Московский Авиационный

Институт

(Национальный Исследовательский Университет)

Институт №8

«Фундаментальная информатика и информационные технологии»

КУРСОВОЙ ПРОЕКТ

по курсу «Практикум на ЭВМ»

II семестр

«Линейные списки»

Выполнил: студент

группы M8O-113Б-21

Хасанов Даниил Рафаилович

Преподаватель:

Довженко Анастасия Александровна

Работа сдана

«\_\_»\_\_\_\_\_\_\_\_\_\_\_2022

Оценка\_\_\_\_\_\_

**Задание**

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 шести стандартных  
действий:  
1.Печать списка  
2. Вставка нового элемента в список  
3. Удаление элемента из списка  
4. Подсчет длины списка

**Вариант 7.3.12**

Тип элемента списка: 7.Машинное слово.

Вид списка: 3.Линейный однонаправленный с барьерным элементом.

Нестандартное действие: 12.Проверить упорядоченность элементов списка.

**Алгоритм решения**

Проект состоит из восьми частей:

1. main.c – основной файл программы;
2. iterator.c, iterator.h, struct.c, struct.h для реализации динамической структуры данных;
3. machine\_word.c, machine\_word.h для нормализации машинного слова до 8 байт;
4. makefile – для сборка программы.

После сборки программы в терминал выводиться Menu:

Menu:

1) Add an item to the end of the list

2) Remove an item from the list

3) Output a list

4) Print the length of the list

5) Completing a variant task

6) Menu

7) Delete the list and terminate the program

Рассмотрим функции реализованные в программе.

Файл struct.c:

1. listCreate() - Создание списка. Выделяем память для нашего списка и устанавливаем барьерное значение, которое также будет помогать нам в дальнейшем
2. listPrint() - Печать списка.
3. listInsert() - Вставка элемента в конец списка. Вводим значение элемента, которое хотим вставить. И пока не дойдем до барьерного элемента, идем вперед, чтобы вставить наш элемент.
4. listRemove() - Удаление элемента из списка. Получаем на вход элемент, который хотим удалить. Если такого элемента не существует, предупреждаем об ошибке.
5. listLen() - Вычисление длины списка. Идём вперед до барьерного элемента и каждый раз прибавляем к счетчику +1.
6. execute() - Функция варианта. Пока существует следующий элемент и он не равен барьерному значение сравниваем текущий и следующий элемент списка, переставляя итератор вперёд после каждой проверки.
7. equality\_data() – Функция для проверки равенства машинных слов.
8. comparison\_data() – Функция сравнения элементов, используемая в функции execute.

Файл iterator.c:

1. IteratorCreate() - Создание итератора. Создаем наш итератор, выделяем память и ставим его в начало для дальнейшего передвижения по списку
2. iteratorNext() - Получение следующего значения в списке элементов
3. iteratorGet() - Получение того значения, на котором остановились, возвращаем его.
4. IteratorSet() - Установка итератора в определенное значение.Устанавливаем итератор в нужное нам значение

Файл machine\_word.c:

1. scan\_str() – считывает строку из 1 и 0;
2. conversion () – переводит полученную строку из двоичной системы в десятичную;
3. get\_word() – даёт указатель на массив состоящий из байтов вводимого машинного слова;
4. print\_word() – печатает машинное слово.

**Листинг программы**

Main.c

#include <stdio.h>

#include <stdlib.h>

#include <inttypes.h>

#include "struct.h"

#include "iterator.h"

#include "machine\_word.h"

// printf("%ld\n", 1234 >> (0 \* 8) & 255); // 0 for 1 byte 1 for 2 byte etc

//обработать ввод без ведущих нулей перевести в 10 систему и брать байт

void menu()

{

printf("Menu:\n");

printf("1) Add an item to the end of the list\n");

printf("2) Remove an item from the list\n");

printf("3) Output a list\n");

printf("4) Print the length of the list\n");

printf("5) Completing a variant task\n");

printf("6) Menu\n");

printf("7) Delete the list and terminate the program\n");

}

int main(void)

{

list \*lst = listCreate();

listIterator \*it = iteratorCreate(lst);

char \*str = (char \*)malloc(SIZE\_STR \* sizeof(char));

char state, space = ' ';

int index;

menu();

while (1)

{

scanf("%c%c", &state, &space);

iteratorSet(lst->head, it);

if (state == '\n' || state == ' ')

continue;

switch (state)

{

case '1':

printf("Please enter a machine word :\n");

str = (char \*)realloc(str, SIZE\_STR \* sizeof(char));

scan\_str(str);

listInsert(lst, get\_word(str, conversion(str)));

printf("Machine word added\n");

break;

case '2':

printf("Please enter the index of the element you want to delete :\n");

str = (char \*)realloc(str, SIZE\_STR \* sizeof(char));

scan\_str(str);

listRemove(lst, get\_word(str, conversion(str)));

break;

case '3':

for (int i = 0; i < listLen(lst); i++)

{

printf("-----------------------");

}

printf("\n");

for (int i = 0; i < listLen(lst); i++)

{

printf(" ");

}

printf("LIST");

for (int i = 0; i < listLen(lst); i++)

{

printf(" ");

}

printf("\n");

for (int i = 0; i < listLen(lst); i++)

{

printf("-----------------------"); // 22

}

printf("\n");

listPrint(lst);

break;

case '4':

printf("Length of the list: %d\n", listLen(lst));

break;

case '5':

if (!execute(lst, listLen(lst)))

{

printf("List is not sorted!\n");

}

else

printf("List is sorted!\n");

break;

case '6':

menu();

break;

case '7':

free(it);

free(lst);

return 0;

default:

printf("There is no action with this number, select a command from the menu\n");

break;

}

}

}

Struct.h

#ifndef STRUCT\_H

#define STRUCT\_H

#include <inttypes.h>

typedef uint8\_t elem\_type;

typedef struct \_list\_node list\_node;

typedef struct \_list\_node

{

elem\_type \*data;

list\_node \*next;

} list\_node;

typedef struct

{

list\_node \*head;

} list;

int comparison\_data(elem\_type \*data1, elem\_type \*data2);

int equality\_data(elem\_type \*data1, elem\_type \*data2);

list \*listCreate();

void listPrint(list \*lst);

void listInsert(list \*lst, elem\_type \*data);

void listRemove(list \*lst, elem\_type \*data);

int listLen(list \*lst);

int execute(list \*lst, int size);

#endif

Struct.c

#include <stdio.h>

#include <stdlib.h>

#include "struct.h"

#include "iterator.h"

#include "machine\_word.h"

uint8\_t \*barrier;

int comparison\_data(elem\_type \*data1, elem\_type \*data2)

{

for (int i = 0; i < 8; i++)

{

if (data1[i] > data2[i])

{

printf("%d %d\n", data1[i], data2[i]);

return 0;

}

}

return 1;

}

int equality\_data(elem\_type \*data1, elem\_type \*data2)

{

for (int i = 0; i < 8; i++)

{

if (data1[i] != data2[i])

return 0;

}

return 1;

}

list \*listCreate()

{

printf("Set up a barrier: \n");

char \*str = (char \*)malloc(SIZE\_STR \* sizeof(char));

list \*lst = (list \*)malloc(sizeof(list));

lst->head = (list\_node \*)malloc(sizeof(list\_node));

lst->head->next = NULL;

scan\_str(str);

lst->head->data = get\_word(str, conversion(str));

barrier = get\_word(str, conversion(str));

return lst;

}

void listPrint(list \*lst)

{

listIterator \*it = iteratorCreate(lst);

if (it->node)

{

while (it->node != NULL)

{

if (!(equality\_data(it->node->data, barrier)))

print\_word(it->node->data);

it->node = it->node->next;

}

printf("BARRIER");

printf("\n");

}

}

void listInsert(list \*lst, elem\_type \*data)

{

listIterator \*it = iteratorCreate(lst);

if (it->node)

{

while (it->node->next)

{

iteratorNext(it);

}

list\_node \*tail = (list\_node \*)malloc(sizeof(list\_node));

tail->next = NULL;

tail->data = barrier;

it->node->data = data;

it->node->next = tail;

}

else

printf("The list does not exist!\n");

free(it);

}

void listRemove(list \*lst, elem\_type \*data)

{

int flag = 0;

listIterator \*it = iteratorCreate(lst);

list\_node \*prew;

if (it->node)

{

if (equality\_data(it->node->data, data))

{

flag = 1;

lst->head = it->node->next;

free(it->node);

it->node = NULL;

}

else

{

while (it->node->next)

{

if (equality\_data(it->node->next->data, data))

{

prew = it->node;

iteratorNext(it);

flag = 1;

if (it->node->next != NULL)

{

prew->next = it->node->next;

}

free(it->node);

it->node = NULL;

break;

}

iteratorNext(it);

}

}

if (flag)

printf("Item removed\n");

else

printf("The item is not in the list\n");

}

else

printf("The list does not exist!\n");

free(it);

}

int listLen(list \*lst)

{

int count = 0;

listIterator \*it = iteratorCreate(lst);

iteratorSet(lst->head, it);

if (it->node)

{

while (it->node->next && !(equality\_data(it->node->next->data, barrier)))

{

count++;

iteratorNext(it);

}

}

else

printf("The list does not exist!\n");

free(it);

count++;

return count;

}

int execute(list \*lst, int size)

{

listIterator \*it = iteratorCreate(lst);

while (it->node->next && !(equality\_data(it->node->next->data, barrier)))

{

if (!(comparison\_data(it->node->data, it->node->next->data)))

return 0;

iteratorNext(it);

}

return 1;

}

Iterator.h

#ifndef ITERATOR\_H

#define ITERATOR\_H

#include "struct.h"

typedef struct \_listIterator

{

list\_node \*node;

} listIterator;

listIterator \*iteratorCreate(list \*lst);

void iteratorNext(listIterator \*it);

list\_node \*iteratorGet(listIterator \*it);

void iteratorSet(list\_node \*l, listIterator \*it);

#endif

Iterator.c

#include <stdlib.h>

#include <stdio.h>

#include "iterator.h"

listIterator \*iteratorCreate(list \*lst)

{

if (lst != NULL)

{

listIterator \*it = (listIterator \*)malloc(sizeof(listIterator));

it->node = lst->head;

return it;

}

else

return NULL;

}

void iteratorNext(listIterator \*it)

{

if (it->node == NULL)

{

printf("End of the list\n");

}

else

{

it->node = it->node->next;

}

}

list\_node \*iteratorGet(listIterator \*it)

{

return it->node;

}

void iteratorSet(list\_node \*lst, listIterator \*it)

{

it->node = lst;

}

Machine\_word.h

#ifndef \_WORD\_H\_

#define \_WORD\_H\_

#include <inttypes.h>

#include <stdlib.h>

#define SIZE\_STR 64

#define SIZE\_BYTE 8

void scan\_str(char \*str);

uint64\_t conversion(char \*str);

uint8\_t \*get\_word(char \*str, uint64\_t digit);

void print\_word(uint8\_t \*word);

#endif //\_WORD\_H\_

Machine\_word.c

#include "machine\_word.h"

#include <stdio.h>

#include <stdlib.h>

#include <inttypes.h>

#include <stdbool.h>

#include <string.h>

#include <math.h>

void scan\_str(char \*str)

{

char symb = ' ';

uint8\_t count\_str = 0;

bool end\_str = true, flag = false;

while (symb != '\n' && count\_str < 255)

{

scanf("%c", &symb);

if (symb == '\n' || symb == ' ')

{

end\_str = false;

}

if (symb == '1')

flag = true;

if (end\_str && flag)

{

str[count\_str] = symb;

count\_str++;

}

}

}

uint64\_t conversion(char \*str)

{

uint64\_t digit = 0;

for (int i = strlen(str) - 1; i > -1; i--)

{

if (str[i] == '1')

digit += pow(2, strlen(str) - i - 1);

}

return digit;

}

uint8\_t \*get\_word(char \*str, uint64\_t digit)

{

uint8\_t \*word = (uint8\_t \*)malloc(SIZE\_BYTE \* sizeof(uint8\_t));

for (int i = SIZE\_BYTE - 1; i >= 0; i--)

{

word[i] = digit >> ((8 - i - 1) \* 8) & 255;

}

return word;

}

void print\_word(uint8\_t \*word)

{

printf("[ ");

for (int i = 0; i < SIZE\_BYTE; i++)

{

printf("%d ", word[i]);

}

printf("] -> ");

}

Makefile

TARGET = run

CC = gcc

SRC = $(wildcard \*.c)

OBJ = $(patsubst %.c, %.o, $(SRC))

$(TARGET) : $(OBJ)

$(CC) $(OBJ) -o $(TARGET)

%.o : %.c

$(CC) -c $< -o $@

clean :

del $(TARGET).exe \*.o

Работа программы

PS D:\Лабы\Курсовая2семестр\КП8> make

gcc -c iterator.c -o iterator.o

gcc -c main.c -o main.o

gcc -c machine\_word.c -o machine\_word.o

gcc -c struct.c -o struct.o

PS D:\Лабы\Курсовая2семестр\КП8> .\run.exe.

PS D:\Лабы\Курсовая2семестр\КП8> .\run.exe

Set up a barrier:

Menu:

1) Add an item to the end of the list

3) Output a list

4) Print the length of the list

5) Completing a variant task

6) Menu

7) Delete the list and terminate the program

There is no action with this number, select a command from the menu

PS D:\Лабы\Курсовая2семестр\КП8> make clean

del run.exe \*.o

PS D:\Лабы\Курсовая2семестр\КП8> make

gcc -c iterator.c -o iterator.o

gcc -c main.c -o main.o

gcc -c machine\_word.c -o machine\_word.o

gcc -c struct.c -o struct.o

gcc iterator.o main.o machine\_word.o struct.o -o run

PS D:\Лабы\Курсовая2семестр\КП8> .\run.exe

Set up a barrier:

Menu:

1) Add an item to the end of the list

2) Remove an item from the list

3) Output a list

4) Print the length of the list

5) Completing a variant task

6) Menu

7) Delete the list and terminate the program

1

Please enter a machine word :

1111

Machine word added

3

-----------------------

LIST

-----------------------

[ 0 0 0 0 0 0 0 15 ] -> BARRIER

1

Please enter a machine word :

111111

Machine word added

1

PS D:\Лабы\Курсовая2семестр\КП8> make

gcc -c iterator.c -o iterator.o

gcc -c main.c -o main.o

gcc -c machine\_word.c -o machine\_word.o

gcc -c struct.c -o struct.o

gcc iterator.o main.o machine\_word.o struct.o -o run

PS D:\Лабы\Курсовая2семестр\КП8> .\run.exe

Set up a barrier:

Menu:

1) Add an item to the end of the list

2) Remove an item from the list

3) Output a list

4) Print the length of the list

5) Completing a variant task

6) Menu

7) Delete the list and terminate the program

1

Please enter a machine word :

111

Machine word added

1

Please enter a machine word :

1010011

Machine word added

3

----------------------------------------------

LIST

----------------------------------------------

[ 0 0 0 0 0 0 0 7 ] -> [ 0 0 0 0 0 0 0 83 ] -> BARRIER

2

Please enter the index of the element you want to delete :

111

Item removed

3

-----------------------

LIST

-----------------------

[ 0 0 0 0 0 0 0 83 ] -> BARRIER

1

Please enter a machine word :

11111111111111

Machine word added

3

----------------------------------------------

LIST

----------------------------------------------

[ 0 0 0 0 0 0 0 83 ] -> [ 0 0 0 0 0 0 63 255 ] -> BARRIER

4

Length of the list: 2

5

List is sorted!

2

Please enter the index of the element you want to delete :

11

The item is not in the list

3

----------------------------------------------

LIST

----------------------------------------------

[ 0 0 0 0 0 0 0 83 ] -> [ 0 0 0 0 0 0 63 255 ] -> BARRIER

7

PS D:\Лабы\Курсовая2семестр\КП8>

**Вывод:** данная курсовая работа была сложной, в особенности реализация машинного слова, но из-за этого и интересной. Умение работать и реализовывать линейные списки хороший навык, который пригодиться мне в будущем.