**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Программирование»**

Тема: **Динамические структуры данных**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Коршунов П.И. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Изучение основных механизмов языка С++ путем разработки структур данных стека на основе динамической памяти.

## Задание.

Вариант 2. Стековая машина.

Требуется написать программу, которая последовательно выполняет подаваемые ей на вход арифметические операции над числами с помощью стека на базе списка.

1) Реализовать класс CustomStack, который будет содержать перечисленные ниже методы. Стек должен иметь возможность хранить и работать с типом данных int.

Структура класса узла списка:

struct ListNode {

ListNode\* mNext;

int mData;

};

Объявление класса стека:

class CustomStack {

public:

// методы push, pop, size, empty, top + конструкторы, деструктор

private:

// поля класса, к которым не должно быть доступа извне

protected: // в этом блоке должен быть указатель на голову

ListNode\* mHead;

};

Перечень методов класса стека, которые должны быть реализованы:

void push(int val) - добавляет новый элемент в стек

void pop() - удаляет из стека последний элемент

int top() - доступ к верхнему элементу

size\_t size() - возвращает количество элементов в стеке

bool empty() - проверяет отсутствие элементов в стеке

2) Обеспечить в программе считывание из потока stdin последовательности (не более 100 элементов) из чисел и арифметических операций (+, -, \*, / (деление нацело)) разделенных пробелом, которые программа должна интерпретировать и выполнить по следующим правилам:

Если очередной элемент входной последовательности - число, то положить его в стек,

Если очередной элемент - знак операции, то применить эту операцию над двумя верхними элементами стека, а результат положить обратно в стек (следует считать, что левый операнд выражения лежит в стеке глубже),

Если входная последовательность закончилась, то вывести результат (число в стеке).

Если в процессе вычисления возникает ошибка:

например вызов метода pop или top при пустом стеке (для операции в стеке не хватает аргументов),

по завершении работы программы в стеке более одного элемента,

программа должна вывести "error" и завершиться.

Примечания:

Указатель на голову должен быть protected.

Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено.

Предполагается, что пространство имен std уже доступно.

Использование ключевого слова using также не требуется.

Структуру ListNode реализовывать самому не надо, она уже реализована.

## Выполнение работы

Были реализован класс *CustomStack*, который представляет собой пользовательский стек, основанный на связном списке. *CustomStack()* конструктор по умолчанию, создает новый пустой стек с головой, указывающей на новый узел со значением 0. *CustomStack(ListNode \* head)* конструктор, создающий новый стек с головой, указывающей на указанный узел. *~CustomStack()* деструктор, освобождающий память, выделенную для всех узлов в стеке. *void operation(char oper)* мсетод, выполняющий операцию над двумя верхними элементами стека. Если в стеке менее двух элементов, выводится сообщение об ошибке и программа завершается. В противном случае, из стека извлекаются два верхних элемента, выполняется операция и результат помещается обратно в стек. *void* *push(int val)* метод, добавляющий новый элемент в стек, если стек пуст, голова указывает на новый узел с указанным значением, а противном случае, новый узел добавляется в конец связного списка. *void pop()* метод, удаляющий верхний элемент из стека, если стек пуст, выводится сообщение об ошибке и программа завершается. В противном случае, удаляется последний узел из связного списка. *int top()* метод, возвращающий значение верхнего элемента в стеке, если стек пуст, выводится сообщение об ошибке и программа завершается. *size\_t size()* метод, возвращающий количество элементов в стеке. *bool empty()* метод, проверяющий, пуст ли стек. В *main()* считывается строка с выражением, которое состоит из целых чисел и арифметических операций. Строка разбивается на токены с помощью функции *strtok().* Затем создается новый узел *head* со значением первого токена и создается экземпляр класса *CustomStack*, инициализированный этим узлом. Далее, в цикле происходит обработка остальных токенов. Если токен представляет собой арифметическую операцию, вызывается метод *operation*() для выполнения этой операции над двумя верхними элементами стека. Если токен представляет собой целое число, вызывается метод *push*() для добавления этого числа в стек. После обработки всех токенов, если в стеке остался ровно один элемент, выводится его значение. В противном случае, выводится сообщение об ошибке.

Разработанный программный код см. в приложении А.

## Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 1 -10 - 2 \* | error |  |
|  | 1 -10 - 2 \* | 22 | - |

## Выводы

Были изучены основные механизмы языка С++ путем разработки структур данных стека на основе динамической памяти.

# Приложение А Исходный код программы

Название файла: Korshunov\_Petr\_lb4.c

class CustomStack {

public:

CustomStack(): mHead{new ListNode{nullptr, 0}} {}

CustomStack(ListNode \* head): mHead{head} {}

~CustomStack() {

while(!empty()){

pop();

}

}

void operation(char oper){

if(size() < 2){

cout << "error" << endl;

exit(0);

}

int right = top();

pop();

int left = top();

pop();

if(oper == '+'){ push(left + right); }

else if(oper == '-'){ push(left - right); }

else if(oper == '/'){ push(left / right); }

else if(oper == '\*'){ push(left \* right); }

}

void push(int val){

if(empty()){

mHead = new ListNode{nullptr, val};

return;

}

ListNode \* temp = mHead;

while(temp->mNext != nullptr){

temp = temp->mNext;

}

ListNode \* node = new ListNode{nullptr, val};

temp->mNext = node;

return;

}

void pop(){

if(empty()){

cout << "error" << endl;

exit(0);

}

ListNode \* temp = mHead;

if(temp->mNext == nullptr){

delete temp->mNext;

mHead = nullptr;

return;

}

while(temp->mNext->mNext != nullptr){

temp = temp->mNext;

}

delete temp->mNext->mNext;

temp->mNext = nullptr;

return;

}

int top(){

if(empty()){

cout << "error" << endl;

exit(0);

}

ListNode \* temp = mHead;

while(temp->mNext != nullptr){

temp = temp->mNext;

}

return temp->mData;

}

size\_t size(){

if(empty()){

return 0;

}

size\_t len = 1;

ListNode \* temp = mHead;

while(temp->mNext != nullptr){

len++;

temp = temp->mNext;

}

return len;

}

bool empty(){

return mHead == nullptr;

}

private:

protected:

ListNode\* mHead;

};

int main() {

char s[100];

cin.getline(s, 100);

char \* p;

char \*\* ls = new char\* [100];

p = strtok(s, " ");

int c = 0;

while (p != NULL) {

ls[c++] = p;

p = strtok(NULL, " ");

}

ListNode\* head = new ListNode{nullptr, atoi(ls[0])};

CustomStack stack(head);

for(size\_t i = 1; i < c; i++){

if(strlen(ls[i]) == 1 & (\*ls[i] == '+' | \*ls[i] == '-' | \*ls[i] == '/' | \*ls[i] == '\*')){

stack.operation(\*ls[i]);

}else{

stack.push(atoi(ls[i]));

}

}

if((stack.size() != 1)){

cout << "error" << endl;

}else{

cout << stack.top() << endl;

}

return 0;

}