Week 4: Stack ADT, House Keeping Functions, and Templates

CSCI 2100 Data Structures
Fall 2017

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University
Stack

- A **stack** is an **abstract data type (ADT)** that stores data in such a way that the last piece of data stored, is the first one retrieved
  - also called last-in, first-out (LIFO)

- Only access to the stack is the top element
  - consider trays in a cafeteria
    - to get the bottom tray out, you must first remove all of the elements above

- An abstract data type (ADT) is basically a logical description or a specification of components of the data and the operations that are allowed, that is independent of the implementation.
  - E.g., Stack, Queue, List, Set, Map, Graph
Stack Two Main Functions

- **Push**
  - the operation to place a new item at the top of the stack
- **Pop**
  - the operation to remove the next item from the top of the stack
Stack

push(16)

item = pop()
item = 16
Implementing a Stack

- At least two different ways to implement a stack
  - array
  - linked list

- Which method to use depends on the application
  - what advantages and disadvantages does each implementation have?
Implementing Stacks: Array

● Advantages
  ▪ best performance

● Disadvantage
  ▪ fixed size

● Basic implementation
  ▪ initially empty array
  ▪ field to record where the next data gets placed into
  ▪ if array is full, push() returns false
    • otherwise adds it into the correct spot
  ▪ if array is empty, pop() returns null
    • otherwise removes the next item in the stack
int main() {
    int choice, element;
    stack st;

    while(true)
    {
        cout << "1.push 2.pop 3.display 4.exit" << endl;
        cout << "Enter ur choice = ";
        cin >> choice;
        switch(choice)
        {
            case 1:  cout <<"Enter the element = ";
                     cin >> element;
                     st.push(element);
                     break;
            case 2:  st.pop(); break;
            case 3:  st.display(); break;
            case 4:  exit(0);
        }
    }
    return 0;
}
Complete the class by yourself

```cpp
#include<iostream>
using namespace std;

class stack {
    private:
        int stk[5];
        int top;
    public:
        stack() {
            top=-1;
        }
        void push(int x) {
        }
        void pop() {
        }
        void display() {
    }
};
```

- **Key Points:**
  - **push()**
    - If pushed without problem, then print "inserted element = " and the value.
    - If array is full, then print as "Warn: stack over flow" and return to main.
  - **pop()**
    - If popped without problem, then print "deleted element = " and the value.
    - If you cannot pop anymore, then "Warn: stack under flow" and return to main.
  - **display()**
    - For loop of the array if elements exist.
    - If not, then print "Warn: stack empty" and return to main.
Complete the class by yourself

```cpp
#include<iostream>
using namespace std;

class stack {
    private:
        int stk[5];
        int top;

    public:
        stack() {
            top=-1;
        }

        void push(int x) {
        }

        void pop() {
        }

        void display() {
        }
};
```

CAS-MCS-7Q2ZG4:week4 ahnt$ ./stack_array_simple
1.push 2.pop 3.display 4.exit
Enter ur choice = 1
Enter the element = 1
inserted element = 1
1.push 2.pop 3.display 4.exit
Enter ur choice = 1
Enter the element = 2
inserted element = 2
1.push 2.pop 3.display 4.exit
Enter ur choice = 1
Enter the element = 3
inserted element = 3
1.push 2.pop 3.display 4.exit
Enter ur choice = 1
Enter the element = 4
inserted element = 4
1.push 2.pop 3.display 4.exit
Enter ur choice = 1
Enter the element = 5
inserted element = 5
1.push 2.pop 3.display 4.exit
Enter ur choice = 1
Enter the element = 6
Warn: stack over flow
1.push 2.pop 3.display 4.exit
Enter ur choice = 3
1 2 3 4 5
1.push 2.pop 3.display 4.exit
Enter ur choice = 4
CAS-MCS-7Q2ZG4:week4 ahnt$
Exception Handling

- In previous, we just printed out the warning message.
- How to throw an exception?

```cpp
double sqrt(double number) {
    if (number < 0)
        throw domain_error("number is negative");
}
```

- Change the previous example as below:

```cpp
if (empty()) throw runtime_error("Stack is Empty");
```
Git

- Version control repository system
- Project hosting services
  - Github
  - Bitbucket
  - Sourceforge
- Git server on CS@SLU
- Let’s test Git on the hopper.slu.edu
What is a copy constructor?

- The **copy constructor** is a constructor which creates an object by initializing it with an object of the same class, which has been created previously.
  - Initialize one object from another of the same type.
  - Copy an object to pass it as an argument to a function.
  - Copy an object to return it from a function.

- Note:
  - If a copy constructor is not defined in a class, the compiler itself defines one.
  - If the class has pointer variables and has some dynamic memory allocations, then it is a must to have a copy constructor.
Copy constructor format

classname (const classname &obj) {
    // body of constructor
}

```cpp
#include <iostream>

using namespace std;

class Rectangle {
public:
    Rectangle(int a, int b) {
        width = a;
        height = b;
        cout << "Default constructor" << endl;
    }

    Rectangle(const Rectangle &obj) {
        width = obj.width;
        height = obj.width;
        cout << "Copy constructor" << endl;
    }

    ~Rectangle() {
        cout << "Destructor" << endl;
    }

    int area() {
        return (width*height);
    }

private:
    int width, height;
};

int main() {
    Rectangle rect1 (5, 5);
    Rectangle rect2 = rect1;

    cout << "Area of rect1 : " << rect1.area() << endl;
    cout << "Area of rect2 : " << rect2.area() << endl;

    return 0;
}
```
Example class when copy constructor is needed?

```cpp
#include<iostream>
#include<string>
using namespace std;

class String
{
    private:
        char *s;
        int size;
    public:
        String(const char *str = NULL); // constructor
        ~String() { delete [] s; } // destructor
        String(const String&); // copy constructor
        void print() { cout << s << endl; } // Function to print string
        void change(const char *); // Function to change
    private:
        String::String(const char *str)
        {
            size = strlen(str);
            s = new char[size+1];
            strcpy(s, str);
        }

        void String::change(const char *str)
        {
            delete [] s;
            size = strlen(str);
            s = new char[size+1];
            strcpy(s, str);
        }
        String::String(const String& old_str)
        {
            size = old_str.size;
            s = new char[size+1];
            strcpy(s, old_str.s);
        }

        int main()
        {
            String str1("GeeksQuiz");
            String str2 = str1;
            str1.print(); // what is printed ?
            str2.print();
            str2.change("GeeksforGeeks");
            str1.print(); // what is printed now ?
            str2.print();
            return 0;
        }
};
```
Shallow Copy vs Deep Copy

● Shallow Copy:
  ▪ The data members of one object are copied into the data members of another object without taking any dynamic memory pointed to by those data members into consideration. ("memberwise copy")

● Deep Copy:
  ▪ Any dynamic memory pointed to by the data members is duplicated and the contents of that memory is copied (via copy constructors and assignment operators -- when overloaded)
Assignment Operator

- The assignment operator = is used to assign the left-hand operand the value of the right hand operator.
When do I need to write an assignment operator?

- First, you should understand that if you do not declare an assignment operator, the compiler gives you one implicitly.

- The implicit assignment operator does member-wise assignment of each data member from the source object.

- In general, any time you need to write your own custom copy constructor, you also need to write a custom assignment operator.
Recall - Swap: call By Pointer

```c
// function definition to swap the values.
void swap(int *x, int *y) {
    int temp;
    temp = *x;    // save the value at address x
    *x = *y;      // put y into x
    *y = temp;   // put x into y
    return;
}

#include <iostream>
using namespace std;

int main () {
    int a = 100, b=200; // local variable declaration:
    cout << "Before swap, values of a, b :" << a << ", " << b << endl;
    swap(&a, &b); // indicates pointers to a and b (addresses)
    cout << "After swap, values of a, b :" << a << ", " << b << endl;
    return 0;
}
```

```
CAS-MCS-7Q22G4:Tests ahnt$ ./swap_reference
Before swap, values of a, b :100, 200
After swap, values of a, b :200, 100
CAS-MCS-7Q22G4:Tests ahnt$
```
Templates

- **Templates** help in defining *generic classes and functions* and hence allow generic programming.

- Generic programming is an approach where generic data types are used as parameters and the same piece of code work for various data types.

- The format for declaring function templates:

  ```
  template <class identifier> function_declaration;
  template <typename identifier> function_declaration;
  ```

  The only difference between both prototypes is the use of either the keyword class or the keyword typename. Its use is indistinct, since both expressions have exactly the same meaning and behave exactly the same way.
```cpp
#include <iostream>

using namespace std;

// function definition to swap the values.
template<typename T>
void myswap(T &x, T &y) {
    T temp = x;
    x = y;    // put y into x
    y = temp; // put x into y
    return;
}

int main () {
    int a = 100, b=200;    // local variable declaration:
    cout << "Before swap, values of a, b :" << a << ", " << b << endl;
    myswap(a, b);    // indicates pointers to a and b (addresses)
    cout << "After swap, values of a, b :" << a << ", " << b << endl;
    return 0;
}
```
Git Initial setup

- Connect to hopper.slu.edu
- Make a Git directory
  
  $ cd ~/courses/csci2100
  
  $ mkdir Git
  
  $ cd Git

- Let git know your name and email
  
  $ git config --global user.name "Jane Smith"
  
  $ git config --global user.email jane.smith@slu.edu

- Set your preferred editor (you can also use vi, emacs, etc.)
  
  $ git config --global core.editor "vi"
Git with lab4

Git Initial setup

- Create the initial course repository.
  
  $ git clone git@git.cs.slu.edu:courses/fall17/csci2100-01

- Do your work in the directory it creates
  
  $ cd csci2100-01

- Add in your personal course repository
  
  $ git remote add my git@git.cs.slu.edu:courses/fall17/csci2100-01/jsmith
Git with lab4

- Some of you created lab04 branch in the previous class.
- Let’s remove the branch and start again.
  
  ```
  $ git branch
  ```

- If you see the branch name “lab04”, as below
  
  ```
  * lab4
  master
  ```

- Then change the location from lab04 to master with
  
  ```
  $ git checkout master
  ```

- Now, you can delete the branch.
  
  ```
  $ git branch -D lab4
  ```
Git with lab4

- Fetch and check out assignment1 from origin.
  ```
  $ git fetch origin
  $ git checkout -b lab4 origin/lab4
  ```

- Check whether you have the lab4.pdf file in the current directory.
  ```
  $ ls -al *
  ```

- Complete your homework. In this case, copy the file into here.
  ```
  $ cp ~/courses/csci2100/lab4/doubles.cpp
  ```

- Add the file and commit
  ```
  $ git add doubles.cpp
  $ git commit –m “lab4 solution upload”
  ```

- Push work to the repository
  ```
  $ git push my lab4
Stack with Templates

- HW3