UNIVERSITI SAINS MALAYSIA

SCHOOL OF COMPUTER SCIENCES
UNIVERSITI SAINS MALAYSIA
Semester II Session 2023/2024

CPT113 – Programming Methodology & Data Structures
Tutorial Week 11
Stacks and Queues

search
3.

4
3
2
1

aux.

Learning Outcomes:

- Understanding the use of static and dynamic stacks and queues

1. Using the stack ADT defined in the lecture notes (DynIntStack), create a member function which is able to search an item inside the stack without violating the preliminary rules of a stack (FILO). Upon finding or not finding the search item print a suitable message. If the item is found remove it from the stack. The order of the stack should remain untouched in case of removing the item.

Students may implement the function differently based on their creativity. However, below is a very simple example of it.

```
void DynIntStack::search(int searchItem)
{
   bool found = false;
   int aux;
   DynIntStack temp;

   while(!isEmpty())
   {
      pop(aux);
      if(aux != searchItem)
      {
         temp.push(aux);
      }
      else
      {
         found = true;
         cout << "Item exist in the stack" << endl;
      }
   }
   if(!found)
      cout << "Item does not exist in the stack" << endl;

   while(temp.top)
   {
      temp.pop(aux);
      push(aux);
   }
}
```

```
void search (int searchItem)

bool found = false
int aux
DynIntstack temp
while (! isEmpty ())
{
    pop (aux);
    if (aux ! = searchItem)
        temp. push (aux)
    else
        found = true
}
if (! found)

while (temp. top )
{
    temp. pop (aux)
    push (aux)
}
```

2. Based on your Stack ADT, convert it into Stack Template.

3. Given the following myStack class:

```
class myStack {
      myStack stackCopy(const myStack);
      const myStack& operator=(const myStack&);
      bool isEmptyStack() const;
      bool isFullStack() const;
      void initializeStack(); //Function to initialize the stack to an
      empty state.
      void push(const int& newItem);
      void pop(const int& item);
      myStack();
      ~myStack();
};
```

Construct a complete method for stackCopy using available methods in class myStack.

```
myStack myStack::stackCopy(const myStack stack){
      int temp ;
      myStack tempStack1, tempStack2;
      while (!stack.isEmptyStack()) {
            stack.pop(temp);
            tempStack1.push(temp);
      }
      while (!tempStack1.isEmptyStack()) {
            tempStack1.pop(temp);
            tempStack2.push(temp);
      }
      return tempStack2;
}
```

4. Describe the two operations that all queues perform.
Enqueue – insert at the rear of queue
Dequeue – remove from the front of queue

dequeue - remove from front
enqueue - enter from back

5. Consider the following statements:

```
queueType<int> queue;
int x, y;
```

Assume queue is empty, show what is output by the following segment of code:

←

~~6~~ 8 6+5 =11 16 6 5

```
x = 6;
y = 8;
~~queue.enqueue(x)~~;
~~queue.enqueue(y~~);
~~queue.dequeue(x)~~;
~~queue.enqueue(x + 5)~~;
~~queue.enqueue(16~~);
~~queue.enqueue(x~~);
~~queue.enqueue(y - 3)~~;
cout << "Queue Elements: ";
while (!queue.isEmpty())
{
queue.dequeue(x);
cout << x << " ";
}
```

Queue elements: 8 11 16 6 5

8 11 16 6 5

6. (a) Construct a header file for a dynamic queue using template to work on any data types.
(b) Demonstrate a main program that creates a dynamic queue that can hold a series of double type values multiplied by the constant, pi. Then dequeue all these values and display them. Use header file in (a).
(c) Demonstrate a main program that creates a dynamic queue that can hold strings, then prompts the user to enter a series of names that are enqueue. Then dequeue all the names and display them. Use header file in (a).

(a)

```
#ifndef DYN_QUE_H
#define DYN_QUE_H
// Dynqueue class template
template <class T>
class Dynque
{
      private:
            struct QueueNode
            {
                  T value;
                  QueueNode *next;
            };
            QueueNode *front;
            QueueNode *rear;
            int numItems;
      public:
            Dynque();
            ~Dynque();
            void enqueue(T);
            void dequeue(T &);
            bool isEmpty();
            bool isFull();
            void clear();
};

//************************************************
// Constructor                                   *
//************************************************

template <class T>
Dynque<T>::Dynque()
{
      front = nullptr;
      rear = nullptr;
      numItems = 0;
}

//************************************************
// Destructor                                    *
//************************************************

template <class T>
Dynque<T>::~Dynque()
{
      clear();
}
```

```
//************************************************
// Function enqueue inserts the value in num     *
// at the rear of the queue.                     *
//************************************************

template <class T>
void Dynque<T>::enqueue(T num)
{
      QueueNode *newNode = nullptr; newNode = new QueueNode;
      newNode->value = num;
      newNode->next = nullptr;
      if (isEmpty())
            front = rear = newNode;
      else
      {
            rear->next = newNode;
            rear = newNode;
      }
      numItems++;
}

//************************************************
// Function dequeue removes the value at the     *
// front of the queue, and copies it into num.   *
//************************************************

template <class T>
void Dynque<T>::dequeue(T &num)
{
      QueueNode *temp = nullptr;
      if (isEmpty())
            cout << "The queue is empty.\n";
      else
      {
            num = front->value;
            temp = front->next;
            delete front;
            front = temp;
            numItems--;
      }
}

//************************************************
// Function isEmpty returns true if the queue    *
// is empty, and false otherwise.                *
//************************************************

template <class T>
bool Dynque<T>::isEmpty()
{
      bool status = true;
      if (numItems)
      {
            status = false;
      }
      return status;
}
```

```
//************************************************
// Function clear dequeues all the elements      *
// in the queue.                                 *
//************************************************

template <class T>
void Dynque<T>::clear()
{
      T value; // Dummy variable for dequeue
      while (!isEmpty())
      {
            dequeue(value);
      }
}

#endif
```

(b)

```
#include<iostream>
#include "Dynqueue.h"
using namespace std;
int main()
{
      // Constant for the number of items
      const int NUM_ITEMS = 5;
      // Create a dynamic queue object.
      Dynque<double> queue;
      cout << "Enqueuing " << NUM_ITEMS << " items...\n";
      // enqueue some items.
      for (int x = 0; x < NUM_ITEMS; x++)
            queue.enqueue(x * 3.14);
      // Deqeue and retrieve all items in the queue
      cout << "The values in the queue were:\n";
      while (!queue.isEmpty())
      {
            double value;
            queue.dequeue(value);
            cout << value << endl;
      }
      return 0;
}
```

(c)

```
#include <iostream>
#include <string>
#include "Dynqueue.h"
using namespace std;
const int QUEUE_SIZE = 5;
int main()
{
      string name;
      // Create a Queue.
      Dynque<string> queue;
      // Enqueue some names.
      for(int count = 0; count < QUEUE_SIZE; count++)
      {
            cout << "Enter an name: ";
            getline(cin, name);
            queue.enqueue(name);
      }
```

```
        // Dequeue the names and display them.
        cout << "\nHere are the names you entered:\n";
        for (int count = 0; count < QUEUE_SIZE; count++)
        {
              queue.dequeue(name);
              cout << name << endl;
        }
        return 0;
}
```

7. Construct a program that opens a text file and reads its contents into a queue of characters. The program should then dequeue each character and substitute it with the character that comes five places after it in the ASCII character set (e.g. a will become f), and store it in a second file. Use the same queue header file from the previous question.

```
#include<iostream>
#include<fstream>
#include<cstdlib>
#include "Dynqueue.h"
using namespace std;
int main()
{
        fstream inFile, outFile;
        Dynque<char> myQ;
        char ch, c, filename[20];
        // Input name of file
        cout << "Enter name of file : ";
        cin >> filename;
        // Open file in input mode
        inFile.open(filename, ios :: in );
        // Test whether file can be opened or not.
        if(inFile.fail())
        {
              cout << "Error in opening source file...." << endl;
        return 0;
        }
        // Read file character by character until end-of-file marker is reached
        while(!inFile.eof())
        {
              inFile.get ( ch );
              myQ.enqueue ( ch );
        }
        inFile.close(); // Close input file
        outFile.open("encrypt.txt", ios::out); // open file in output mode
        // Test whether destination file opens or not
        if(outFile.fail())
        {
              cout << "Error creating destination file....";
        return 0;
        }
        cout << "\nEncrypting contents of " << filename << " in encrypt.txt"
        << endl;
        while ( !myQ.isEmpty())
        {
              myQ.dequeue(ch);
              if ( ch == '\n' )
                    c = ch;
```

```
            else
                c = ch + 5;
            outFile << c;
        }
        // Close output file
        outFile.close();
        return 0;
}

/*
OUTPUT
Enter name of file : test.txt
Encrypting contents of test.txt in encrypt.txt
<.... Contents of test.txt ....>
Hello, this is just a test file.
<.... Contents of encrypt.txt ....>
Mjqqt1%ymnx%nx%ozxy%f%yjxy%knqj33
*/
```

8. Construct a program that opens two text files and reads its contents into two separate queues. Determine whether the files are identical using these queues. Display appropriate messages.

```
#include<iostream>
#include<fstream>
#include<cstdlib>
#include "Dynqueue.h"
using namespace std;
int main()
{
        // Create an input file stream object.
        fstream file1("file1.txt", ios::in);
        // Create an output file stream object.
        fstream file2("file2.txt", ios::in);
        // Create two queues to hold characters.
        Dynque<char> queue1;
        Dynque<char> queue2;
        char ch1, ch2;
        // Read all the characters from file #1 and enqueue them in queue1.
        file1.get(ch1);
        while (!file1.eof())
        {
                queue1.enqueue(ch1);
                file1.get(ch1);
        }
        // Read all the characters from file #2 and enqueue them in queue2.
        file2.get(ch2);
        while (!file2.eof())
        {
                queue2.enqueue(ch2);
                file2.get(ch2);
        }
        // Close the files.
        file1.close();
        file2.close();
        // Dequeue the characters from each queue
        // one-at-a-time and compare them.
        bool status = true;
        while (!queue1.isEmpty() && !queue2.isEmpty())
        {
                queue1.dequeue(ch1);
```

```
            queue2.dequeue(ch2);
            if (ch1 != ch2)
            {
                  status = false;
            }
      }
      if (status)
      {
            cout << "The files are identical.\n";
      }
      else
      {
            cout << "The files are not identical.\n";
      }
      return 0;
}


/*
OUTPUT
<.... Contents of file1.txt ....>
This is a test file.
<.... Contents of file2.txt ....>
This is a test file.
Output: The files are identical.
Otherwise: The files are not identical.
*/
```