Отчёт по практикуму по циклу дисциплин «Информатика».
Задание №8.
Выполнил студент группы М8О-111Б-23: Тимофеева Ирина № по списку 21

Работа выполнена: «24» мая 2024 г.
Преподаватель: каф. 806 Никулин Сергей Петрович
Входной контроль знаний с оценкой: _____
Отчет сдан «24» мая 2024 г.
Итоговая оценка: _____
Подпись преподавателя: _____

1. Тема: реализация линейных списков на си.

2. Цель работы: составить и отладить программу на языке Си для обработки линейного списка заданной
организации с отображением списка на динамические структуры.

3. Задание: реализация линейного двунаправленного списка с элементами булонского типа с
нестандартным действием: удалить элементы списка со значениями, находящимися в заданном диапазоне.

4. Оборудование: Оборудование ПЭВМ студента, если использовалось:
Процессор AMD Ryzen 5 5600H, ОП 16 ГБ, SSD 250 ГБ, мониторы 15" Full Hd Display и 27" BenQ
BL2780T. Другие устройства: принтер Canon MG4520S, мышь Logitech g403, наушники HyperX Cloud
Alpha.

5. Программное обеспечение: Программное обеспечение ПЭВМ студента, если использовалось:
Операционная система семейства Ubuntu, наименование версия VirtualBox Ubuntu 20.04.3
LTS,
интерпретатор команд bash версия 5.0.17. Система программирования C. Редактор текстов
VI версия 8.1

6. Идея, метод, алгоритм решения задачи [в формах: словесной, псевдокода, графической (блок-схема,
диаграмма, рисунок, таблица) или формальные спецификации с пред- и постусловиями]:

Задача состоит в том, чтобы вводить элементы в линейный список и выполнять над ним заданные действия.

7. Сценарий выполнения работы (план работы, первоначальный текст программы в черновике (можно
на отдельном листе) и тесты либо соображения по тестированию).
Сценарий выполнения работы:
Подготовить набор тестов.
Придумать алгоритм решения поставленной задачи.
Используя функции заданной системы программирования, реализовать придуманный алгоритм.
Провести тестирование программы, чтобы убедиться в корректности её работы.
Сформировать протокол.
Тексты программ:

list.c:

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

// инициализация структуры двунаправленного списка
List* CreateList()
{
	List* new_list = (List*)malloc(sizeof(List));
	new_list->head = NULL;
	new_list->current = NULL;
	new_list->size = 0;
	return new_list;
}

// итератор текущего элемента списка
Node* GetNode(List* list)
{
	Node* p = list->current;
	if (p) list->current = list->current->next;
	return p;
}

// итератор первого элемента
Node* GetFirstNode(List* list)
{
	list->current = list->head;
	return list->head;
```

```
}

// итератор последнего элемента
Node* GetLastNode(List* list)
{
	Node* p = GetFirstNode(list);
	Node* tail;
	while(p != NULL)
	{
		tail = p;
		p = GetNode(list);
	}
	return tail;
}

// размер списка
int ListSize(List* list)
{
	return list->size;
}

// добавить элемент в конец списка
void AddListNode(List* list, ValueType val)
{
	Node *temp = (Node*)malloc(sizeof(Node));
	Node *p;
	temp->value = val;
	temp->next = temp->prev = NULL;
	if (ListSize(list) == 0)
	{
		list->head = list->current = temp;
	}
	else
	{
		p = GetLastNode(list);
		p->next = temp;
		temp->prev = p;
		//list->tail = temp;
	}
	list->size++;
}

// вставить элемент в позицию indx
Node* InsertNode(Node* root, int indx, ValueType val)
{
```

```
	int i;
	Node* p = root;
	for(i = 0; i < indx; i++)
	{
		root = root->next;
		if(root==NULL) return p;
	}
	Node* temp = (Node*)malloc(sizeof(Node));
	temp->value = val;
	if (root->prev==NULL)
	{
		temp->next = root;
		temp->prev = NULL;
		root->prev = temp;
		root = temp;
		p = root;
	}
	else
	{
		temp->next = root;
		temp->prev = root->prev;
		root->prev->next = temp;
		root->prev = temp;
	}
	return p;
}

// вставить элемент в список
void InsertListNode(List* list, int indx, ValueType val)
{
	list->head = InsertNode(GetFirstNode(list), indx, val);
	list->size++;
}

// удалить элемент с индексом indx
void DeleteListNode(List* list, int indx)
{
	Node* p = GetFirstNode(list);
	int i = 0;
	while ((p = GetNode(list)) && i < indx)
	{
		i++;
	}
	if (p==NULL) return;
	if (p->prev)
```

```
            p->prev->next = p->next;
        else
        {
            list->head = p->next;
            //if (p->next == NULL) list->tail = p->next;
        }
        if (p->next) p->next->prev = p->prev;
        free(p);
        list->size--;
}

// удалить элементы списка в диапазоне значений L..R
void DeleteListRange(List* list, int L, int R)
{
        Node* p = GetFirstNode(list);
        int i = 0;
        while (p = GetNode(list))
        {
            if (p->value >= L && p->value<=R)
            {
                DeleteListNode(list, i--);
            }
            i++;
        }
}

// печать списка
void PrintListNodes(List* list)
{
        Node* p = GetFirstNode(list);
        while (p = GetNode(list))
        {
            printf("[%4d]", p->value);
        }
        printf("\n");
}

// удалить список
void DestroyList(List* list)
{
        Node* p;
        while (p = GetNode(list))
        {
            free(p);
        }
```

```
	free(list);
}
```

list.h:

```
#ifndef _DECK_H_
#define _DECK_H_

#define N 12

typedef int ValueType;

typedef struct Node{
	ValueType value;
	struct Node *prev;
	struct Node *next;
} Node;

// итератор для навигации
typedef struct {
	Node* head;
	Node* current;
	unsigned int size;
} List;

// инициализация структуры двунаправленного списка
List* CreateList();
// итератор текущего элемента списка
Node* GetNode(List* list);
// итератор первого элемента
Node* GetFirstNode(List* list);
// итератор последнего элемента
Node* GetLastNode(List* list);
// добавить элемент в конец списка
void AddListNode(List* list, ValueType val);
// вставить элемент в позицию indx
void InsertListNode(List* list, int indx, ValueType val);
// удалить элемент с индексом indx
void DeleteListNode(List* list, int indx);
// удалить элементы списка в диапазоне значений L..R
void DeleteListRange(List* list, int L, int R);
// печать списка
void PrintListNodes(List* list);
// удалить список
void DestroyList(List* list);
```

```
#endif
```

linelist.c:

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

int main()
{
  // инициализация генератора случайных чисел
  srand(time(NULL));

  int i, v;
  char c;
  int done = 0;
  List* NodesList = CreateList();
  while (!done)
  {
     printf("Main menu:\n");
     printf("1 - make list\n");
     printf("2 - print list\n");
     printf("3 - add list item\n");
     printf("4 - insert list item\n");
     printf("5 - delete list item\n");
     printf("6 - delete range items\n");
     printf("0 - exit\n");
     printf("Enter command: ");
     c = getchar();
     getchar();
     printf("\n-----------------------------------------------\n");
     switch (c)
     {
     case '0':
         done = 1;
         break;
     case '1':
                        // заполнение списка случайными числами
                for (i = 0; i < N; i++)
                AddListNode(NodesList, rand() % 100);
                        if (ListSize(NodesList) > 0)
                        printf("List created!\n");
                        else
                printf("Error of list creating!\n");
         break;
     case '2':
```

```
            // печать списка
    printf("List is:\n");
                PrintListNodes(NodesList);
    printf("\n");
    break;
case '3':
            // добавление элемента в конец списка
            printf("Enter new item value:");
            scanf("%d", &v);
            getchar();
            AddListNode(NodesList, v);
            printf("New item is added!\n");
    break;
case '4':
            // вставка нового элемента
            printf("Enter new item value:");
            scanf("%d", &v);
            printf("Enter position:");
            scanf("%d", &i);
            getchar();
            InsertListNode(NodesList, i, v);
            printf("New item inserted!\n");
    break;
case '5':
            // удаление элемента
            printf("Enter index for delete:");
            scanf("%d", &i);
            getchar();
            DeleteListNode(NodesList, i);
            printf("Item deleted!\n");
    break;
case '6':
            // удаление элементов списка со значениями в заданном диапазоне
            printf("Enter low range value:");
            scanf("%d", &v);
            printf("Enter high range value:");
            scanf("%d", &i);
            getchar();
            DeleteListRange(NodesList, v, i);
            printf("Delete range completed!\n");
    break;
default:
    printf("incorrect input!\n");
    printf("\n");
}
```

```
    }

  DestroyList(NodesList);

  return 0;
}
```

8. Распечатка протокола (подклеить листинг окончательного варианта программы с тестовыми
примерами, подписанный преподавателем).

```
#листинг программы работы с двунаправленным списком
#заголовочный файл
irina@Irina-VivoBook:~/Prog/Prog_C/Lab8$ cat list.h
#ifndef _DECK_H_
#define _DECK_H_

#define N 12

typedef int ValueType;

typedef struct Node{
        ValueType value;
        struct Node *prev;
        struct Node *next;
} Node;

// итератор для навигации
typedef struct {
        Node* head;
        Node* current;
        unsigned int size;
} List;

// инициализация структуры двунаправленного списка
List* CreateList();
// итератор текущего элемента списка
Node* GetNode(List* list);
// итератор первого элемента
Node* GetFirstNode(List* list);
// итератор последнего элемента
Node* GetLastNode(List* list);
// добавить элемент в конец списка
void AddListNode(List* list, ValueType val);
// вставить элемент в позицию indx
```

```
void InsertListNode(List* list, int indx, ValueType val);
// удалить элемент с индексом indx
void DeleteListNode(List* list, int indx);
// удалить элементы списка в диапазоне значений L..R
void DeleteListRange(List* list, int L, int R);
// печать списка
void PrintListNodes(List* list);
// удалить список
void DestroyList(List* list);

#endif

#библиотека функций
irina@Irina-VivoBook:~/Prog/Prog_C/Lab8$ cat list.c
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

// инициализация структуры двунаправленного списка
List* CreateList()
{
	List* new_list = (List*)malloc(sizeof(List));
	new_list->head = NULL;
	new_list->current = NULL;
	new_list->size = 0;
	return new_list;
}

// итератор текущего элемента списка
Node* GetNode(List* list)
{
	Node* p = list->current;
	if (p) list->current = list->current->next;
	return p;
}

// итератор первого элемента
Node* GetFirstNode(List* list)
{
	list->current = list->head;
	return list->head;
}

// итератор последнего элемента
Node* GetLastNode(List* list)
```

```
{
	Node* p = GetFirstNode(list);
	Node* tail;
	while(p != NULL)
	{
		tail = p;
		p = GetNode(list);
	}
	return tail;
}

// размер списка
int ListSize(List* list)
{
	return list->size;
}

// добавить элемент в конец списка
void AddListNode(List* list, ValueType val)
{
	Node *temp = (Node*)malloc(sizeof(Node));
	Node *p;
	temp->value = val;
	temp->next = temp->prev = NULL;
	if (ListSize(list) == 0)
	{
		list->head = list->current = temp;
	}
	else
	{
		p = GetLastNode(list);
		p->next = temp;
		temp->prev = p;
		//list->tail = temp;
	}
	list->size++;
}

// вставить элемент в позицию indx
Node* InsertNode(Node* root, int indx, ValueType val)
{
	int i;
	Node* p = root;
	for(i = 0; i < indx; i++)
	{
```

```
            root = root->next;
            if(root==NULL) return p;
        }
        Node* temp = (Node*)malloc(sizeof(Node));
        temp->value = val;
        if (root->prev==NULL)
        {
            temp->next = root;
            temp->prev = NULL;
            root->prev = temp;
            root = temp;
            p = root;
        }
        else
        {
            temp->next = root;
            temp->prev = root->prev;
            root->prev->next = temp;
            root->prev = temp;
        }
        return p;
}

// вставить элемент в список
void InsertListNode(List* list, int indx, ValueType val)
{
        list->head = InsertNode(GetFirstNode(list), indx, val);
        list->size++;
}

// удалить элемент с индексом indx
void DeleteListNode(List* list, int indx)
{
        Node* p = GetFirstNode(list);
        int i = 0;
        while ((p = GetNode(list)) && i < indx)
        {
            i++;
        }
        if (p==NULL) return;
        if (p->prev)
            p->prev->next = p->next;
        else
        {
            list->head = p->next;
```

```
            //if (p->next == NULL) list->tail = p->next;
        }
        if (p->next) p->next->prev = p->prev;
        free(p);
        list->size--;
}

// удалить элементы списка в диапазоне значений L..R
void DeleteListRange(List* list, int L, int R)
{
        Node* p = GetFirstNode(list);
        int i = 0;
        while (p = GetNode(list))
        {
            if (p->value >= L && p->value<=R)
            {
                DeleteListNode(list, i--);
            }
            i++;
        }
}

// печать списка
void PrintListNodes(List* list)
{
        Node* p = GetFirstNode(list);
        while (p = GetNode(list))
        {
            printf("[%4d]", p->value);
        }
        printf("\n");
}

// удалить список
void DestroyList(List* list)
{
        Node* p;
        while (p = GetNode(list))
        {
            free(p);
        }
        free(list);
}

# основная программа
```

```
irina@Irina-VivoBook:~/Prog/Prog_C/Lab8$ cat linelist.c
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

int main()
{
  // инициализация генератора случайных чисел
  srand(time(NULL));

  int i, v;
  char c;
  int done = 0;
  List* NodesList = CreateList();
  while (!done)
  {
     printf("Main menu:\n");
     printf("1 - make list\n");
     printf("2 - print list\n");
     printf("3 - add list item\n");
     printf("4 - insert list item\n");
     printf("5 - delete list item\n");
     printf("6 - delete range items\n");
     printf("0 - exit\n");
     printf("Enter command: ");
     c = getchar();
     getchar();
     printf("\n-----------------------------------------------\n");
     switch (c)
     {
     case '0':
         done = 1;
         break;
     case '1':
                         // заполнение списка случайными числами
                 for (i = 0; i < N; i++)
                 AddListNode(NodesList, rand() % 100);
                         if (ListSize(NodesList) > 0)
                         printf("List created!\n");
                         else
                 printf("Error of list creating!\n");
         break;
     case '2':
                 // печать списка
         printf("List is:\n");
```

```
                    PrintListNodes(NodesList);
        printf("\n");
        break;
    case '3':
                // добавление элемента в конец списка
                printf("Enter new item value:");
                scanf("%d", &v);
                getchar();
                AddListNode(NodesList, v);
                printf("New item is added!\n");
        break;
    case '4':
                // вставка нового элемента
                printf("Enter new item value:");
                scanf("%d", &v);
                printf("Enter position:");
                scanf("%d", &i);
                getchar();
                InsertListNode(NodesList, i, v);
                printf("New item inserted!\n");
        break;
    case '5':
                // удаление элемента
                printf("Enter index for delete:");
                scanf("%d", &i);
                getchar();
                DeleteListNode(NodesList, i);
                printf("Item deleted!\n");
        break;
    case '6':
                // удаление элементов списка со значениями в заданном диапазоне
                printf("Enter low range value:");
                scanf("%d", &v);
                printf("Enter high range value:");
                scanf("%d", &i);
                getchar();
                DeleteListRange(NodesList, v, i);
                printf("Delete range completed!\n");
        break;
    default:
        printf("incorrect input!\n");
        printf("\n");
    }
}
```

```
  DestroyList(NodesList);

  return 0;
}
```

```
#компиляция программы
irina@Irina-VivoBook:~/Prog/Prog_C/Lab8$ gcc *.c -o list
linelist.c: In function ‘main’:
linelist.c:8:9: warning: implicit declaration of function ‘time’ [-Wimplicit-function-
declaration]
    8 |   srand(time(NULL));
      |         ^~~~
```

```
#запуск программы
irina@Irina-VivoBook:~/Prog/Prog_C/Lab8$ ./list
Main menu:
1 - make list
2 - print list
3 - add list item
4 - insert list item
5 - delete list item
6 - delete range items
0 - exit
Enter command: 1

---------------------------------------------
List created!
Main menu:
1 - make list
2 - print list
3 - add list item
4 - insert list item
5 - delete list item
6 - delete range items
0 - exit
Enter command: 2

---------------------------------------------
List is:
[  67][  18][  29][  86][   6][  73][  71][  69][   6][  98][  58][  44]

Main menu:
1 - make list
```

```
2 - print list
3 - add list item
4 - insert list item
5 - delete list item
6 - delete range items
0 - exit
Enter command: 3

---------------------------------------------
Enter new item value:555
New item is added!
Main menu:
1 - make list
2 - print list
3 - add list item
4 - insert list item
5 - delete list item
6 - delete range items
0 - exit
Enter command: 2

---------------------------------------------
List is:
[  67][  18][  29][  86][   6][  73][  71][  69][   6][  98][  58][  44][ 555]

Main menu:
1 - make list
2 - print list
3 - add list item
4 - insert list item
5 - delete list item
6 - delete range items
0 - exit
Enter command: 4

---------------------------------------------
Enter new item value:17
Enter position:3
New item inserted!
Main menu:
1 - make list
2 - print list
3 - add list item
4 - insert list item
5 - delete list item
```

6 - delete range items
0 - exit
Enter command: 2

List is:
[67][18][29][17][86][6][73][71][69][6][98][58][44][555]

Main menu:
1 - make list
2 - print list
3 - add list item
4 - insert list item
5 - delete list item
6 - delete range items
0 - exit
Enter command: 5

Enter index for delete:7
Item deleted!
Main menu:
1 - make list
2 - print list
3 - add list item
4 - insert list item
5 - delete list item
6 - delete range items
0 - exit
Enter command: 2

List is:
[67][18][29][17][86][6][73][69][6][98][58][44][555]

Main menu:
1 - make list
2 - print list
3 - add list item
4 - insert list item
5 - delete list item
6 - delete range items
0 - exit
Enter command: 6

```
-------------------------------------------
Enter low range value:1
Enter high range value:39
Delete range completed!
Main menu:
1 - make list
2 - print list
3 - add list item
4 - insert list item
5 - delete list item
6 - delete range items
0 - exit
Enter command: 2

-------------------------------------------
List is:
[  67][  86][  73][  69][  98][  58][  44][ 555]

Main menu:
1 - make list
2 - print list
3 - add list item
4 - insert list item
5 - delete list item
6 - delete range items
0 - exit
Enter command: 0
```

9. Дневник отладки должен содержать дату и время сеансов отладки и основные события (ошибки в
сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки
приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в
написании и отладке программы.

10. Замечания автора по существу работы: замечания отсутствуют.

11. Вывод: в ходе выполнения данного задания практикума я научилась работать с линейными списками в
СП Си.

Подпись студента _____