Lab 11
Implementation and Applications of Stacks and Queues

<u>Section 1: Guess program outputs.</u>

1.
```
#include <iostream>
#include <cstdlib>
#include <string>

using namespace std;

class DynIntStack
{
   struct StackNode
   {
      int value;
      StackNode *next;

      StackNode(int value1, StackNode *next1 = NULL)
      {
         value = value1;
         next = next1;
      }
   };
   StackNode *top;
public:
   DynIntStack() { top = nullptr; }
   ~DynIntStack();

   void push(int);
   void pop(int &);
   bool isEmpty() const;

   // Stack Exception
   class Underflow {};
};

void DynIntStack::push(int num)
{
   top = new StackNode(num, top);
}

void DynIntStack::pop(int &num)
{
   StackNode *temp;

   if (isEmpty()) { throw DynIntStack::Underflow(); }
   else
   {
      // Pop value off top of stack
      num = top->value;
      temp = top;
      top = top->next;
      delete temp;
   }
}
```

```
bool DynIntStack::isEmpty() const
{
   return top == nullptr;
}

DynIntStack::~DynIntStack()
{
   StackNode* garbage = top;
   while (garbage != nullptr)
   {
      top = top->next;

      garbage->next = nullptr;
      delete garbage;
      garbage = top;
   }
}

int main()
{
   DynIntStack stack;
   int popped_value;

   for (int value = 5; value <= 15; value = value + 5)
   {
      cout << "Push: " << value << "\n";
      stack.push(value);
   }
   cout << "\n";

   for (int k = 1; k <= 3; k++)
   {
      cout << "Pop: ";
      stack.pop(popped_value);
      cout << popped_value << endl;
   }

   try
   {
      cout << "\nAttempting to pop again... ";
      stack.pop(popped_value);
   }
   catch (DynIntStack::Underflow)
   {
      cout << "Underflow exception occurred.\n";
   }

   return 0;
}
```

2.

```
#include <iostream>
#include <string>
#include <cstdlib>

using namespace std;

class DynIntQueue
{
   struct QueueNode
   {
      int value;
      QueueNode *next;
      QueueNode(int value1, QueueNode *next1 = nullptr)
      {
         value = value1;
         next = next1;
      }
   };
   QueueNode *front;
   QueueNode *rear;
public:
   DynIntQueue();
   ~DynIntQueue();

   void enqueue(int);
   void dequeue(int &);
   bool isEmpty() const;
   void clear();
};

DynIntQueue::DynIntQueue()
{
   front = nullptr;
   rear = nullptr;
}

DynIntQueue::~DynIntQueue()
{
   QueueNode* garbage = front;
   while (garbage != nullptr)
   {
      front = front->next;
      garbage->next = nullptr;
      delete garbage;
      garbage = front;
   }
}

void DynIntQueue::enqueue(int num)
{
   if (isEmpty())
   {
      front = new QueueNode(num);
      rear = front;
   }
   else
   {
      rear->next = new QueueNode(num);
      rear = rear->next;
   }
```

```
}

void DynIntQueue::dequeue(int& num)
{
   QueueNode* temp = nullptr;
   if (isEmpty())
   {
      cout << "The queue is empty.\n";
      exit(1);
   }
   else
   {
      num = front->value;
      temp = front;
      front = front->next;
      delete temp;
   }
}

bool DynIntQueue::isEmpty() const
{
   if (front == nullptr)
      return true;
   else
      return false;
}

void DynIntQueue::clear()
{
   int value;   // Dummy variable for dequeue

   while (!isEmpty())
      dequeue(value);
}

int main()
{
   DynIntQueue iQueue;

   for (int k = 1; k <= 5; k++)
      iQueue.enqueue(k*k);

   while (!iQueue.isEmpty())
   {
      int value;
      iQueue.dequeue(value);
      cout << value << "  ";
   }

   return 0;
}
```

Section 2: Review Questions and Exercises

1. Suppose the following operations were performed on an empty stack:
push(0);
push(9);
push(12);
push(1);
Insert numbers in the following diagram to show what will be stored in the static stack after the operations have executed.

	top of stack
	bottom of stack

2. Suppose the following operations were performed on an empty stack:
push(8);
push(7);
pop();
push(19);
push(21);
pop();
Insert numbers in the following diagram to show what will be stored in the static stack after the operations have executed.

	top of stack
	bottom of stack

3. Suppose the following operations are performed on an empty queue:
enqueue(5);
enqueue(7);
enqueue(9);
enqueue(12);
Insert numbers in the following diagram to show what will be stored in the static queue after the operations have executed.

front			rear

4. Suppose the following operations are performed on an empty queue:
enqueue(5);
enqueue(7);
dequeue();
enqueue(9);
enqueue(12);
dequeue();
enqueue(10);
Insert numbers in the following diagram to show what will be stored in the static queue after the operations have executed.

front			rear

Section 3: Programming Challenges

1. Dynamic Queue Template
In the class you studied DynIntQueue, a class that implements a dynamic queue of integers.Write a template that will create a dynamic queue of any data type. Demonstrate the class with a driver program.

2. Stack-based Evaluation of Postfix Expressions
Write a program that reads postfix expressions and prints their values. Each input expression should be entered on its own line, and the program should terminate when the user enters a blank line. Assume that there are only binary operators and that the expressions contain no variables. Your program should use a stack. Here are sample input-output pairs:

78	78
78 6 +	84
78 6 + 9 2 - /	12