Задание 8: Линейные списки

Цель работы

Составить и отладить программу на языке Си для обработки линейного списка заданной организации с отображением списка на динамические структуры или на массив. Навигацию по списку следует реализовать с применением итераторов. Предусмотреть выполнение одного нестандартного и четырёх стандартных действий:

1. Печать списка.
2. Вставка нового элемента в список.
3. Удаление элемента изсписка.
4. Подсчёт длины списка.

Задание

Тип элемента списка: целый

Вид списка: линейный однонаправленный с барьерным элементом

Нестандартное действие: отчистить список, если в нём есть элемент, равный заданному значению

Алгоритм работы программы

Навигация по списку и основные действия:

Для навигации пользователю доступны функции перемещения к следующему элементу списка и возвращение к первому элементу. Перемещение к следующему элементу возможно, если пользователь не находится в конце списка.

Пользователю доступны стандартные операции над списком:

1. Печать списка: перемещаемся в начало списка, идем до конца списка, распечатывая все элементы.
2. Вставка нового элемента перед текущим: создаем новый элемент, полю-указателю нового элемента присваиваем адрес следующего за текущим, полю-указателю текущего элемента присваиваем адрес нового элемента.
3. Вставка на первое место: создаем новый элемент, полю-указателю нового элемента присваиваем адрес нынешнего первого элемента, полю-указателю барьерного элемента присваиваем адрес нового элемента
4. Удаление текущего элемента: запоминаем указатель на текущий элемент, находим предыдущий элемент, полю-указателю предыдущего элемента присваиваем адрес следующего элемента, очищаем память, выделенную под текущий элемент, по сохраненному указателю.
5. Подсчёт длины списка: перемещаемся в начало списка, идем до конца списка, инкрементируя счётчик при каждом перемещении, когда перемещение больше

не возможно печатаем значение счётчика.

Очищение списка, если в нём есть элемент, равный заданному:

Обходим список, если при этом встречается элемент, равный заданному, запускаем вспомогательную функцию, рекурсивно очищающую список. Вспомогательная функция рекурсивно доходит до конца списка, после чего очищает память, выделенную под элемент, для которого в последний раз была вызвана эта функция.

Функции и структуры

Для хранения элементов списка используется структура Item, содержащая целую переменную value и ссылку на структуру Item – next. Список представлен в виде указателя на барьерный элемент типа Item, ссылающегося на первый элемент.

List Next(List it)	Переход к следующему элементу списка
List first(List m)	Переход к первому элементу списка
void Print(List m)	Печать списка
void Add(List it)	Добавление элемента в список после текущего
void Del(List it,List m)	Удаление текущего элемента
void Ins(List m)	Вставка элемента на первое место
void clear(List *m)	Очищение списка
void Clear(List m, int x)	Очищение списка, если в нём есть элемент, равный заданному
void Length(List m)	Вычисление длины списка

Код программы

```
#include<stdio.h>
#include<stdlib.h>

struct Item{
	int value;
	struct Item* next;
};


typedef struct Item* List;

//NEXT
```

```
List Next(List it){
	if(it->next!=NULL) return it->next; else return it;
}
//FIRST
List first(List m){
	if(m->next!=NULL)return m->next; else return m;
}
//PRINT
void Print(List m){
	List n=m->next;
	while(n!=NULL){
	   printf("%d ",n->value);
	   n=n->next;
	}
}
//ADD
void Add(List *it, int b, List m){
	List n=(List)malloc(sizeof(struct Item));
	n->value=b;
	n->next=(*it)->next;
	(*it)->next=n;
	if(*it==m) *it=m->next;
}
//DELETE
void Del(List it, List m){
	if(m->next!=NULL){
			List n=m;
			while(n->next!=it){
			   n=n->next;
			}
			n->next=it->next;
			free(it);
			it=n;
	}
}
//INSERT
void Ins(List m, int x){
	List n=m;
	Add(&n, x, m);
}
//CLEAR
void clear(List m){
	if(m->next!= NULL) clear(m->next);
	free(m);
}
//OPTION
void Clear(List *it, List m, int x){
   int i=0;
   if(m->next!=NULL){
	List n=m->next;
			while(n!=NULL){
			   if(n->value==x) {i++; break;}
```

```
                        n=n->next;}
                n=m;
                if(i!=0){
                        *it=m;
                        clear(m->next);
                        m->next=NULL;
                }
        }
        else printf("List is empty\n");
}
//LENGTH
void Length(List m){
        List n=first(m)->next;
        int i=0;
        while(n!=NULL){
           i++;
           n=n->next;
        }
        printf("%d\n",i);
}

int main(){
        printf("q - Quit\np - Print\nf - First\n\
a - Add\nd - Delete\nl - Length\nc - Clear\n\
i - Insert to first\nn - Next\n\n");
        int x;
        List m = NULL;
        List it = NULL;
        struct Item bar;
        bar.value=0;
        m=&bar;
        it=m;
        bar.next=NULL;
        char a=' ';
        while(1 > 0){
        if(a!='\n') printf("Enter command:\n");
        scanf("%c",&a);
        if(a!='\n') printf("\n");

        switch(a){
                case 'q':
                        return 0;
                        break;
                case 'p':
                        if(m->next==NULL) printf("List is empty\n");
                        else Print(m);
                        printf("\n");
                        break;
                case 'f':
                        it=first(m);
                        if(it!=m)printf("%d",it->value);
                        else printf("List is empty\n");
```

```
                        printf("\n");
                        break;
                case 'a':
                        scanf("%d",&x);
                        Add(&it, x, m);
                        if(it->next==NULL) /
                        printf("%d was added at first place\n", it->value);
                        else
                        printf("Added element: %d\n\n",it->next->value);
                        break;
                case 'd':
                        if(m->next==NULL) printf("List is empty\n");
                        else Del(it, m);
                        printf("\n");
                        break;
                case 'l':
                        Length(m);
                        printf("\n");
                        break;
                case 'c':
                        if(m->next==NULL) printf("List is empty\n");
                        else{
                        printf("Enter value: ");
                        scanf("%d",&x);
                        Clear(&it, m, x);}
                        printf("\n");
                        break;
                case 'i':
                        printf("Enter value: ");
                        scanf("%d",&x);
                        Ins(m, x);
                        break;
                case 'n':
                   it=Next(it);
                   if(it!=m)printf("%d\n",it->value);
                   else printf("List is empty\n");
                   printf("\n");
        }
}
}
```

Вывод программы

```
q - Quit
p - Print
f - First
a - Add
d - Delete
l - Length
c - Clear
i - Insert to first
n - Next
```

Enter command:
p

List is empty

Enter command:
f

List is empty

Enter command:
d

List is empty

Enter command:
l

0

Enter command:
c

List is empty

Enter command:
a

5
5 was added at first place
Enter command:
p

5
Enter command:
a

4
Added element: 4

Enter command:
p

5 4
Enter command:
a

3
Added element: 3

Enter command:
p

5 3 4
Enter command:
i

Enter value: 4
Enter command:
p

4 5 3 4
Enter command:
a

6
Added element: 6

Enter command:
p

4 5 6 3 4
Enter command:
n

6

Enter command:
d

Enter command:
c

Enter value: 6

Enter command:
p

4 5 3 4
Enter command:
c

Enter value: 3

Enter command:
p

List is empty

Enter command:
a

4

4 was added at first place
Enter command:
p

4
Enter command:
d

Enter command:
p

List is empty

Enter command:
q

Заключение

В ходе работы мной был реализован линейный однонаправленный список с барьерным элементом. Преимущества барьерного элемента заключаются в том, что список может быть не пустым до начала работы с ним (при создании списка итератор может указывать на барьерный элемент), но тогда при удалении необходимо проверять, не указывает ли итератор на барьерный элемент; если элементы списка целочисленные, то можно хранить длину списка без использования дополнительной переменной в поле значения барьерного элемента, чтобы не проходить список каждый раз при запросе длины; список может быть объявлен как барьерный элемент, что позволяет не хранить лишнюю ссылку.