Data Structure Program Assignment #4
(Due: PM: 9:00, March 31, 2022)

Template Bag Class and Inheritance

Instructor: Jiann-Jone Chen

- **Introduction**

The template can be considered as a parameterized type. Template functions in C++ make it easier to reuse classes and functions. In this project, you are asked to use the template to present container classes, such as bag, queue, and stack classes. Inheritance is also used to simplify the program design for container classes. In the lecture class, we showed some template function implementation examples for stack and queue classes, so that a container class can support template functions.

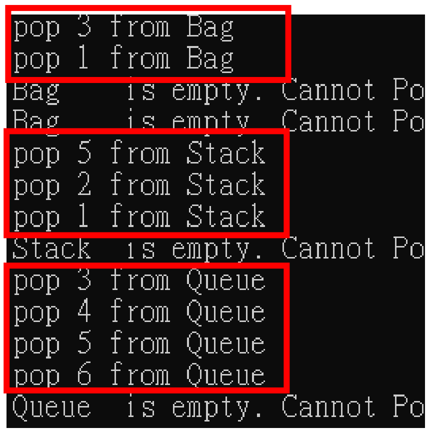
A Bag class can be considered as the base class to design other container classes, e.g., Stack and Queue. The template Bag class is shown below:

- **Steps**

1. Design the base class, Bag Class
   As shown, basic member functions (IsEmpty(), IsFull(), Empty(), Full()) and data members (*array, capacity, top, name[10]) are defined in this base class, so the derived class, Stack and Queue, can reuse them to eliminate the time-consuming redesign process. Since the data are stored and accessed through the pointer, T *array, which is defined in the Bag class. A friend function

```
friend   ostream& operator<<<>(ostream&, const Bag <T>&);
```

   is designed in the Bag class to print out the data stored in the array of a derived class.
   A member function `void message(const T& x);` is declared to deal with the message printing. For example, when you pop( ) a value x=5 from the stack class, the program had to print out pop 5 from Stack. As shown on the right figure.



```
pop 3 from Bag
pop 1 from Bag
Bag     is empty. Cannot Po
Bag     is empty. Cannot Po
pop 5 from Stack
pop 2 from Stack
pop 1 from Stack
Stack   is empty. Cannot Po
pop 3 from Queue
pop 4 from Queue
pop 5 from Queue
pop 6 from Queue
Queue   is empty. Cannot Po
```

```
#include <iostream>
#include <string.h>
#include <iomanip>
using namespace std;

template <class T>
class Bag {
     friend    ostream& operator<<<>(ostream&, const Bag <T>&);
public:
     Bag(int BagSize) :MaxSize(BagSize), name("Bag"), top(0) {
          array = new T[MaxSize];
     };
     ~Bag() {delete[] array; };   // destructor
     virtual int  Size() const;
     virtual bool IsEmpty() const;
     virtual bool IsFull() const;
     virtual void Push(const T&);
     virtual T* Pop(T &);
protected:
     T *array;
     int MaxSize;
     int top;   // position of the top element
     char name[20];
     void Empty() {
      cout <<"   "<<setw(6) << left << name << " is empty. Cannot Pop()\n";  };
     void Full(const T& x) {
      cout <<"   "<< setw(6) << left << name << " is full for " << x << endl; };
     void message(const T& x);
};

template <class T>
ostream& operator<<<>(ostream& os, const Bag <T>& b) {
     os << "   All Data in "<<right<<setw(6)<<b.name<< '['<<b.MaxSize<<"] are: ";
     for (int i = 0; i < b.MaxSize; i++)
          os << "(" << i << ") " << b.array[i] << setw(3) << " ";

     os << "\n   Valid data among them are: ";
     for (int i = 0; i < b.top; i++)
          os << "(" << i << ") " << b.array[i] << setw(3) << " ";
     cout << "\n\n";
     return os;
}
```

2. Design the Derived Class: a Stack Class
   In the Stack class, the Push() function is the same as that in the Bag class, but the Pop() function (in the lecture note, it is the delete() function).   That's why only the Pop() function is re-designed.

```
#include "bag.h"

template <class T>
class Stack: public Bag<T> {

public:
    Stack(int StackCapacity) :Bag(StackCapacity) {
        strcpy(name, "Stack");
    };
    ~Stack() {};
    T* Pop(T &); // delete the element from stack
};
```

3. Design the derived Class: the Queue Class

   In the Queue class, we adopt a circular queue structure to store the data to avoid frequent data shift operations. Both Push() and Pop() functions have to be re-designed to implement the required data operations. In addition, the operator overloading function

   `friend ostream& operator<<<>(ostream&, const Queue <T>&);`

   had to be re-designed to print out the data stored in the circular queue.

```
#include "bag.h"
template <class T>
class Queue:public Bag<T> {
    friend    ostream& operator<<<>(ostream&, const Queue <T>&);

public:
    Queue(int QueueSize):Bag(QueueSize) {
        front = rear = -1;
        strcpy(name, "Queue");
    };
    ~Queue() {};
    T* Pop(T&);           // delete the element from queue
    void Push(const T&);
private:
    int rear, front; // circular list implementation
};
```

4. The main program is shown below.

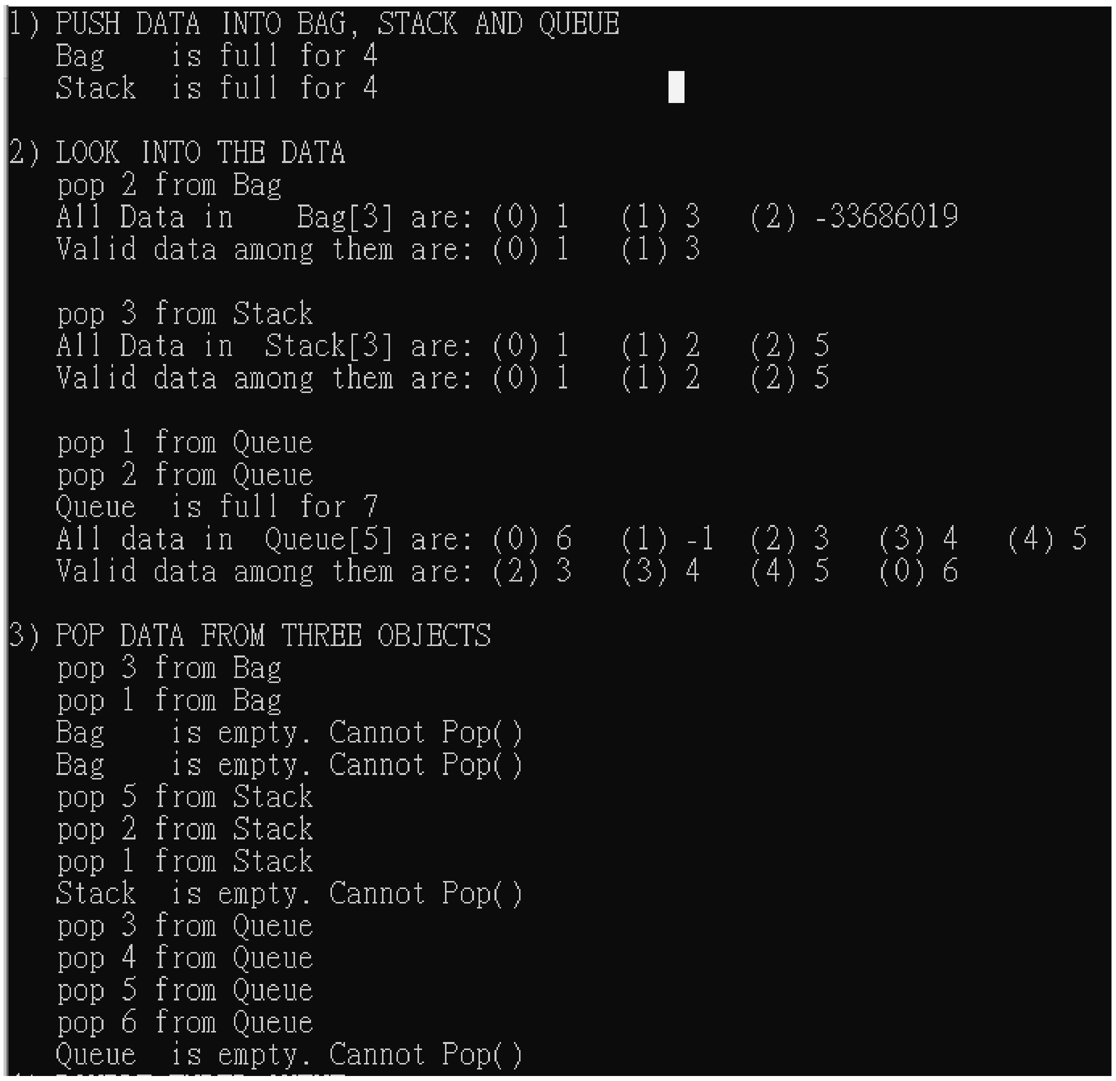
main.cpp

```
#include "stack.h"
#include "bag.h"
#include "queue.h"
int main(){
     Bag<int> b(3);      // uses Bag constructor to create array of size 3
     Stack<int> s(3);    // uses Stack constructor to create array of size 3
     Queue<int> q(5);    // circular queue needs one more pace
     int i=0, x[8]={1,2,3,4,5,6,7,8},t;
     cout << "1) PUSH DATA INTO BAG, STACK AND QUEUE\n";
          while(i < 4) {
               b.Push(x[i]);    // use Bag::Push
               s.Push(x[i]);    // Stack::Push not defined, use Bag::Push
               q.Push(x[i++]);  // Queue::Push override Bag::Push for circular list
          }
          cout << "\n2) LOOK INTO THE DATA\n";
          b.Pop(t); cout << b;
          s.Pop(t); s.Push(x[4]);  cout << s;
          q.Pop(t); q.Pop(t);
          q.Push(x[4]); q.Push(x[5]); q.Push(x[6]);
          cout << q;

          cout << "3) POP DATA FROM THREE OBJECTS\n";
          i = 0;  while (i < 4) b.Pop(x[i++]);
          i = 0;  while (i < 4) s.Pop(x[i++]);
          i = 0;  while (i < 5) q.Pop(x[i++]);

     return 0;
};
```

The execution results are shown below:



```
1) PUSH DATA INTO BAG, STACK AND QUEUE
   Bag    is full for 4
   Stack  is full for 4

2) LOOK INTO THE DATA
   pop 2 from Bag
   All Data in    Bag[3] are: (0) 1   (1) 3   (2) -33686019
   Valid data among them are: (0) 1   (1) 3

   pop 3 from Stack
   All Data in  Stack[3] are: (0) 1   (1) 2   (2) 5
   Valid data among them are: (0) 1   (1) 2   (2) 5

   pop 1 from Queue
   pop 2 from Queue
   Queue  is full for 7
   All data in  Queue[5] are: (0) 6   (1) -1  (2) 3   (3) 4   (4) 5
   Valid data among them are: (2) 3   (3) 4   (4) 5   (0) 6

3) POP DATA FROM THREE OBJECTS
   pop 3 from Bag
   pop 1 from Bag
   Bag    is empty. Cannot Pop()
   Bag    is empty. Cannot Pop()
   pop 5 from Stack
   pop 2 from Stack
   pop 1 from Stack
   Stack  is empty. Cannot Pop()
   pop 3 from Queue
   pop 4 from Queue
   pop 5 from Queue
   pop 6 from Queue
   Queue  is empty. Cannot Pop()
```

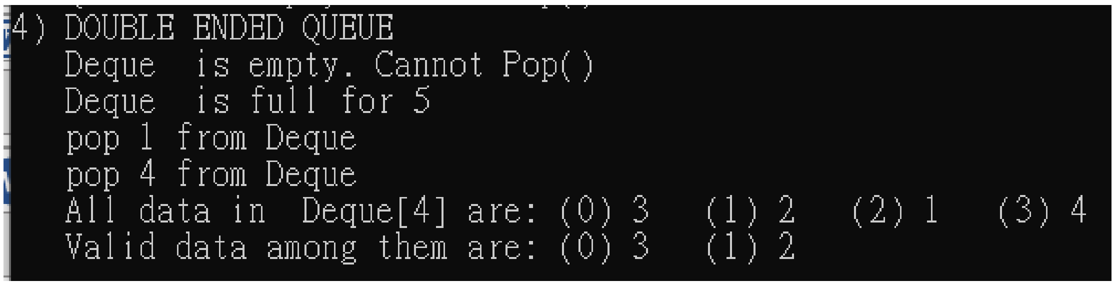
5. A demo project is provided for you to quickly start designing the program. Click the bag.sln to start the programming.

- **Requirement (85%)**
  1. You had to submit the complete project such that the TA can recompile your programs to verify correctness.
  2. Write a short report to describe
     (1) What is all about the program?
     (2) Describe your program by writing notes for each instruction.
     (3) How do you improve this program? List your contributions.
- Bonus: (15%)
  1. In our lecture, a circular queue with capacity n can only utilize n-1 item space instead of n. Can you modify the algorithm to fully

utilize all available space?

2. You are encouraged to modify the queue class to enable push and pop from both sides, i.e., a double-ended queue class (Deque) with Pushf(), Pushr(), Popf(), and Popr() functions, in which Pushf() means Push from the front and Popr() means Pop from the rear. When the following instructions are added, execution results look like in the following back figure.

```
        cout << "4) DOUBLE ENDED QUEUE\n";
    Deque <int> dq(4);
        dq.Popr(t);
        dq.Pushf(3); dq.Pushr(2);
        dq.Pushf(4); dq.Pushr(1);
        dq.Pushf(5);
        dq.Popr(t);  dq.Popf(t);
        cout << dq;
```



```
4) DOUBLE ENDED QUEUE
   Deque  is empty. Cannot Pop()
   Deque  is full  for 5
   pop 1 from Deque
   pop 4 from Deque
   All data in  Deque[4] are: (0) 3   (1) 2   (2) 1   (3) 4
   Valid data among them are: (0) 3   (1) 2
```

- Note: For this homework, you can discuss with other classmates about the program design instead of copying programs. If you finished the project very early, don't share your program with others. Otherwise, the credits will also be shared by students who submit the same program contents.