Отчет по лабораторной работе N 25-26 по курсу

"Фундаментальная информатика"

Студент группы: М80-107Б-21, Павлов Иван Дмитриевич
Контакты: pavlov.id.2003@gmail.com
Работа выполнена: 08.04.2022
Преподаватель: Найденов Иван Евгеньевич

1 Тема

Автоматизация сборки программ модульной структуры на языке Си с использованием утилиты *make*. Абстрактные типы данных. Модульное программирование на языке Си.

2 Цель работы

Изучить принцип работы утилиты make. Составить Makefile для модульной программы. Составить и отладить модуль определений и модуль реализации для абстрактного типа данных. Составить программный модуль, сортирующий экземпляр указанного типа данных заданным методом.

3 Задание

- 25 ЛР: Составить Makefile для сборки задания из 26 ЛР;
- 26 ЛР: Реализовать линейный список и его сортировку методом вставки.

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

6.1 Реализация структуры данных

В данной ЛР представлена реализация двусвязного кольцевого списка. Данную реализацию линейного списка я выбрал за возможность доступа к предыдущему элементу, что упрощает алгоритм сортировки вставкой. Изначально созданная структура представляет собой терминирующий элемент, указатели *prev* и *next* в ней ссылаются на саму структуру.

6.2 Реализация процедур АТД

- *void* create: создаем пустой список; память под терминирующий элемент не выделяем, указатели *prev* и *next* указывают на саму структуру. Выполняется в начале программы;
- *void* insert: добавляем новый узел между двумя, идущими подряд (используем ссылку на *next*);

- *void* list_remove: удаляем указанный элемент с помощью ссылок на *prev* и *next*;
- *node** push_front: вставка нового элемента в начало. Выделяем память под новый узел, присваиваем ссылке на него указанное значение, выполняем *insert* для исходного списка и нового узла. Возвращаем ссылку. Работает за $O(1)$;
- *node** push_back: используя свойства кольцевого списка, вызываем push_front от list->prev. Работает за $O(1)$;
- *void* list_delete: удаляет узел с помощью list_remove и чистит память;
- *void* pop_front: если список не пустой, вызывает list_delete от *list->next*. Работает за $O(1)$;
- *void* pop_back: если список не пустой, вызывает list_delete от *list->prev*. Работает за $O(1)$;
- *int* is_empty: проверяет список на пустоту: если *prev* и *next* указывают на саму структуру, то возвращает *true*;
- *void* print: пробегаем циклом по узлам списка и выводим их значения на экран. Работает за $O(n)$;
- *void* erease: удаляет элемент по его значению. Поиск элемента в списке за $O(n)$ и удаление функцией list_delete за $O(1)$. Итого, $O(n)$;
- *void* list_insert: добавляет элемент перед заданным. Функция содержит поиск, выделение памяти под новую ссылку и insert от ссылки и предыдущего элемента. Соответственно, $O(n)$;
- *int* size: циклом подсчитываем количество узлов и умножаем на размер узла. $O(n)$;
- *void* destroy: удалить структуру и освободить память. Выполняется в конце программы.

6.3 Реализация вспомогательной процедуры и сортировки вставкой

- *void* procedure: запоминаем значение элемента списка, создаем ссылку на предыдущий элемент, пока предыдущий элемент больше значения элемента ссылки (или не терминирующий), меняем местами их значения;
- *void* insertion_sort: сортировка вставкой. Циклом пробегаем список до конца и вызываем *procedure*. Работает за $O(n^2)$.

6.4 Сборка программы

Листинг 1: 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
```

В Makefile указываем компилятор gcc со стандартными флагами, в исполняемый файл main собираем скомпилированные из main.c, list.c и list.h файлы main.o и list.o. Реализуем make clean, которая удаляет все исполняемые файлы.

7 Распечатка протокола

Листинг 2: list.h

```
#ifndef list_h
#define list_h

typedef struct node
{
  int key;
  struct node *prev;
  struct node *next;
} node;

void create(node *list);

void insert(node *m, node *p);

void list_remove(node *t);

node *push_front(node *list, int value);

node *push_back(node *list, int value);

void list_delete(node *list);

void pop_front(node *list);

void pop_back(node *list);

int is_empty(node * list);

void print(node *list);

void erease(node *list, int value);

void list_insert(node *list, int k, int value);

void destroy(node *list);

int size(node *list);

void procedure(node* list, node* tmp);

void insertion_sort(node* list);

#endif
```

Листинг 3: list.c

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

void create(node *list)
{
  list->next = list;
  list->prev = list;
}

void insert(node *m, node *p)
{
  node *n = m->next;
  p->prev = m;
  p->next = n;
  m->next = p;
```

```
17        n->prev = p;
18      }
19
20      void list_remove(node *t)
21      {
22        node *s = t->prev;
23        node *u = t->next;
24        s->next = u;
25        u->prev = s;
26      }
27
28      node *push_front(node *list, int value)
29      {
30        node *tmp = malloc(sizeof(node));
31        tmp->key = value;
32        insert(list, tmp);
33        return tmp;
34      }
35
36      node *push_back(node *list, int value)
37      {
38        push_front(list->prev, value);
39      }
40
41      void list_delete(node *list)
42      {
43        list_remove(list);
44        free(list);
45        return;
46      }
47
48      void pop_front(node *list)
49      {
50        if (list->next != list) {
51          list_delete(list->next);
52        }
53        return;
54      }
55
56      void pop_back(node *list)
57      {
58        if (list->prev != list) {
59          list_delete(list->prev);
60        }
61        return;
62      }
63
64      int is_empty(node * list)
65      {
66        return list->prev == list && list->next == list;
67      }
68
69      void print(node *list)
70      {
71        for (node *tmp = list->next; tmp != list; tmp = tmp->next) {
72          printf("%d ", tmp->key);
73        }
74        printf("\n");
75      }
76
77      void erease(node *list, int value)
78      {
79        for (node *tmp = list->next; tmp != list; tmp = tmp->next) {
80          if (tmp->key == value) {
81            list_delete(tmp);
82            return;
```

```
        }
      }
    }

    void list_insert(node *list, int k, int value)
    {
      for (node *tmp = list->next; tmp != list; tmp = tmp->next) {
        if (tmp->key == k) {
          node *res = malloc(sizeof(node));
          res->key = value;
          insert(tmp->prev, res);
          return;
        }
      }
    }

    void destroy(node *list)
    {
      while(!is_empty(list)) {
        pop_front(list);
      }
    }

    int size(node *list)
    {
      int cnt = 0;
      for (node *tmp = list->next; tmp != list; tmp = tmp->next) {
        cnt++;
      }
      return cnt * sizeof(node);
    }

    void procedure(node *list, node* tmp)
    {
      int value = tmp->key;
      node *tmp_prev = tmp->prev;
      while ((tmp_prev != list) && (tmp_prev->key > value)) {
        tmp_prev->next->key = tmp_prev->key;
        tmp_prev = tmp_prev->prev;
      }
      tmp_prev->next->key = value;
    }

    void insertion_sort(node* list)
    {
      node *tmp = NULL;
      for (tmp = list->next; tmp != list; tmp = tmp->next)
      {
        procedure(list, tmp);
      }
    }
```

Листинг 4: main.c

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

    int main() {
      node root;
      node *d = &root;
      create(d);
      int choose, flag = 1;
      int value;
      int key;
      while(flag) {
```

```
13          printf("1: print;\t2: print size;\t3: push front;\t4: push back;\t5: insert;\t6
                : is empty;\n");
14          printf("7: pop front;\t8: pop back;\t9: erase;\t10: procedure;\t11: sort;\
                tother: exit;\n");
15          scanf("%d", &choose);
16          switch(choose) {
17            case 1:
18            print(d);
19            break;
20            case 2:
21            printf("size is %d\n", size(d));
22            break;
23            case 3:
24            printf("enter value: ");
25            scanf("%d", &value);
26            push_front(d, value);
27            break;
28            case 4:
29            printf("enter value: ");
30            scanf("%d", &value);
31            push_back(d, value);
32            break;
33            case 5:
34            printf("enter key: ");
35            scanf("%d", &key);
36            printf("enter value: ");
37            scanf("%d", &value);
38            list_insert(d, key, value);
39            break;
40            case 6:
41            if (is_empty(d)) {
42              printf("Yes\n");
43            } else {
44              printf("No\n");
45            }
46            break;
47            case 7:
48            pop_front(d);
49            break;
50            case 8:
51            pop_back(d);
52            break;
53            case 9:
54            printf("enter key: ");
55            scanf("%d", &key);
56            erease(d, key);
57            break;
58            case 10:
59            procedure(d, d->prev);
60            break;
61            case 11:
62            insertion_sort(d);
63            break;
64            default:
65            flag = 0;
66            break;
67          }
68        }
69        destroy(d);
70      }
```

8 Вывод

В данной ЛР я познакомился с модульным программированием, сборкой программы, реализовал абстрактный тип данных - структуру линейный список и сортировку вставкой для него.