Отчёт по лабораторной работе № 25-26 по курсу Языки и методы программирования

Студент группы М8О-104Б-19 Черница Артём Александрович, № по списку 22

Контакты www, e-mail, icq, skype aachernitsa@mai.education

Работа выполнена: « 1 » июня 20 20 г.

Преподаватель: Титов В.К. каф.806 Вычислительная математика и программирование

Входной контроль знаний с оценкой _____

Отчёт сдан « » _____201 __ г., итоговая оценка _____

Подпись преподавателя _____

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

2. **Цель работы:** Составить и отладить модуль определений и модуль реализации по заданной схеме модуля определений для абстрактного типа данных. Составить программный модуль, сортирующий экземпляр указанного абстрактного типа данных заданным методом, используя только операции, импортированные из модуля.

3. **Задание** (*вариант №* 22): Стек. Поиск двух подряд идущих элементов, первый из которых больше второго, перестановка их местами, если найдены. Сортировка методом пузырька.

4. **Оборудование(лабораторное)**:
ЭВМ _____, процессор _____, имя узла сети_____ с ОП _____ Мб,
НМД_____Мб. Терминал_____адрес _____. Принтер _____
Другие устройства _____

Оборудование ПЭВМ студента, если использовалось:
Процессор Intel Core i5 1.6GHz с ОП 8192 Мб, НМД 128000 Мб. Монитор _____
Другие устройства _____

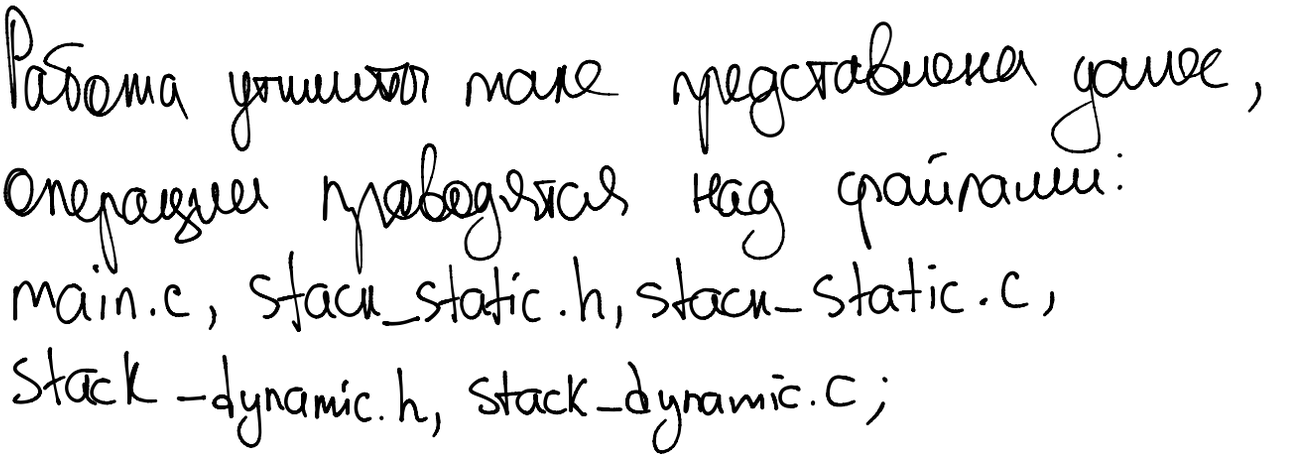
5. **Программное обеспечение(лабораторное):**
Операционная система семейства _____, наименование _____ версия _____
интерпретатор команд _____ версия _____
Система программирования _____ версия _____
Редактор текстов _____ версия _____
Утилиты операционной системы_____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных _____

Программное обеспечение ЭВМ студента, если использовалось:
Операционная система семейства MacOS, наименование Catalina версия 10.15.4
интерпретатор команд zsh версия _____.
Система программирования C версия _____
Редактор текстов vim версия _____
Утилиты операционной системы gcc, cat
Прикладные системы и программы make
Местонахождение и имена файлов программ и данных на домашнем компьютере_____

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

В данной работе используется модульная система: основной файл main.cpp подключает заголовочные файлы stack_static.h и static_dynamic.h, реализация которых находится в файлах stack_static.c и stack_dynamic.c соответственно. С помощью утилиты make каждый раз компилируются только изменённые файлы, что очень удобно. Используются заголовочные файлы по очереди, сначала реализация отображением на массив, затем на динамическую структуру (односвязный список). Сам же алгоритм обмена элементов работает следующий образом: т.к. используется структура Стек, то нельзя просто переставить элементы, потому что формально у нас нет к ним доступа. Для этого сначала заводится временный стек, который будет хранить элементы, которые мы обменяли (в дальнейшем его можно расширить на реализацию, где находятся все пары в текущем направлении за проход, а не только одна). Последний добавленный элемент (он же первый) выталкивается из оригинального стека помощью pop(), а значение перед этим помещается в буферную переменную командой peak(). На следующем шаге идёт проверка, если следующий элемент больше, то мы сначала записываем во временной массив элемент меньший (буферная переменная), а затем больший. Если же условие не выполнилось, то записываем в таком порядке в каком брали. После этого выходим из цикла, и все элементы которые мы вытолкнули теперь надо втолкнуть с помощью push() обратно из временного стека. Таким образом, мы переставляем элементы. Сортировка же пузырьком выполняется просто: мы вызываем команду перестановки в цикле от 0 до size (размер стека) и для каждого такого шага внутри запускаем цикл от i до size и вызываем перестановку. Массив будет отсортирован.
Отличие реализации отображением на массив и отображением на список отличается немного отличными заголовочными файлами и разницей с обращением со структурами (копирование, вставка, добавление, выделение памяти), в остальном они идентичны. У отображения на массив более простая реализация, а также более высокая скорость работы, по сравнению с отображением на список, но список потребляет в общем случае меньше памяти для работы над данной структуры, которая у меня получилась.
Важно отметить, что для последовательно добавленных элементов 5 2 4 1 3 стек будет выведен как 3 1 4 2 5, т.е. как бы обозначить, что последний добавленный элемент будет взят или удален первым.
Рекурсия не была использована, т.к. алгоритм сортировки пузырьком эффективнее реализуется итеративным способом, а для остальных операций нецелесообразно было использовать рекурсию.

7. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].



Для проверки работоспособности программы были использованы следующий тесты:
4 2 3 5 1 -> 1 2 3 4 5 -> вывод 5 4 3 2 1, т.к. сортируется по возрастанию и самый большой становится первым на peak() или pop().
49 95 58 6 29 69 81 78 11 37 -> вывод 95 81 78 69 58 49 37 29 11 6
Для чисел, сгенерированных ГПСЧ требуется ручная проверка.

Пункты 1-7 отчета составляются ***строго до*** *начала лабораторной работы.*

Допущен к выполнению работы. Подпись преподавателя _____

```
198097@client17:~/Lab26$ cat head.txt
**********************************
*                                *
*    Лабораторная работа №25-26  *
*  Автоматизация сборки программ *
* модульной структуры на языке Си*
*  с использованием утилиты make.*
*   Абстрактные типы данных.     *
*           Рекурсия.            *
*   Модульное программирование   *
*          на языке Си.          *
*  Выполнил студент группы 104   *
*  Черница Артём Александрович   *
*                                *
**********************************
198097@client17:~/Lab26$ cat stack_static.h
#ifndef LAB26_STACK_STATIC_H
#define LAB26_STACK_STATIC_H

typedef struct {
   int size;
   int *array;
} SStack;

void init(SStack *stack);
int empty(const SStack *stack);
void push(SStack *stack, int value);
void pop(SStack *stack);
int peak(const SStack *stack);
int size(SStack *stack);
void print(const SStack *stack);
void spswap(SStack *stack); // special swap
void sort(SStack *stack);

#endif //LAB26_STACK_STATIC_H
198097@client17:~/Lab26$ cat stack_static.c
#include "stack_static.h"
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void init(SStack* stack) {
   stack->size = 0;
   stack->array = NULL;
}

int empty(const SStack *stack) {
   return stack->size == 0;
}

void push(SStack *stack, int value) {
   if (empty(stack)) {
      stack->array = malloc(sizeof(int));
      stack->size = 1;
      stack->array[stack->size-1] = value;
      return;
   }
```

```
   stack->array = realloc(stack->array, (stack->size + 1) * sizeof(int));
   stack->size += 1;
   stack->array[stack->size - 1] = value;
}

void pop(SStack *stack) {
   if (empty(stack)) {
      return;
   }

   stack->array = realloc(stack->array, (stack->size - 1) * sizeof(int));
   stack->size -= 1;
}

int peak(const SStack *stack) {
   if (empty(stack)) {
      return 0;
   }

   return stack->array[stack->size - 1];
}

int size(SStack *stack) {

   return stack->size;
}

void print(const SStack *stack) {
   printf("[---> ");
   for (int i = 0; i < stack->size; ++i) {
      printf("%-4d", stack->array[stack->size - i - 1]);
   }
   printf("]\n");
}

void spswap(SStack *stack) {
   SStack temp; init(&temp);
   SStack copy; init(&copy);
   copy.array = malloc(sizeof(int) * stack->size);
   memcpy(copy.array, stack->array, stack->size * sizeof(int));
   copy.size = stack->size;

   while (!empty(&copy)) {
      int buf = peak(&copy);
      pop(&copy);
      if (peak(&copy) > buf) {
         push(&temp, peak(&copy));
         push(&temp, buf);
         pop(&copy);
         break;
      }
      else {
         push(&temp, buf);
      }
   }

   while (!empty(&temp)) {
      push(&copy, peak(&temp));
      pop(&temp);
```

```
   }

   memcpy(stack->array, copy.array, stack->size * sizeof(int));
   free(temp.array);
   free(copy.array);
}

void sort(SStack *stack) {
   for (int i = 0; i < stack->size; ++i) {
      for (int j = i; j < stack->size; ++j) {
         spswap(stack);
      }
   }
}
198097@client17:~/Lab26$ cat stack_dynamic.h
#ifndef LAB26_STACK_DYNAMIC_H
#define LAB26_STACK_DYNAMIC_H

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

typedef struct {
   int size;
   Item* root;
} DStack;

void init(DStack *stack);
int empty(const DStack *stack);
void push(DStack *stack, int value);
void pop(DStack *stack);
int peak(const DStack *stack);
int size(DStack *stack);
void print(const DStack *stack);
void spswap(DStack *stack); // special swap
void sort(DStack *stack);

#endif //LAB26_STACK_DYNAMIC_H
198097@client17:~/Lab26$ cat stack_dynamic.c
#include "stack_dynamic.h"
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>

void init(DStack *stack) {
   stack->size = 0;
   stack->root = malloc(sizeof(Item));
}

int empty(const DStack *stack) {
   return stack->size == 0;
}

void push(DStack *stack, int value) {
   if (empty(stack)) {
       stack->size = 1;
       if (stack->root == NULL) {
         stack->root = malloc(sizeof(Item));
```

```
        }
        stack->root->value = value;
        return;
    }

    Item *item = malloc(sizeof(Item));
    item = stack->root;
    for (int i = 0; i < stack->size - 1; ++i) {
       item = item->next;
    }
    item->next = malloc(sizeof(Item));
    item->next->value = value;
    stack->size += 1;
//   free(item);
}

void pop(DStack *stack) {
    if (empty(stack)) {
       return;
    }

    if (stack->size == 1) {
       stack->size = 0;
       free(stack->root);
       stack->root = NULL;
       return;
    }

    Item *item = malloc(sizeof(Item));
    item = stack->root;
    for (int i = 0; i < stack->size - 2; ++i) {
       item = item->next;
    }
    free(item->next);
    item->next = NULL;
    stack->size -=1;
}

int peak(const DStack *stack) {
    if (empty(stack)) {
       return 0;
    }

    Item *item = malloc(sizeof(Item));
    item = stack->root;
    for (int i = 0; i < stack->size - 1; ++i) {
       item = item->next;
    }

//   free(item);
    return item->value;
}

int size(DStack *stack) {
    return stack->size;
}

void print(const DStack *stack) {
    if (empty(stack)) {
```

```
      printf("Stack is empty...\n");
      return;
   }

   printf("[---> ");

   DStack copy; init(&copy);
   Item *item = malloc(sizeof(Item));
   item = stack->root;
   for (int i = 0; i < stack->size; ++i) {
      push(&copy, item->value);
      item = item->next;
   }

   DStack temp; init(&temp);
   for (int i = 0; i < stack->size; ++i) {
      push(&temp, peak(&copy));
      pop(&copy);
   }

   item = temp.root;
   for (int i = 0; i < temp.size; ++i) {
      printf("%-4d", item->value);
      item = item->next;
   }

   printf("]\n");
}

void spswap(DStack *stack) {
   DStack temp; init(&temp);
   DStack copy; init(&copy);
   Item *item = malloc(sizeof(Item));
   item = stack->root;
   for (int i = 0; i < stack->size; ++i) {
      push(&copy, item->value);
      item = item->next;
   }
   copy.size = stack->size;

   while (!empty(&copy)) {
      int buf = peak(&copy);
      pop(&copy);
      if (peak(&copy) > buf) {
         push(&temp, peak(&copy));
         push(&temp, buf);
         pop(&copy);
         break;
      }
      else {
         push(&temp, buf);
      }
   }

   while (!empty(&temp)) {
      push(&copy, peak(&temp));
      pop(&temp);
   }
```

```
   int count = stack->size;
   for (int i = 0; i < count; ++i) {
      pop(stack);
   }
   item = copy.root;
   for (int i = 0; i < count; ++i) {
      push(stack, item->value);
      item = item->next;
   }
}

void sort(DStack *stack) {
   for (int i = 0; i < stack->size; ++i) {
      for (int j = i; j < stack->size; ++j) {
         spswap(stack);
      }
   }
}
198097@client17:~/Lab26$ cat Makefile
CCFLAGS = -g -Wall -std=c99

main: main.o stack_static.o
      gcc -o main main.o stack_static.o
main.o: main.c stack_static.h
      gcc -c $(CCFLAGS) main.c
stack_static.o: stack_static.c stack_static.h
      gcc -c $(CCFLAGS) stack_static.c
198097@client17:~/Lab26$ cat MakeFile2
CCFLAGS = -g -Wall -std=c99

main: main.o stack_dynamic.o
      gcc -o main main.o stack_dynamic.o
main.o: main.c stack_dynamic.h
      gcc -c $(CCFLAGS) main.c
stack_dynamic.o: stack_dynamic.c stack_dynamic.h
      gcc -c $(CCFLAGS) stack_dynamic.c
198097@client17:~/Lab26$ cat main.c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
//#include "stack_dynamic.h"
#include "stack_static.h"

int main() {
   SStack sstack;
   init(&sstack);
   push(&sstack, 4);
   push(&sstack, 2);
   push(&sstack, 3);
   push(&sstack, 5);
   push(&sstack, 1);
   print(&sstack);
   sort(&sstack);
   print(&sstack);
   int s = size(&sstack);
   for (int i = 0; i < s; ++i) {
      pop(&sstack);
   }
   push(&sstack, 49);
```

```
   push(&sstack, 95);
   push(&sstack, 58);
   push(&sstack, 6);
   push(&sstack, 29);
   push(&sstack, 69);
   push(&sstack, 81);
   push(&sstack, 78);
   push(&sstack, 11);
   push(&sstack, 37);
   print(&sstack);
   sort(&sstack);
   print(&sstack);

   return 0;
}
198097@client17:~/Lab26$ make
gcc -c -g -Wall -std=c99 main.c
gcc -c -g -Wall -std=c99 stack_static.c
gcc -o main main.o stack_static.o
198097@client17:~/Lab26$ vi main.c
198097@client17:~/Lab26$ cat main.c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include "stack_dynamic.h"
//#include "stack_static.h"

int main() {
   DStack dstack;
   init(&dstack);
   push(&dstack, 4);
   push(&dstack, 2);
   push(&dstack, 3);
   push(&dstack, 5);
   push(&dstack, 1);
   print(&dstack);
   sort(&dstack);
   print(&dstack);
   int s = size(&dstack);
   for (int i = 0; i < s; ++i) {
      pop(&dstack);
   }
   push(&dstack, 49);
   push(&dstack, 95);
   push(&dstack, 58);
   push(&dstack, 6);
   push(&dstack, 29);
   push(&dstack, 69);
   push(&dstack, 81);
   push(&dstack, 78);
   push(&dstack, 11);
   push(&dstack, 37);
   print(&dstack);
   sort(&dstack);
   print(&dstack);

   return 0;
}
198097@client17:~/Lab26$ make --file Makefile2
```

```
gcc -c -g -Wall -std=c99 main.c
gcc -c -g -Wall -std=c99 stack_dynamic.c
gcc -o main main.o stack_dynamic.o
198097@client17:~/Lab26$ make --file Makefile2
make: `main' is up to date.
198097@client17:~/Lab26$ touch main.c
198097@client17:~/Lab26$ make --file Makefile2
gcc -c -g -Wall -std=c99 main.c
gcc -o main main.o stack_dynamic.o
198097@client17:~/Lab26$
```

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

См. далее результат

```
198097@client17:~/Lab26$ cat head.txt
**********************************
*                                *
*    Лабораторная работа №25-26  *
*  Автоматизация сборки программ *
* модульной структуры на языке Си*
*  с использованием утилиты make.*
*   Абстрактные типы данных.     *
*           Рекурсия.            *
*   Модульное программирование   *
*          на языке Си.          *
*  Выполнил студент группы 104   *
*  Черница Артём Александрович   *
*                                *
**********************************
198097@client17:~/Lab26$ make --file Makefile2
make: `main' is up to date.
198097@client17:~/Lab26$ ./main
[---> 1   5   3   2   4   ]
[---> 5   4   3   2   1   ]
[---> 37  11  78  81  69  29  6   58  95  49  ]
[---> 95  81  78  69  58  49  37  29  11  6   ]
198097@client17:~/Lab26$ vi main.c
198097@client17:~/Lab26$ make
gcc -c -g -Wall -std=c99 main.c
gcc -g -Wall -std=c99 -o main main.o stack_static.o
198097@client17:~/Lab26$ ./main
[---> 1   5   3   2   4   ]
[---> 5   4   3   2   1   ]
[---> 37  11  78  81  69  29  6   58  95  49  ]
[---> 95  81  78  69  58  49  37  29  11  6   ]
198097@client17:~/Lab26$ vi main.c
198097@client17:~/Lab26$ cat main.c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
//#include "stack_dynamic.h"
#include "stack_static.h"

int main() {
   SStack sstack;
   init(&sstack);
   srand(time(0));
   for (int i = 0; i < 10; ++i) {
      push(&sstack, rand()%20);
   }
   print(&sstack);
   sort(&sstack);
   print(&sstack);

   return 0;
}
198097@client17:~/Lab26$ make
gcc -c -g -Wall -std=c99 main.c
gcc -g -Wall -std=c99 -o main main.o stack_static.o
198097@client17:~/Lab26$ ./main
[---> 14  2   7   14  9   14  5   13  5   12  ]
[---> 14  14  14  13  12  9   7   5   5   2   ]
198097@client17:~/Lab26$
```

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.

№	Лаб. или дом.	Дата	Время	Событие	Действие по исправлению	Примечание

10. **Замечания автора** по существу работы _____

11. **Выводы**

Познакомился с утилитой make, научился писать сценарии для сборки модульных программ. Реализовал структуру данных стек двумя способами: отображением на массив и на список. Создал функцию для сортировки методом пузырька, с помощью реализованной процедуры обмена. Задачу считаю завершённой.

Недочёты при выполнении задания могут быть устранены следующим образом:_____

Подпись студента