Advanced Object-Oriented Programming, Spring 2018

Homework Assignment #1

Due midnight Wednesday, April 4, 2018

Instructions

1. If any question is unclear, please ask for a clarification.
2. You may try to reuse as much of the source code supplemented as possible.
3. Unless stated otherwise, all the line numbers for the program listings are for reference only.
4. You are required to do all the homework assignments on Linux using g++.
5. You are required to give your TA a demo of your program. Make sure that your program can compile and run on the server machine, which will be used for the demo.
6. For the program that you write, you are required to include a Makefile. Otherwise, your homework will not be graded—meaning that you will receive zero marks.
7. Unless stated otherwise, you are required to work on the homework assignment individually.
8. No late homework will be accepted.

Programming Project

The purpose of this homework assignment is to get you acquainted with Linux, Gnu **make**, and GCC (Gnu compiler collection).

This assignment requires that you reimplement the stack programs (Listings 2 and 4) in both C and C++ using singly linked list so that the size of the stack is limited by the available physical memory instead of 100 as specified by the symbolic constant `STACK_SIZE` $= 100$. You are further required to keep the interfaces (Listings 1 and 3) intact—that is, the public functions in C and the public member functions in C++—so that the users of your stack programs do not have to change their programs because the implementation of your stack programs is changed, as we discussed in the classroom.

Listing 1: src/c/stack.h

```
1 #ifndef __STACK_H__
2 #define __STACK_H__
3
4 #define STACK_SIZE 100
```

```
5
6  struct stack {
7      int sp;
8      int stk[STACK_SIZE];
9  };
10
11 extern void push(struct stack* this, int x);
12 extern int pop(struct stack* this);
13 extern struct stack* new_stack();
14 extern void delete_stack(struct stack* stk);
15
16 #endif
```

Listing 2: src/c/stack.c

```
1  #include <stdlib.h>
2  #include "stack.h"
3
4  void push(struct stack* this, int x)
5  {
6      this->stk[++this->sp] = x;
7  }
8
9  int pop(struct stack* this)
10 {
11     return this->stk[this->sp--];
12 }
13
14 struct stack* new_stack()
15 {
16     struct stack* stk = malloc(sizeof(struct stack));
17     stk->sp = -1;
18     return stk;
19 }
20
21 void delete_stack(struct stack* stk)
22 {
23     free(stk);
24 }
```

Listing 3: src/c++/stack.h

```
1  #ifndef __STACK_H__
2  #define __STACK_H__
3
4  class Stack {
5      enum { STACK_SIZE = 100 };
6
7      int stk[STACK_SIZE];
8      int sp;
9  public:
10     Stack()
11     {
12         sp = -1;
13     }
14     void push(int x)
15     {
16         stk[++sp] = x;
17     }
18     int pop()
19     {
20         return stk[sp--];
21     }
```

```
};

#endif
```

Listing 4: src/c++/stack.c

```
#include "stack.h"
```

To simply the homework assignment, you may try to reuse as much of the code provided as you can. Moreover, to make it easier for you to test your programs, also given are the driver programs (Listings 5, 6, 7 and 8) that are going to be used to test to see if your programs work during the demo and that are not allowed to be modified.

Listing 5: c/main.c

```
#include <stdio.h>
#include "stack.h"

const int NUM_ITEMS = 200;

int main()
{
    struct stack* stk1 = new_stack();
    struct stack* stk2 = new_stack();

    int i;

    for (i = 0; i < NUM_ITEMS; i++) {
        push(stk1, 100+i);
        push(stk2, 600+i);
    }

    printf("Dump of stack 1:\n");
    for (i = 0; i < NUM_ITEMS; i++)
        printf("%d\n", pop(stk1));

    printf("Dump of stack 2:\n");
    for (i = 0; i < NUM_ITEMS; i++)
        printf("%d\n", pop(stk2));

    delete_stack(stk1);
    delete_stack(stk2);

    return 0;
}
```

Listing 6: c/main2.c

```
#include <stdio.h>
#include "stack.h"

const int NUM_ITEMS = 200;

int main()
{
    struct stack* stk1 = new_stack();
    struct stack* stk2 = new_stack();

    int i;

    for (i = 0; i < NUM_ITEMS; i++) {
        push(stk1, 200+i);
        push(stk2, 700+i);
```

```
    }

    printf("Dump of stack 1:\n");
    for (i = 0; i < NUM_ITEMS; i++)
        printf("%d\n", pop(stk1));

    printf("Dump of stack 2:\n");
    for (i = 0; i < NUM_ITEMS; i++)
        printf("%d\n", pop(stk2));

    delete_stack(stk1);
    delete_stack(stk2);

    return 0;
}
```

Listing 7: c++/main.c

```
#include <iostream>
#include "stack.h"

using std::cout;
using std::endl;

const int NUM_ITEMS = 200;

int main()
{
    Stack stk1;
    Stack stk2;

    for (int i = 0; i < NUM_ITEMS; i++) {
        stk1.push(100+i);
        stk2.push(600+i);
    }

    cout << "Dump of stack 1:" << endl;
    for (int i = 0; i < NUM_ITEMS; i++)
        cout << stk1.pop() << endl;

    cout << "Dump of stack 2:" << endl;
    for (int i = 0; i < NUM_ITEMS; i++)
        cout << stk2.pop() << endl;

    return 0;
}
```

Listing 8: c++/main2.c

```
#include <iostream>
#include "stack.h"

using std::cout;
using std::endl;

const int NUM_ITEMS = 200;

int main()
{
    Stack stk1;
    Stack stk2;

    for (int i = 0; i < NUM_ITEMS; i++) {
```

```
15          stk1.push(200+i);
16          stk2.push(700+i);
17      }
18
19      cout << "Dump of stack 1:" << endl;
20      for (int i = 0; i < NUM_ITEMS; i++)
21          cout << stk1.pop() << endl;
22
23      cout << "Dump of stack 2:" << endl;
24      for (int i = 0; i < NUM_ITEMS; i++)
25          cout << stk2.pop() << endl;
26
27      return 0;
28  }
```

Note that all of the driver programs are trying to push 200 items (as specified by the symbolic constant `NUM_ITEMS` = 200) into the stack before the pop function is invoked. This will certainly cause the array version to overflow. However, the linked list version you are going to implement should be doing a fine job.

Also, make sure that you clean up all the memory allocated at the end of your programs.

Grading Policy

The grading policy for this assignment is as follows:

- 10 points if a **Makefile**, which contains at least three targets—**all**, **dep**, and **clean**—is provided and would successfully compile your programs. Otherwise, your program will not be graded.
- 80 points if your programs compile without errors and warnings, and the answers are correct.
- 10 points if your programs are well modularized and structured.

Gentle Reminder

1. If you have never had experience on using Linux, start earlier. It may take you quite a while to get acquainted with it.
2. If you have never had Linux installed on your system, it is time to get it installed.