**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Программирование»**

Тема: Стандартная библиотека Си

|  |  |  |
| --- | --- | --- |
| Студент гр. 6303 |  | Шевченко Д.В. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2017

**Цель работы:**

Требуется написать программу, моделирующую работу стека, реализовав перечисленные ниже методы. Программе на вход подается последовательность команд с новой строки, в зависимости от которых программа выполняет ту или иную операцию и выводит результат ее выполнения с новой строки.

**Перечень команд:**

**push n** -  добавляет целое число n в стек. Программа должна вывести "**ok**"

**pop** - удаляет из стека последний элемент и выводит его значение на экран

**top** - программа должна вывести верхний элемент стека н а экран не удаляя его из стека

**size** - программа должна вывести количество элементов в стеке

**exit** - программа должна вывести "**bye**" и завершить работу

Если в процессе вычисления возникает ошибка (например вызов метода **pop** при пустом стеке), программа должна вывести "**error**" и завершиться.

Стек требуется реализовать самостоятельно на базе списка.

**Описание работы программы:**

Программа представлена в виде трёх вспомогательных функций, и одной основной:

1. void push\_cell\_in\_stack(stack \* root\_of\_stack, int number) – получает в качестве аргумента указатель на корневой элемент списка, и добавляет сразу за ним новый элемент, содержащий число number переданное так же в качестве аргумента.
2. int pop\_cell\_from\_stack(stack \* root\_of\_stack) – получает в качестве аргумента указатель на корневой элемент списка, и удаляет первый следующий за ним элемент, затем очищает память. Возвращает число содержавшееся в удаленном элементе.
3. void free\_stack(stack \* root\_of\_stack) – получает в качестве аргумента указатель на корневой элемент списка, очищает всю выделенную в процессе работы программы динамическую память.
4. int top\_cell\_of\_stack(stack \* root\_of\_stack) – получает в качестве аргумента указатель на корневой элемент списка, возвращает число хранимое в элементе следующем сразу за корневым, что эквивалентно верхнему элементу списка.
5. Основная функция int main()

Содержит основную логику программы, считывает строку, в бесконечном цикле сравнивает строки полученные на ввод с доступными для обработки командами, в соответствии с полученными командами вызывает функции описанные выше, выводит количество элементов в списке, или завершает работу программы.

|  |
| --- |
| Приложение 1. Исходный код программы:  // lab №2 "simulation of stack" by Shevchenko Danil, 6303 group  #include <stdlib.h> |
| #include <stdio.h> |
| #include <string.h> |
|  |
| typedef struct stack // структура описывающая элемент стека | |
| { | |
| int store\_in\_cell; // хранимое значение | |
| struct stack \* prev\_cell; // указатель на следующий элемент стека (сверху->вниз) | |
| } stack; | |
|  | |
| void push\_cell\_in\_stack(stack \* root\_of\_stack, int number); // добавление элемента в список | |
|  | |
| int pop\_cell\_from\_stack(stack \* root\_of\_stack); // удаление элемента из списка и возврат хранившегося значения | |
|  | |
| int top\_cell\_of\_stack(stack \* root\_of\_stack); // вывод значения верхнего значения списка | |
|  | |
| void free\_stack(stack \* root\_of\_stack); // очистка всей выделенной под стек памяти | |
|  | |
| int main() | |
| { | |
| stack \* root\_of\_stack = (stack\*)malloc(sizeof(stack)); // создаем корень списка (не является элементом стека) | | |
| root\_of\_stack->prev\_cell = NULL; // задействуется только как основа, по сути -1 элемент списка | | |
| int count\_stack\_cell = 0; // счетчик элементов в стеке | | |
| char command[5]; | | |
| do | |
| { | |
| fgets(command, 5, stdin); // считывание команды | |
| if (!strcmp(command, "push")) | |
| { | |
| int number; | |
| scanf("%d", &number); | |
| printf("ok\n"); | |
| getchar(); | |
| push\_cell\_in\_stack(root\_of\_stack, number); // добавление ранее считанного числа в список (стек) | |
| ++count\_stack\_cell; // увеличение счетчика элементов на 1 | |
| } | |
| else if (!strcmp(command, "pop\n")) | |
| { | |
| if (!count\_stack\_cell) // если счетчик = 0, не существует элемента для удаления | |
| { | |
| printf("error\n"); // ошибка и завершение работы | |
| free\_stack(root\_of\_stack); // очистка памяти | |
| return 0; | |
| } | |
| printf("%d\n", pop\_cell\_from\_stack(root\_of\_stack)); // удаление элемента и вывод удаленного числа | |
| --count\_stack\_cell; // уменьшение счетчика элементов на 1 | |
| } | |
| else if (!strcmp(command, "top\n") && (count\_stack\_cell!=0)) | |
| { | |
| printf("%d\n", top\_cell\_of\_stack(root\_of\_stack)); // вывод значения верхнего элемента стека | |
| } | |
| else if (!strcmp(command, "size")) | |
| { | |
| printf("%d\n", count\_stack\_cell); // вывод кол-ва элементов в стеке | |
| getchar(); | |
| } | |
| else if (!strcmp(command, "exit")) | |
| { | |
| printf("bye\n"); // выход из программы "легальным способом" | |
| free\_stack(root\_of\_stack); // очистка памяти | |
| return 0; | |
| } | |
| else | |
| { | |
| printf("error"); // если подана неверная команда, сообщение об ошибке | |
| free\_stack(root\_of\_stack); // очистка памяти и завершение из программы | |
| return 0; | |
| }; | |
| } while (1); | |
| return 1;// если каким-то образом цикл был завершен, сообщение о серьезной ошибке и ненулевой код возврата | |
| } | |
|  | |
| void push\_cell\_in\_stack(stack \* root\_of\_stack, int number) | |
| { | |
| stack \* created\_cell = (stack\*)malloc(sizeof(stack)); // создание нового элемента | |
| created\_cell->store\_in\_cell = number; // присвоение значения новому элементу | |
| created\_cell->prev\_cell = root\_of\_stack->prev\_cell; // внедрение элемента в начало списка | |
| root\_of\_stack->prev\_cell = created\_cell; | |
| return; | |
| } | |
|  | |
| int pop\_cell\_from\_stack(stack \* root\_of\_stack) | |
| { | |
| stack \* poping\_elem = root\_of\_stack->prev\_cell; | |
| int poping\_number = poping\_elem->store\_in\_cell; // сохранение значения удаляемого элемента | |
| root\_of\_stack->prev\_cell = root\_of\_stack->prev\_cell->prev\_cell; // отвязывание удаляемого элемента | |
| free(poping\_elem); // очистка памяти удаляемого элемента | |
| return poping\_number; // возврат числа хранившегося в удаленном элементе | |
| } | |
|  | |
| int top\_cell\_of\_stack(stack \* root\_of\_stack) | |
| { | |
| return root\_of\_stack->prev\_cell->store\_in\_cell; // можно было реализовать просто указателем | |
| } | |
|  | |
| void free\_stack(stack \* root\_of\_stack) | |
| { | |
| stack \* buffer; // буферный указатель структурного типа | |
| while (root\_of\_stack->prev\_cell != NULL) | |
| { | |
| buffer = root\_of\_stack; // ввиду особенностей одностороннего списка, вынужденно | |
| while (buffer->prev\_cell->prev\_cell != NULL) // доходим до конца списка и очищаем последний элемент | |
| { | |
| buffer = buffer->prev\_cell; | |
| } | |
| free(buffer); | |
| buffer->prev\_cell = NULL; | |
| } |
| } |

**Составление отчёта:**

* По лабораторной работе составлен отчет Lr\_2\_orchet.docx и файл с кодом программы source.c  
  В репозитории на Github создана от мастера ветка Shevchenko\_Danil\_Lab\_2 с помощью команды «git checkout -b Shevchenko\_Danil\_Lab\_2».
* Папка с файлами отчета и кода была помещена в ветку репозитория командами:  
  «git add Lab\_2\_Shevchenko\_Danil\_2\_sem»  
  «git commit»   
  «git push origin Shevchenko\_Danil\_Lab\_2»
* После чего был сделан request на ветку master.

**Вывод**: Были изучены методы работы со стеком, его структурные особенности, методы моделирования работы стека. Так же были изучены методы очистки памяти в одностороннем списке, усовершенствованы навыки обработки строк и в частности строковых команд.