Отчет по курсовому проекту N 8 по курсу

"Фундаментальная информатика"

Студент группы: М80-107Б-21, Павлов Иван Дмитриевич
Контакты: pavlov.id.2003@gmail.com
Работа выполнена: 30.04.2022
Преподаватель: Найденов Иван Евгеньевич

1 Тема

Линейные списки

2 Цель работы

Составить и отладить программу на языке Си для обработки линейного списка заданной организации с отображением на динамические структуры

3 Задание

Вид списка: линейный двунаправленный с барьерным элементом. Нестандартное действие: поменять местами второй и предпоследний элементы списка.

4 Оборудование

Процессор: AMD Ryzen 5 4600H with Radeon Graphics
ОП: 7851 Мб
НМД: 256 Гб
Монитор: 1920x1080

5 Программное обеспечение

Операционная система семейства: linux (ubuntu), версия 20.04.3 LTS
Интерпретатор команд: bash, версия 5.0.17(1)-release.
Система программирования: gcc*, версия 17
Редактор текстов: emacs, версия 25.2.2
Утилиты операционной системы: subl, make
Прикладные системы и программы: sublime text, bash
Местонахождение и имена файлов программ и данных на домашнем компьютере: /home/ggame/newlabs/

6 Идея, метод, алгоритм решения задачи

Заведем структуры:

- *struct Item* - Структура элемента списка. Содержит указатели на предыдущий и следующий элемент, а также значение текущего элемента;
- *Iterator* - Структура итератора. Содержит указатель на элемент списка;
- *List* - Структура списка. Содержит указатель на барьерный элемент (терминатор), а также размер списка.

Опишем следующие функции:

- *int Equal* - Функция сравнения двух итераторов. Итераторы равны тогда и только тогда, когда указывают на один и тот же элемент;
- *Iterator *Next* - Функция перехода к следующему элементу. Возвращает итератор на следующий элемент;

- *Iterator *Prev* - Функция перехода к предыдущему элементу. Возвращает итератор на предыдущий элемент;
- *float Fetch* - Функция возвращает значение элемента, на который указывает итератор;
- *void Store* - Функция присваивает значение t элементу, на который указывает итератор;
- *void Create* - Функция создания списка. Выделяет память для барьерного элемента, указателю на следующий элемент присваивает значение барьерного элемента. Присваивает размеру списка значение 0;
- *Iterator First* - Функция возвращает итератор на первый элемент списка;
- *Iterator Last* - Функция возвращает указатель на последний элемент списка. В данном случае терминатор;
- *int Empty* - Функция проверки на пустоту списка;
- *Iterator search_prev* - Функция поиска предыдущего элемента. С помощью итераторов пробегает по списку и выполняет поиск элемента, удовлетворяющего условию: Значение следующего элемента равно заданному;
- *Iterator Insert* - Функция вставки элемента в список. Возвращает итератор на только что вставленный элемент;
- *Iterator Delete* - Функция удаления. Возвращает итератор на следующий элемент после удаляемого;
- *int Size* - Функция возвращает размер списка;
- *void print_list* - Функция печати списка. С помощью итераторов пробегает по всем элементам списка и выводит их значения;
- *void Task* - Функция меняет местами второй и предпоследний элемент. Ищет их с помощью итераторов;
- *void Destroy* - Функция уничтожения списка.

7 Сценарий выполнения работы

Тест программы:

Листинг 1: tests

```
ggame@ggame:~/newlabs/kp8/r$ ./main
1. Print list  2. Insert in list  3. Delete from list  4. Size  5. Task  other. Exit
1
NULL
1. Print list  2. Insert in list  3. Delete from list  4. Size  5. Task  other. Exit
2
Write value
1
1. Print list  2. Insert in list  3. Delete from list  4. Size  5. Task  other. Exit
2
Write value
2
Write the number where you want insert element
1
1. Print list  2. Insert in list  3. Delete from list  4. Size  5. Task  other. Exit
1
2.0000 -> 1.0000 -> NULL
1. Print list  2. Insert in list  3. Delete from list  4. Size  5. Task  other. Exit
5
```

```
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
1
1.0000 -> 2.0000 -> NULL
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
3
Write number of deleted element
1
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
2
Write value
4
Write the number where you want insert element
1
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
2
Write value
5
Write the number where you want insert element
1
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
1
5.0000 -> 4.0000 -> 2.0000 -> NULL
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
5
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
1
5.0000 -> 4.0000 -> 2.0000 -> NULL
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
2
Write value
7
Write the number where you want insert element
1
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
5
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
1
7.0000 -> 4.0000 -> 5.0000 -> 2.0000 -> NULL
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
5
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
1
7.0000 -> 5.0000 -> 4.0000 -> 2.0000 -> NULL
1. Print list  2. Insert in list   3. Delete from list   4. Size   5. Task   other. Exit
0
```

8 Распечатка протокола

Листинг 2: list.h

```
#ifndef LIST_H
```

```
2       #define LIST_H
3
4       struct Item
5       {
6        struct Item *prev;
7        struct Item *next;
8        float data;
9       };
10
11      typedef struct
12      {
13       struct Item *node;
14      } Iterator;
15
16      typedef struct
17      {
18       struct Item *head;
19       int size;
20      } List;
21
22      int Equal(Iterator *lhs, Iterator *rhs);
23
24      Iterator *Next(Iterator *i);
25
26      Iterator *Prev(Iterator *i);
27
28      float Fetch(const Iterator *i);
29
30      void Store(const Iterator *i, const float t);
31
32      void Create(List *l);
33
34      Iterator First(const List *l);
35
36      Iterator Last(const List *l);
37
38      int Empty(const List *l);
39
40      Iterator search_prev(const List *l, const int n);
41
42      Iterator Insert(List *l, Iterator *i, const float t);
43
44      Iterator Delete(List *l, Iterator *i);
45
46      int Size(const List *l);
47
48      void print_list(const List *l);
49
50      void Task(List *l);
51
52      void Destroy(List *l);
53
54      #endif
```

Листинг 3: list.c

```
1       #include <stdio.h>
2       #include <stdlib.h>
3       #include "list.h"
4
5       int Equal(Iterator *lhs, Iterator *rhs)
6       {
7        return lhs->node == rhs->node;
8       }
9
10      Iterator *Next(Iterator *i)
```

```
11      {
12       i->node = i->node->next;
13       return i;
14      }
15
16      Iterator *Prev(Iterator *i)
17      {
18       i->node = i->node->prev;
19       return i;
20      }
21
22      float Fetch(const Iterator *i)
23      {
24       return i->node->data;
25      }
26
27      void Store(const Iterator *i, const float t)
28      {
29       i->node->data = t;
30      }
31
32      void Create(List *l)
33      {
34       l->head = malloc(sizeof(struct Item));
35       l->head->next = l->head->prev = l->head;
36       l->size = 0;
37      }
38
39      Iterator First(const List *l)
40      {
41       Iterator res = { l->head->next };
42       return res;
43      }
44
45      Iterator Last(const List *l)
46      {
47       Iterator res = { l->head };
48       return res;
49      }
50
51      int Empty(const List *l)
52      {
53       Iterator fst = First(l);
54       Iterator lst = Last(l);
55       return Equal(&fst, &lst);
56      }
57
58      Iterator search_prev(const List *l, const int n)
59      {
60       Iterator res = Last(l);
61       for (int i = 0; i <= n - 1; i++) {
62         Next(&res);
63       }
64       return res;
65      }
66
67      Iterator Insert(List *l, Iterator *i, const float t)
68      {
69       Iterator res = { malloc(sizeof(struct Item)) };
70       if(!res.node)
71       return Last(l);
72       res.node->data = t;
73       res.node->next = i->node;
74       res.node->prev = i->node->prev;
75       res.node->prev->next = res.node;
76       i->node->prev = res.node;
```

```
77        l->size++;
78        return res;
79      }
80
81      Iterator Delete(List *l, Iterator *i)
82      {
83        Iterator res = Last(l);
84        if (l->head == NULL) {
85          printf("List doesn't exists\n");
86          return res;
87        }
88        if (Equal(i, &res))
89        return res;
90        res.node = i->node->next;
91        res.node->prev = i->node->prev;
92        i->node->prev->next = res.node;
93        l->size--;
94        free(i->node);
95        i->node = NULL;
96        return res;
97      }
98
99      int Size(const List *l)
100     {
101       if (l->head == NULL) {
102         printf("List doesn't exists\n");
103         return 0;
104       }
105       return l->size;
106     }
107
108     void print_list(const List *l)
109     {
110       if (l->head == NULL) {
111         printf("List doesn't exists\n");
112         return;
113       }
114       Iterator fst = First(l);
115       Iterator lst = Last(l);
116       for (Iterator i = First(l); !Equal(&i, &lst); Next(&i)) {
117         printf("%.4f -> ", Fetch(&i));
118       }
119       printf("NULL\n");
120     }
121
122     void Task(List *l)
123     {
124       if (l->head == NULL || l->head == NULL) {
125         printf("List doesn't exists\n");
126       }
127       Iterator lst = Last(l);
128       Iterator fst = First(l);
129       float tmp;
130
131       if (Equal(&lst, &fst) || Size(l) < 2) {
132         printf("Nesessary nodes doesn't exists\n");
133         return;
134       }
135       if (Equal(Next(&fst), Prev(&lst)) || Size(l) == 2) {
136         lst = *Prev(&lst);
137         tmp = fst.node->data;
138         fst.node->data = lst.node->data;
139         lst.node->data = tmp;
140         return;
141       }
142       lst = *Prev(&lst);
```

```
143       tmp = fst.node->data;
144       fst.node->data = lst.node->data;
145       lst.node->data = tmp;
146      }
147
148      void Destroy(List *l)
149      {
150       struct Item *i = l->head->next;
151       while (i != l->head) {
152         struct Item *pi = i;
153         i = i->next;
154         free(pi);
155       }
156       free(l->head);
157       l->head = 0;
158       l->size = 0;
159      }
```

Листинг 4: main.c

```
1     #include <stdio.h>
2     #include <stdlib.h>
3     #include "list.h"
4
5     int main(int argc, char const *argv[])
6     {
7       List *l = malloc(sizeof(List));
8       Create(l);
9       int num = 0;
10      int choose;
11      int g = 1;
12      while (g) {
13        printf("1. Print list\t 2. Insert in list\t 3. Delete from list\t 4. Size\t 5.
              Task\t other. Exit\n");
14        scanf("%d", &choose);
15        switch (choose) {
16          case 1: {
17            print_list(l);
18            break;
19          }
20          case 2: {
21            Iterator i = Last(l);
22            float val;
23            printf("Write value\n");
24            scanf("%f", &val);
25            if (Size(l) != 0) {
26              printf("Write the number where you want insert element\n");
27              scanf("%d", &num);
28              if (0 <= num && num <= Size(l)) {
29                i = search_prev(l, num);
30              } else {
31                printf("Element with %d number doesn't exists\n", num);
32                break;
33              }
34            }
35            Insert(l, &i, val);
36            break;
37          }
38          case 3: {
39            float prev;
40            printf("Write number of deleted element\n");
41            scanf("%d", &num);
42            if (Size(l) == 1) {
43              Destroy(l);
44              Create(l);
45              break;
```

```
46              }
47              if (0 < num && num < Size(l)) {
48                Iterator i = search_prev(l, num);
49                Delete(l, &i);
50              } else {
51                printf("Element with %d number doesn't exists\n", num);
52              }
53              break;
54            }
55            case 4: {
56              printf("Size is %d\n", Size(l));
57              break;
58            }
59            case 5: {
60              Task(l);
61              break;
62            }
63            default: {
64              Destroy(l);
65              g = 0;
66              break;
67            }
68
69          }
70        }
71        free(l);
72        return 0;
73      }
```

Листинг 5: Makefile

```
1     CC = gcc
2     CFLAGS ?= -g -Wall -Wextra -pedantic -std=c99 -w -pipe -O3 -lm
3     main: main.o list.o
4     $(CC) -o main main.o list.o
5
6     main.o: main.c
7     $(CC) $(CFLAGS) -c main.c
8
9     list.o: list.c list.h
10    $(CC) $(CFLAGS) -c list.c
11
12    clean:
13    rm -rf *.o main
```

9 Вывод

Я научился обрабатывать линейные списки на Си.