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

CPT113 – Programming Methodology & Data Structure
Tutorial Week 9
Templates and Linked Lists

Learning Outcomes:

- Understanding the use of templates and linked lists in class

---

1. The following function accepts an `int` argument and returns half of its value as a `double`. Write a template that will implement this function to accept an argument of any type.

```
double half (int number)
{
      return number/2.0;
}
```

2. Write a function template named arrange that accepts 3 reference variables as arguments and arrange them in ascending order. Write the main function to demonstrate the template with int, double and char data types.

3. Suppose your program uses class template named `List`. Give an example of how you would use `int` as the data type in the definition of a `List` object. (Assume class a default constructor).

```
template<class T>
class List
{
      // members are declared here…
};
```

4. As the following `Rectangle` class is written, all data members are `double`. Rewrite the class as a template that will accept any data type for these members. Secondly, what if the getWidth function definition is to be defined outside the class?

```
class Rectangle
{
      private:
            double width,length;
      public:
      void setData(double w, double l)
            {width = w; length = l;}
      double getWidth()
            {return width;}
      double getLength()
            {return length;}
      double getArea()
```

```
            {return width*length;}
};
```

What if the getWidth function definition is to be defined outside the class?

5. What are the advantages of linked list over arrays?

6. List five basic linked list operations.

7. What is the difference between appending a node and inserting a node?

8. Design your own linked list class to hold a list of integers. The class should have member functions for appending, inserting, deleting and print nodes, as well as constructor and destructor. In the main, write a program to showcase all functionalities of your linked list class.

9. Modify your linked list class by adding a member function `reverse` that rearranges the nodes in the list so that their order is reversed. Add codes to the main to test the new function.

10. Modify your list class by adding a member function for inserting a new item at a specified position. At position 0 means that the value will become the first item on the list. A position equal or greater that the length of the list means the value is placed at the end of list.

11. What is the advantage of using a template to implement a linked list?

12. Given the following LinkedList header file:
    a. Define a class PersonInfo that has member `name, IC, gender` and `age`, and appropriate accessor and mutator methods. Include method `findAge` using `IC`.
    b. Inside main() function, create `person` as an object of type `PersonInfo.` Then, create a `list` as an object of type `LinkedList` that stores nodes of type `PersonInfo`.
    c. Using a menu driven program, implement all the methods in class `LinkedList` on object `person` using methods in class `PersonInfo.`

```
#ifndef LINKEDLIST_H
#define LINKEDLIST_H
#include<iostream>
#include<string>
using namespace std;
//*****************************************
```

```
// The ListNode class creates a type used to *
// store a node of the linked list. *
//*****************************************
template <class T>
class ListNode
{
      public:
            T value; // Node value T of class personInfo - composition
            ListNode<T> *next; // Pointer to the next node
            //Constructor
            ListNode (T nodeValue)
            {
                  value = nodeValue;
                  next = NULL;
            }
};

//*******************
// LinkedList class *
//*******************
template <class T>
class LinkedList
{
      private:
            ListNode<T> *head;
      public:
            // Constructor
            LinkedList()
            { head = NULL; }
            // Destructor
            ~LinkedList();
            // Linked list operations
            void appendNode(T); //to add node
            void insertNode(T); //to insert node
            void deleteNode(T); //to delete node
            void searchNode(T); //to delete node
            void displayList() const; //to display all the nodes
};


//************************************************
// appendNode appends a node containing the value *
// passed into newValue, to the end of the list. *
//************************************************
template <class T>
void LinkedList<T>::appendNode(T newValue)
{
      ListNode<T> *newNode; // To point to a new node
      ListNode<T> *nodePtr; // To move through the list
      // Allocate a new node and store newValue there.
      newNode = new ListNode<T>(newValue);
      // If there are no nodes in the list,make newNode the first node.
      if (!head)
            head = newNode;
      else // Otherwise, insert newNode at end.
      {
            // Initialize nodePtr to head of list.
```

```
            nodePtr = head;
            // Find the last node in the list.
            while (nodePtr->next)
            nodePtr = nodePtr->next;
            // Insert newNode as the last node.
            nodePtr->next = newNode;
      }
}

//************************************************
// displayList shows the value stored in each node *
// of the linked list pointed to by head. *
//************************************************
template <class T>
void LinkedList<T>::displayList() const
{
      ListNode<T> *nodePtr;
      // To move through the list
      // Position nodePtr at the head of the list.
      nodePtr = head;
      // While nodePtr points to a node, traverse the list.
      while (nodePtr)
      {
            // Display the value in this node.
            nodePtr->value.displayPerson();
            cout<< endl;
            // Move to the next node.
            nodePtr = nodePtr->next;
      }
}


//************************************************
// The insertNode function inserts a node with *
// newValue copied to its value member. *
//************************************************
template <class T>
void LinkedList<T>::insertNode(T newValue)
{
      ListNode<T> *newNode;
      ListNode<T> *nodePtr;
      ListNode<T> *previousNode = nullptr;
      // Allocate a new node and store newValue there.
      newNode = new ListNode<T>(newValue);
      // If there are no nodes in the list
      // make newNode the first node
      if (!head)
      {
            head = newNode;
            newNode->next = NULL;
      }
      else // Otherwise, insert newNode
      {
            // Position nodePtr at the head of list.
            nodePtr = head;
            // Initialize previousNode to nullptr.
            previousNode = nullptr;
```

```
            // Skip all nodes whose value is less than newValue.
            while (nodePtr != NULL && nodePtr->value.getIC()
< newValue.getIC() ) // method in PersonInfo
            {
                  nodePtr = nodePtr->next;
            }
            // If the new node is to be the 1st in the list,
            // insert it before all other nodes.
            if (previousNode == NULL)
            {
                  head = newNode;
                  newNode->next = nodePtr;
            }
            else // Otherwise insert after the previous node.
            {
                  previousNode->next = newNode;
                  newNode->next = nodePtr;
            }
      }
}

//**************************************************
// The deleteNode function searches for a node *
// with searchValue as its value. The node, if found, *
// is deleted from the list and from memory. *
//**************************************************

template <class T>
void LinkedList<T>::deleteNode(T searchValue)
{
      ListNode<T> *nodePtr; // To traverse the list
      ListNode<T> *previousNode; // To point to the previous node
      // If the list is empty, do nothing.
      if (!head)
            cout <<"List is empty\n";
      // Determine if the first node is the one.
      if (head->value.getIC() == searchValue.getIC())
      {
            nodePtr=head;
            head=head->next;
            delete nodePtr;
      }
      else
      {
            // Initialize nodePtr to head of list
            nodePtr = head;
            // Skip all nodes whose value member is not equal to num.
            while (nodePtr != NULL && nodePtr->value.getIC()
            != searchValue.getIC())
            {
                  previousNode = nodePtr;
                  nodePtr = nodePtr->next;
            }
            // If nodePtr is not at the end of the list, link the prev node
            // to the node after nodePtr, then delete nodePtr.
            if (nodePtr)
            {
```

```
                previousNode->next = nodePtr->next;
                delete nodePtr;
            }
      }
}


template <class T>
void LinkedList<T>::searchNode(T searchValue)
{
      ListNode<T> *nodePtr; // To traverse
      ListNode<T> *previousNode; // To point to prev node
      // If the list is empty, do nothing.
      if (!head)
            cout <<"List is Empty\n";
      // Determine if the first node is the one.
      if (head->value.getIC() == searchValue.getIC())
      {
            cout<<"\n Found at first node \n";
            head->value.displayPerson(); system("pause");
      }
      else
      {
            // Initialize nodePtr to head of list
            nodePtr = head;
            // Skip all nodes whose value member is not equal to IC.
            while (nodePtr != NULL && nodePtr->value.getIC()
            != searchValue.getIC())
            {
                  previousNode = nodePtr;
                  nodePtr = nodePtr->next;
            }
            // If nodePtr is not at the end of the list,
            // link the previous node to the node after
            // nodePtr, then delete nodePtr.
            if (nodePtr)
            {
                  cout<<"\n Found \n";
                  nodePtr->value.displayPerson();
                  system("pause");
            }
            else
            {
                  cout<<"\nNot found\n"; system("pause");
            }
      }
}


//**************************************************
// Destructor *
// This function deletes every node in the list. *
//**************************************************
template <class T>
LinkedList<T>::~LinkedList()
{
      ListNode<T> *nodePtr; // To traverse the list
```

```
      ListNode<T> *nextNode; // To point to the next node
      // Position nodePtr at the head of the list.
      nodePtr = head;
      // While nodePtr is not at the end of the list...
      while (nodePtr != nullptr)
      {
            // Save a pointer to the next node.
            nextNode = nodePtr->next;
            // Delete the current node.
            delete nodePtr;
            // Position nodePtr at the next node.
            nodePtr = nextNode;
      }
}
#endif
```