חלק 2

1.צריך לשנות את ההגדרה של משתנה size ו משתנה data שלא יהיו const מכיוון שמחסנית צריכה להיות מסוגלת באופן דינאמי ולכן המשתנה size ו data יצטרכו להשתנות ולכן לא יכולים להיות const.

int size; //-const : we need to change value of size in ChangeStackSize

char\*\* data; //-const : we need to change value of data in ChangeStackSize

2. הבנאי של המחסנית מקבל משתנה יחיד מסוג int ולכן יש להגדיר אותו explicit אחרת הקומפיילר עלול להמיר משתנים מסוג int ל מחסנית במקומות לא רצויים. לכן יש להשתמש explicit והקוד שישתמש במחסנית יצטרך לציין במפורש מתי הוא רוצה ליצור מחסנית משתנה int על ידי Statck(int).

explicit Stack(int size) : size(size), data(new char\*[size]), currSize(0) {

}

3. בהורס של המחסנית יש לקרוא לפונקציה delete [] מכיוון ש שהקצנו את data עם סוגריים מרובעים(כלומר מערך) ולכן יש למחוק אותו עם האופרטור delete[]

~Stack() {

delete[] data; //-(data)

}

4. בפונקציה get צריך להחזיר את data[currSize-1] ולא data[currSize] מכיוון ש מערכים מתחילים מ0 ולכן האיבר האחרון נמצא במקום אחד לפני מספר האברים במערך.

char\* get() {

return data[currSize - 1];

// we need to return last string in the stack, so this

// element is on currSize-1 place

}

5. בפונקציה changeStackSize יש להגדיר את הparamter stack בתור refrenece ולא בתור משתנה רגיל, משום שהמשתמש בפונקציה מצפה לשנות את המחסנית שהוא מעביר וכרגע הפונקציה תשתמש בבנאי העתקה ותעדכן מחסנית חדשה ואז תהרוס אותה, והמחסנית של המשתמש לא תשתנה.

שימוש בrefrence יגרום לכך שהמחסנית שהפונקציה היא אותה מחסנית של המשתמש.

friend void ChangeStackSize(int newSize, Stack& stack)

6. בפונקציה ChangeStackSize במקרה שבו מקטינים את גודל המחסנית יש לבדוק שלא מעתיקים איברים שחורגים מהגודל החדש של המחסנית, כלומר אם הקטנו את המחסנית צריך למחוק את האיברים החורגים לאחר ההעתקה.

// delete members that are over the new size of the stack

for (int i = stack.currSize; i > newSize; i--) {

stack.pop();

}

for (int i = 0; i < stack.currSize i++) {

data\_temp[i] = stack.data[i];

}

7. בפונקציה ChangeStackSize יש להגדיר את סוג המשתנה data\_temp ל char\*\*. מכיוון שהוא לא מאותחל בנוסף מכיוון שהפונקציה מוגדרת כfriend אין לה גישה לthis ולכן כשניגשים למשתנה data של statck יש לציין את המשתנה.

char\*\* data\_temp = new char\*[newSize];

delete[] stack.data; // stack.data

stack.data = data\_temp; // we change data so, it hasn't be constant

8. לפי חוק "השלוש הגדולים" במידה וממשים את ההורס יש לבטל או לממש מחדש את בנאי העתקה ואופרטור ההשמה, מכיוון שאחרת יהיו שני מחסניות עם גישה לאותו מערך, ובהשמה לא תתבצע מחיקה למערך הישן.

Stack(const Stack& stack)

: size(size), data(new char\*[stack.size]), currSize(0) {

for (int i = 0; i < stack.currSize; i++) {

this->push(stack.data[i]);

}

}

Stack& operator=(Stack& s) = delete;

הקוד המתוקן:

#include <cstdlib>

#include <cstring>

class Stack {

int size; //-const : we need to change value of size in ChangeStackSize

char\*\* data; //-const : we need to change value of data in ChangeStackSize

int currSize;

public:

// needs to be explicit

explicit Stack(int size) : size(size), data(new char\*[size]), currSize(0) {

}

~Stack() {

while (this->currSize) {

this->pop();

}

delete[] data; //-(data)

}

void push(char\* stringToAdd) {

data[currSize++] =

stringToAdd; // maybe we need to copy everything with strcopy

// otherwise the user need to use the strings it

// passes to the stack

}

char\* get() {

return data[currSize - 1];

// we need to return last string in the stack, so this

// element is on currSize-1 place

}

void pop() {

delete data[--currSize]; // we have to use delete instead free

}

void pop(int numOfElemToPop) {

for (int i = 0; i < numOfElemToPop; ++i) {

pop();

}

}

void pop(

double numOfElemToPop) { // we don't need 2 same functions for int

// and double - compiler can change int

// to double automatic

// probably need to delete this function since

// deleting 4.5 elements is meaningless

for (int i = 0; i < numOfElemToPop; ++i) {

pop();

}

}

friend void ChangeStackSize(int newSize, Stack& stack) {

stack.size = newSize;

char\*\* data\_temp = new char\*[newSize]; // +char\*\*: we need to declare

// variable before it's using

// delete members that are over the new size of the stack

for (int i = stack.currSize; i > newSize; i--) {

stack.pop();

}

for (int i = 0; i < stack.currSize i++) {

data\_temp[i] = stack.data[i];

}

delete[] stack.data; // stack.data

stack.data = data\_temp; // we change data so, it hasn't be constant

}

friend bool operator==(

const Stack& stack,

const Stack& stack1) { //+friend //const //because we added friend, we

// need to add stack to compare with.

if (stack1.currSize != stack.currSize) { // stack.currSize

return false;

}

for (int i = 0; i < stack.currSize; i++) {

if (stack1.data[i] != stack.data[i]) {

return false;

}

}

return true;

}

Stack(const Stack& stack)

: size(size), data(new char\*[stack.size]), currSize(0) {

for (int i = 0; i < stack.currSize; i++) {

this->push(stack.data[i]);

}

}

Stack& operator=(Stack& s) = delete;

};