->next;
        List *p = new List;     //создание нового элемента списка
        p->a=a;                 //занесение значения в новый элемент списка
        t->next=p;      //связывание последнего элемента списка и нового элемента
        p->next=NULL;           //занесение в адресное поле нового элемента NULL
        return beg;
}
```

Для **добавления элемента в k–ю позицию (за исключением первой и последней) списка** необходимо установить указатель (t) на элемент перед вставляемым (позиция k-1), создать новый элемент (p), связать новый элемент и k+1 элемент списка, связать k-1 элемента списка и новый элемент.

Пример 8.4. Функция добавления в k–ю позицию списка (за исключением первой и последней).

```
List * Add_middle(List *beg, const int a)     //добавление в середину списка
{
        List *t=beg;                    //установка указателя t на начало списка
        int k,i=0;
        cout << "ВВЕДИТЕ НОМЕР ПОЗИЦИИ ДЛЯ ВСТАВКИ ЭЛЕМЕНТА
СПИСКА"<<endl;
        cin>>k;
    //установка указателя t на k-1 элемент списка (на  элемент перед вставляемым)
        while(i<k-2)
                {
                        t = t->next;    //проход по списку
                        i++;
                }
        List *p = new List;             //создание нового элемента списка
        p->a=a;                         //занесение значения в новый элемент списка
        p->next=t->next;        // связывание нового элемента и k+1 элемент списка
         t->next=p;             //связывание k-1 элемента списка и нового элемента
         return beg;
}
```

Ниже приведена функция, демонстрирующая удаление элемента с заданным значением.

Пример 8.5. Функция удаления элемента с заданным значением.

```
List * Delete(List *beg, const int a)  //удаление элемента  списка
{
        //если список пуст
        if (beg==NULL)
                {
                        return beg;
                }

//если список не пуст
        List *t = beg;                  //установка указателя t на начало списка
        // удаление первого элемента списка
        if(t->a==a)
                {
                        beg=t->next;   //сдвиг указателя beg на второй элемент списка
                        delete t;               //удаление первого элемента  списка
                        return beg;
                }

        // удаление внутри списка (и последнего элемента списка)
        List *t1 = t->next;      //установка указателя t1 на элемент, следующий за t (на второй)
        while(t1)
                {
                 if(t1->a==a)
                        {
                         t->next=t1->next;     //связывание предыдущего со следующим за удаляемым
                         delete t1;              //удаление  элемента  списка
                         t1=t->next;
                         return beg;
                        }
                        else {
                                t=t1;                   //переход на следующий элемент
                                t1=t1->next;            //переход на следующий элемент
                             }
}
 return beg;
}
```

Задача 8.2. Построить список символов до первой точки. Создать из исходного списка новый, исключив все символы-цифры.

```
#include <iostream>
using namespace std;
struct List
{
        char a;
```

```
        List* next;
};

// построение списка символов до первой точки
List * Init()
{
        List *beg;
        char ch;
        cout << "ВВОДИТЕ  ЭЛЕМЕНТЫ СПИСКА"<<endl;
        cin>> ch;
if (ch!='.')
{
        beg = new List;
        beg->a = ch;
        beg->next = NULL;
        List *p = beg;
        cin>> ch;
        while (ch != '.')
        {
                p->next = new List;
                p=p->next;
                p->a = ch;
                cin>> ch;
                p->next=NULL;
        }
}
return beg;
}

//печать  списка
void Print(List *b)
{
        while(b)
        {
                cout << b->a << " \t ";                 //печать текущего элемента списка
                b = b->next;            //переход на следующий элемент списка
        }
}

//создание из исходного списка нового без символов-цифр
List * new_List(List *b)
{
        List *p1, * f1;
        while (b  && b ->a >='0'&& b ->a <='9')  b = b ->next; //поиск элемента
отличного от цифр
        if (b == NULL) return b;       //новый список создать нельзя;
        else
            {
               p1= new List;              //формирование 1-го элемента нового списка
               p1->a=b->a;
               f1=p1;              // f1 указатель на начало нового списка
               b=b ->next;
```

```
                while (b != NULL)
                    {
                        if  ( b ->a <'0'|| b ->a >'9') // если элемент списка не цифра
                            {
                                p1->next = new List;       //формирование остальных
элементов нового списка
                                p1=p1->next;
                                p1->a=b->a;
                                b = b->next;
                             }
                        else b = b->next;
                    }
                p1->next= NULL;
            }
            return f1;
        }
        int main ()
        {
            setlocale(LC_ALL, "russian");
            List *beg=NULL,*beg_n;
            beg=Init();
            cout << "\nИСХОДНЫЙ СПИСОК"<<endl;
            Print(beg);
            beg_n=new_List(beg);
            if (beg_n == NULL) cout << "\nНовый список создать нельзя";
                    else
                                {
                                    cout << "\nНОВЫЙ СПИСОК"<<endl;
                                    Print(beg_n);
                                }
            return 0;
        }
```

Понятие о стеке и очереди. ***Стек*** – это структура данных, в которой элемент, записанный последним, считывают (он является доступным к обработке) первым. Принцип «последний пришел – первый ушел» (LIFO, last in-first out) используется во многих технических устройствах и в быту: рожок от автомата, посадка пассажиров в вагон с одной дверью и т.д. Стек используют в программировании для реализации рекурсии. Рекурсия выполняется так: сначала все вызовы накапливаются (аналогия такая: пружина сжимается), а потом выполняются вложенные процедуры или функции (пружина распрямляется).

Очередь – это структура данных, в которой элемент, записанный первым, считывают первым. Здесь действует принцип «первый пришёл – первый ушел» (FIFO, first in-first out), хорошо известный в быту: очередь за билетами, очередь на обслуживание и т.п.

Максимально допустимые размеры стека и очереди – важные характеристики реализации языка программирования, определяющие круг задач, которые можно решить. Стеки и очереди описывают и создают в памяти с помощью типа данных – указателя. Соответствующие описания и примеры процедур их обработки можно отыскать в справочниках.