SCHOOL OF COMPUTER SCIENCES
UNIVERSITI SAINS MALAYSIA
Semester II Session 2021/2022

CPT113 – Programming Methodology & Data Structures
Tutorial Week 9
Templates and Linked List

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;
}
```

```
template<class T>
double half(T number)
      return number /2.0 ;
```

```
template <class T>
double half(T 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.

```
// Program to write a function template arrange that accepts three
// reference variables as arguments and arranges them in ascending
//  order.
#include<iostream>
using namespace std;
template < class T >
void arrange ( T &a, T &b, T&c)
{
      T small, medium, large;
      if ( (a < b) && ( b < c ))
            return;
      if ( (a < c) && ( c < b ))
      {
            small = a;
            medium = c;
            large = b;
      }
      else if (( b < c ) && ( c < a ))
      {
            small = b;
            medium = c;
```

```
template <class T>
void arrange(T &a, T &b, T &c)

   T  small, medium, large;
   if((a>b)&&(a>c))
         large = a;
         if (b >c)
               medium = b
               small = c
         else
               medium = c
               small = b
   else if ((b>a)&&(b>c))
         large = b
         if (a>c)
               medium = a
               small = c
         else
               medium = c
               small = a
```

```
            large = a;
        }
        else if (( b < a ) && ( a < c ))
        {
            small = b;
            medium = a;
            large = c;
        }
        else if (( c < b ) && ( b < a ))
        {
            small = c;
            medium = b;
            big = a;
        }
        else
        {
            small = c;
            medium = a;
            big = b;
        }
        a = small;
        b = medium;
        c = big;
        return;
    }
```

```
else
    large = c
    if (a >b)
        small = b
        medium = a
    else
        small = a
        medium =c

Cout<< "Arranged order : "<< a <<
"  " << b << "  "<< c <<end;
```

```
int main()
cout<< "Before : 5 , 3 . 2 "<< end;
arrange (5 ,3 ,2)
cout << " Before: 0·1 , 0·3, 1·0 "<<end;
arrange (0·1 , 0·3 ,1·0)
cout << "Before : a , c , d "<<endl;
arrange ( 'a', 'c', 'd')
return 0;
```

```
int main ()
{
int a = 8, b = 19, c = 10 ;
double x = 23.888, y = -99.9876, z = 0.657;
char ch1 = 'A', ch2 = 'M', ch3 = 'Z';
cout << "Integers before arranging : "
<< a << " , " << b << " , " << c << endl;
arrange ( a, b, c );
cout << "Integers after arranging : "
<< a << " , " << b << " , " << c << endl;
cout << "Floating point numbers before arranging : "
<< x << " , " << y << " , " << z << endl;
arrange ( x, y, z );
cout << "Floating point numbers after arranging : "
<< x << " , " << y << " , " << z << endl;
cout << "Characters before arranging : "
<< ch1 << " , " << ch2 << " , " << ch3 << endl;
arrange ( ch1, ch2, ch3 );
cout << "Characters after arranging : "
<< ch1 << " , " << ch2 << " , " << ch3 << endl;
return 0;
}
/*
OUTPUT
Integers before arranging : 8 , 19 , 10
Integers after arranging : 8 , 10 , 19
Floating point numbers before arranging : 23.888 , -99.9876 , 0.657
Floating point numbers after arranging : -99.9876 , 0.657 , 23.888
Characters before arranging : A , M , Z
Characters after arranging : A , M , Z
*/
```

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…
};
```

```
List <int> list;
```

```
List<int> Obj;  //can elaborate with many members in class List.
```

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;}
};
```

```
template <class T>
class Rectangle
      private:
         T width, length;
      public:
         void setData (T w, T l )
          {                         }
          T getwidth ()
              {       }
            T getLength()
              {       }
             T getArea()
            {          }
```

```
template <class T>
class Rectangle
{
      private:
            T width,length;
      public:
            void setData(T w, T l)
            {width = w; length = l;}
            T getWidth()
            {return width;}
            T getLength()
            {return length;}
            T getArea()
            {return width*length;}
};
```

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

```
T getWidth();
template <class T>
     T List<T>::getWidth()
     { return width; }
```

T getwidth ();

template<class T >
  T List<T> :: getwidth()

5. What are the advantages of linked list over arrays?
   Dynamic memory allocation, can insert and delete data easily.

   Dynamic memory allocation

6. List five basic linked list operations.
   append, traverse, insert, delete, destroy list

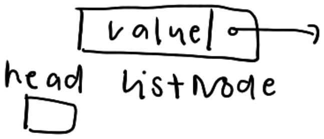
   append, insert, traverse, delete, destroy

7. What is the difference between appending a node and inserting a node?
   Appending is insert at end of list, while insert can be anywhere inside the list.

   append at the end list , insert can be anywhere

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.

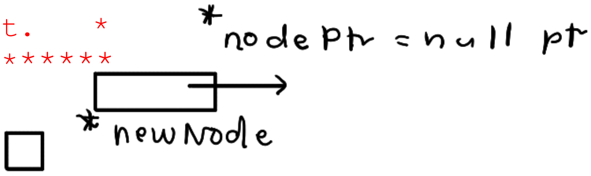
```
// Specification file for the IntList class
#ifndef INTLIST_H
#define INTLIST_H
class IntList
{
     private:
          // Declare a structure for the list
          struct ListNode
          {
               int value;
               struct ListNode *next;
          };
          ListNode *head; // List head pointer
     public:
          IntList() // Constructor
          { head = nullptr; }
          ~IntList(); // Destructor
          void appendNode(int);
          void insertNode(int);
          void deleteNode(int);
          void displayList();
};
#endif
```



```
// Implementation file for the IntList class
#include <iostream>
```

```cpp
#include "IntList.h"
using namespace std;
//**************************************************
// appendNode appends a node containing the        *
// value pased into num, to the end of the list.   *
//**************************************************
void IntList::appendNode(int num)
{
      ListNode *newNode, *nodePtr = nullptr;
      // Allocate a new node & store num
      newNode = new ListNode;
      newNode->value = num;
      newNode->next = nullptr;
      // 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;
      }

}
```



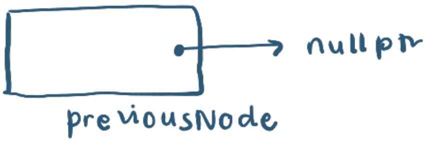
newNode = new ListNode
newNode → value = num
→ next = nullptr
if C! head)
head = newNode
else
nodePtr = head
while (nodePtr → next)
nodePtr = node Ptr → nex
nodePtr → next = newNode.

1. NewNode , nodePtr
2. newNode = new ListNode
3. assign value and next
4. check kalau head xda, jd head
5. kalau x, letak nodePtr kat head and traverse
6. nodePtr → next = newNode.

```cpp
//**************************************************
// displayList shows the value                     *
// stored in each node of the linked list          *
// pointed to by head.                             *
//**************************************************
void IntList::displayList()
{
      ListNode *nodePtr = nullptr;
      nodePtr = head;
      while (nodePtr)
      {
            cout << nodePtr->value << endl;
            nodePtr = nodePtr->next;
      }
}

//**************************************************
// The insertNode function inserts a node with     *
// num copied to its value member.                   *
//**************************************************

void IntList::insertNode(int num)
{
      ListNode *newNode, *nodePtr, *previousNode = nullptr;
      // Allocate a new node & store num
      newNode = new ListNode;
      newNode->value = num;
      // If there are no nodes in the list make newNode the first node
```



① newNode ,nodePtr, prevNode

② allocate new Node

③ assign value New Node = num

④ if head x̄ da. head = new Node
new Node → next = null

⑤ else , pointer letak di head
prev node = Nullptr

⑥ while (nodePtr != null & → value < num)

```
    if (!head)
    {
        head = newNode;
        newNode->next = nullptr;
    }
    else // Otherwise, insert newNode
    {
        // Initialize nodePtr to head of list and previousNode to a
        // null pointer.
        nodePtr = head;
        previousNode = nullptr;
        // Skip all nodes whose value member is less than num.
        while (nodePtr != nullptr && nodePtr->value < num)
        {
            previousNode = nodePtr;
            nodePtr = nodePtr->next;
        }
        // If the new node is to be the 1st in the list, insert it
        // before all other nodes.
        if (previousNode == nullptr)
        {
            head = newNode;
            newNode->next = nodePtr;
        }
        else // Otherwise, insert it after the prev node
        {
            previousNode->next = newNode;
            newNode->next = nodePtr;
        }
    }
}

//**************************************************
// The deleteNode function searches for a node *
// with num as its value. The node, if found, is *
// deleted from the list and from memory. *
//**************************************************
void IntList::deleteNode(int num)
{
    ListNode *nodePtr, *previousNode = nullptr;
    // If the list is empty, do nothing.
    if (!head)
        return;
    // Determine if the first node is the one.
    if (head->value == num)
    {
        nodePtr = head->next;
        delete head;
        head = nodePtr;
    }
    else
    {
        // Initialize nodePtr to head of list
        nodePtr = head;
        // Skip all nodes whose value member is not equal to num.
        while (nodePtr != nullptr && nodePtr->value != num)
        {
            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)
        {
            previousNode->next = nodePtr->next;
            delete nodePtr;
        }
    }
}

//***************************************************
// Destructor                                       *
// This function deletes every node in the list.    *
//***************************************************
IntList::~IntList()
{
    ListNode *nodePtr, *nextNode = nullptr;
    nodePtr = head;
    while (nodePtr != nullptr)
    {
        nextNode = nodePtr->next;
        delete nodePtr;
        nodePtr = nextNode;
    }
}

// Chapter 18, Programming Challenge 1: Your Own Linked List
#include <iostream>
#include "IntList.h"
using namespace std;
int main()
{
    // Create an instance of IntList
    IntList list;
    // Build the list
    list.appendNode(2); // Append 2 to the list
    list.appendNode(4); // Append 4 to the list
    list.appendNode(6); // Append 6 to the list
    // Display the nodes.
    cout << "Here are the initial values:\n";
    list.displayList();
    cout << endl;
    // Insert the value 5 into the list.
    cout << "Now inserting the value 5.\n";
    list.insertNode(5);
    // Display the nodes now.
    cout << "Here are the nodes now.\n";
    list.displayList();
    cout << endl;
    // Delete the node holding 6.
    cout << "Now deleting the last node.\n";
    list.deleteNode(6);
    // Display the nodes.
    cout << "Here are the nodes left.\n";
```

```
	list.displayList();
	return 0;
}
```

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.

```
// Specification file for the IntList class
#ifndef INTLIST_H
#define INTLIST_H
class IntList
{
	private:
		// Declare a structure for the list
		struct ListNode
		{
			int value;
			struct ListNode *next;
		};
		ListNode *head; // List head pointer Destroy function
		void destroy();
	public:
		// Constructor
		IntList()
		{ head = nullptr; }
		// Copy constructor
		IntList(const IntList &);
		// Destructor
		~IntList();
		// List operations
		void appendNode(int);
		void insertNode(int);
		void deleteNode(int);
		void displayList();
		void reverse();
};
#endif

// Implementation file for the IntList class
#include <iostream> // For cout and NULL
#include "IntList.h"
using namespace std;
//**************************************************
// Copy constructor *
//**************************************************
IntList::IntList(const IntList &listObj)
{
	ListNode *nodePtr = nullptr;
	// Initialize the head pointer.
	head = nullptr;
	// Point to the other object's head.
	nodePtr = listObj.head;
	// Traverse the other object.
	while (nodePtr)
```

```
      {
           // Append a copy of the other object's node to this list.
           appendNode(nodePtr->value);
           // Go to the next node in the other object.
           nodePtr = nodePtr->next;
      }
}



//**************************************************
// appendNode appends a node containing the         *
// value pased into num, to the end of the list.    *
//**************************************************
void IntList::appendNode(int num)
{
      ListNode *newNode, *nodePtr = nullptr;
      // Allocate a new node & store num
      newNode = new ListNode;
      newNode->value = num;
      newNode->next = nullptr;
      // 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;
      }
}

//**************************************************
// The print function displays the value             *
// stored in each node of the linked list            *
// pointed to by head.                                 *
//**************************************************
void IntList:: displayList()
{
      ListNode *nodePtr = nullptr;
      nodePtr = head;
      while (nodePtr)
      {
           cout << nodePtr->value << endl;
           nodePtr = nodePtr->next;
      }
}

//**************************************************
// The insertNode function inserts a node with       *
// num copied to its value member.                    *
//**************************************************
void IntList::insertNode(int num)
```

```
{
      ListNode *newNode, *nodePtr, *previousNode = nullptr;
      // Allocate a new node & store num
      newNode = new ListNode;
      newNode->value = num;
      // If there are no nodes in the list make newNode the first node
      if (!head)
      {
            head = newNode;
            newNode->next = nullptr;
      }
      else // Otherwise, insert newNode
      {
            // Initialize nodePtr to head of list and previousNode to a
            // null pointer.
            nodePtr = head;
            previousNode = nullptr;
            // Skip all nodes whose value member is less than num.
            while (nodePtr != nullptr && nodePtr->value < num)
            {
                  previousNode = nodePtr;
                  nodePtr = nodePtr->next;
            }
            // If the new node is to be the 1st in the list, insert it
            // before all other nodes.
            if (previousNode == nullptr)
            {
                  head = newNode;
                  newNode->next = nodePtr;
            }
            else // Otherwise, insert it after the prev node
            {
                  previousNode->next = newNode;
                  newNode->next = nodePtr;
            }
      }
}

//**************************************************
// The deleteNode function searches for a node        *
// with num as its value. The node, if found, is     *
// deleted from the list and from memory.           *
//**************************************************
void IntList::deleteNode(int num)
{
      ListNode *nodePtr, *previousNode = nullptr;
      // If the list is empty, do nothing.
      if (!head)
            return;
      // Determine if the first node is the one.
      if (head->value == num)
      {
            nodePtr = head->next;
            delete head;
            head = nodePtr;
      }
      else
```

```
    {
        // Initialize nodePtr to head of list
        nodePtr = head;
        // Skip all nodes whose value member is not equal to num.
        while (nodePtr != nullptr && nodePtr->value != num)
        {
            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)
        {
            previousNode->next = nodePtr->next;
            delete nodePtr;
        }
    }
}
```

newHead
nodePtr
newNode
tempPtr .

```
void IntList::reverse()
{
    ListNode *newHead = nullptr,
    *newNode,
    *nodePtr,
    *tempPtr;
    // Traverse the list, building a copy of it in reverse order.
    nodePtr = head;
    while (nodePtr)
    {
        // Allocate a new node & store the value of the current list
        //  node in it.
        newNode = new ListNode;
        newNode->value = nodePtr->value;
        newNode->next = nullptr;
        // Shift the existing nodes in the new list down one,
        //  inserting the new node at the top.
        if (newHead != nullptr)
        {
            tempPtr = newHead;
            newHead = newNode;
            newNode->next = tempPtr;
        }
        else
        {
            newHead = newNode;
        }
        // Go to the next node in the list.
        nodePtr = nodePtr->next;
    }
    head = newHead;
    // Destroy the existing list.
    destroy();
    // Make head point to the new list.
    head = newHead;
}
```

```
//***************************************************
// The destroy function destroys all the nodes      *
// in the list.                                        *
//***************************************************
void IntList::destroy()
{
      ListNode *nodePtr, *nextNode = nullptr;
      nodePtr = head;
      while (nodePtr != nullptr)
      {
            nextNode = nodePtr->next;
            delete nodePtr;
            nodePtr = nextNode;
      }
      head = nullptr;
}


//***************************************************
// Destructor                                          *
// This function deletes every node in the list.    *
//***************************************************
IntList::~IntList()
{
      destroy();
}

// Chapter 18, Programming Challenge 4: List Reverse
#include <iostream>
#include "IntList.h"
using namespace std;
int main()
{
      // Create an instance of IntList.
      IntList list;
      // Build a list.
      list.appendNode(2); // Append 2 to the list
      list.appendNode(4); // Append 4 to the list
      list.appendNode(6); // Append 6 to the list
      // Display the nodes.
      cout << "Here are the values in myList:\n";
      list.displayList();
      cout << endl;
      // Reverse the list
      cout << "Reversing the list...\n";
      list.reverse();
      // Display the nodes again
      cout << "Here are the reversed values in myList:\n";
      list.displayL displayList();
      cout << endl;
      return 0;
}
```

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.

```
// Specification file for the IntList class
#ifndef INTLIST_H
#define INTLIST_H
class IntList
{
     private:
          // Declare a structure for the list
          struct ListNode
          {
               int value;
               struct ListNode *next;
          };
          ListNode *head; // List head pointer
          // Destroy function
          void destroy();
     public:
          // Constructor
          IntList()
               { head = nullptr; }
          // Copy constructor
          IntList(const IntList &);
          // Destructor
          ~IntList();
          // List operations
          void appendNode(int);
          void insertNode(int);
          void deleteNode(int);
          void displayList();
          void reverse();
          void insert(int, int);
};
#endif

// Implementation file for the IntList class
#include <iostream>
#include "IntList.h"
using namespace std;
//**************************************************
// Copy constructor                                *
//**************************************************
IntList::IntList(const IntList &listObj)
{
     ListNode *nodePtr = nullptr;
     // Initialize the head pointer.
     head = nullptr;
     // Point to the other object's head.
     nodePtr = listObj.head;
     // Traverse the other object.
     while (nodePtr)
     {
          // Append a copy of the other bject's node to this list.
          appendNode(nodePtr->value);
```

```
                  // Go to the next node in the other object.
                  nodePtr = nodePtr->next;
            }
}

//**************************************************
// appendNode appends a node containing the         *
// value pased into num, to the end of the list.    *
//**************************************************
void IntList::appendNode(int num)
{
      ListNode *newNode, *nodePtr = nullptr;
      // Allocate a new node & store num
      newNode = new ListNode;
      newNode->value = num;
      newNode->next = nullptr;
      // 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;
      }
}

//**************************************************
// The print function displays the value            *
// stored in each node of the linked list           *
// pointed to by head.                               *
//**************************************************
void IntList::displayList()
{
      ListNode *nodePtr = nullptr;
      nodePtr = head;
      while (nodePtr)
      {
            cout << nodePtr->value << endl;
            nodePtr = nodePtr->next;
      }
}

//**************************************************
// The insertNode function inserts a node with      *
// num copied to its value member.                  *
//**************************************************
void IntList::insertNode(int num)
{
      ListNode *newNode, *nodePtr, *previousNode = nullptr;
      // Allocate a new node & store num
```

```
    newNode = new ListNode;
    newNode->value = num;
    // If there are no nodes in the list make newNode the first node
    if (!head)
    {
        head = newNode;
        newNode->next = nullptr;
    }
    else // Otherwise, insert newNode
    {
        // Initialize nodePtr to head of list and previousNode to a
        // null pointer.
        nodePtr = head;
        previousNode = nullptr;
        // Skip all nodes whose value member is less than num.
        while (nodePtr != nullptr && nodePtr->value < num)
        {
            previousNode = nodePtr;
            nodePtr = nodePtr->next;
        }
        // If the new node is to be the 1st in the list, insert it
        // before all other nodes.
        if (previousNode == nullptr)
        {
            head = newNode;
            newNode->next = nodePtr;
        }
        else // Otherwise, insert it after the prev node
        {
            previousNode->next = newNode;
            newNode->next = nodePtr;
        }
    }
}


//**************************************************
// The deleteNode function searches for a node     *
// with num as its value. The node, if found, is   *
// deleted from the list and from memory.          *
//**************************************************
void IntList::deleteNode(int num)
{
    ListNode *nodePtr, *previousNode = nullptr;
    // If the list is empty, do nothing.
    if (!head)
        return;
    // Determine if the first node is the one.
    if (head->value == num)
    {
        nodePtr = head->next;
        delete head;
        head = nodePtr;
    }
    else
```

```
      {
            // Initialize nodePtr to head of list
            nodePtr = head;
            // Skip all nodes whose value member is not equal to num.
            while (nodePtr != nullptr && nodePtr->value != num)
            {
                  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)
            {
                  previousNode->next = nodePtr->next;
                  delete nodePtr;
            }
      }
}

void IntList::reverse()
{
      ListNode *newHead = nullptr,
      *newNode,
      *nodePtr,
      *tempPtr;
      // Traverse the list, building a copy of it in reverse order.
      nodePtr = head;
      while (nodePtr)
      {
            // Allocate a new node & store the value of the current list
            // node in it.
            newNode = new ListNode;
            newNode->value = nodePtr->value;
            newNode->next = nullptr;
            // Shift the existing nodes in the new list down one,
            // inserting the new node at the top.
            if (newHead != nullptr)
            {
                  tempPtr = newHead;
                  newHead = newNode;
                  newNode->next = tempPtr;
            }
            else
            {
                  newHead = newNode;
            } // Go to the next node in the list.
            nodePtr = nodePtr->next;
      }
      // Destroy the existing list.
      destroy();
      // Make head point to the new list.
      head = newHead;
}
```

```
//**************************************************
// The destroy function destroys all the nodes      *
// in the list.                                      *
//**************************************************
void IntList::destroy()
{
      ListNode *nodePtr, *nextNode = nullptr;
      nodePtr = head;
      while (nodePtr != nullptr)
      {
            nextNode = nodePtr->next;
            delete nodePtr;
            nodePtr = nextNode;
      }
      head = nullptr;
}


//**************************************************
// Destructor                                       *
// This function deletes every node in the list.     *
//**************************************************
IntList::~IntList()
{
      destroy();
}


//*********************************************
// LinkedList::insert                          *
// Insert item at given position               *
//*********************************************
void IntList::insert(int value, int pos)
{
      // Allocate a new node & store the value of the current list node in
      // it.
      ListNode *newNode = new ListNode;
      newNode->value = value;
      newNode->next = nullptr;
      if (head == nullptr)
      {
            head = newNode;
            return;
      }
      if (pos == 0)
      {
            newNode->next = head;
            head = newNode;
            return;
      }
      ListNode *p = head; // Walk through nodes
      int numberToSkip = 1; //
      while (numberToSkip <= pos)
      {
            if (p->next == nullptr || numberToSkip == pos)
```

```
        {
            ListNode *tempPtr = p->next;
            p->next = newNode;
            newNode->next = tempPtr;
            return;
        }
        // Not at end and have not skipped enough So skip another one
        p = p->next;
        numberToSkip++;
    }
}
```

```
// Chapter 18, Programming Challenge 6: Member Insertion by Position
#include <iostream>
#include "IntList.h"
using namespace std;
int main()
{
    // Create an instance of IntList.
    IntList myList;
    // Build a list.
    myList.appendNode(2); // Append 2 to the list
    myList.appendNode(4); // Append 4 to the list
    myList.appendNode(6); // Append 6 to the list
    // Display the nodes.
    cout << "Here are the values in myList:\n";
    myList.displayList();
    // Insert a node at position 0.
    myList.insert(99, 0);
    cout << "Inserting a value into myList:\n";
    myList.displayList();
    // Insert a node at position 99.
    myList.insert(100, 99);
    cout << "Inserting a value into myList:\n";
    myList.displayList();
    // Insert a node at position 2.
    myList.insert(98, 2);
    cout << "Inserting a value into myList:\n";
    myList.displayList();
    return 0;
}
```

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

    To store linked lists of any data types.

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 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
```

(a)

```
     #ifndef PERSONINFO_H
     #define PERSONINFO_H
     #include<string>
     #include<cstdlib>
     #include <iostream>
     using namespace std;
```

```
//*****************
//PersonInfoclass *
//*****************

class PersonInfo
{
    private:
        string Name, IC, Gender;
        int Age;
    public:
        void setData(string n, string ic, string g)
            { Name=n; IC = ic; Gender = g; }

        void displayPerson()
            { cout<<"\n\t"<<IC<<" "<<Name<<" "<<Gender<<"
"<<Age<<endl;}

        string getName()
            { return Name; }

        string getIC()
            { return IC; }

        string getGender()
            { return Gender; }

        int getAge()
            { return Age;}

        void findAge()
            {  string tempYear;

                int tempAge;
                tempYear = IC.substr(0,2);
                tempAge = atoi(tempYear.c_str());
                if (tempAge < 20) // assume age <= 100
                    Age = 2023 - (tempAge+2000);
                else
                    Age = 2023 - (tempAge+1900);
            }
      // Constructor
      PersonInfo() {}
};
#endif
```

(b)

```
    PersonInfo person; // person as object type
    LinkedList<PersonInfo> list; // list as object type class linked list
```

(c)

```
#include<iostream>
#include<string>
#include<cstdlib>
#include "LinkedList.h"
#include "PersonInfo.h"
using namespace std;

void menu()
{
    system("clr");
    cout<<"\t *** WELCOME ***\n";
    cout<<"\t 1. Append a person information\n";
    cout<<"\t 2. Delete a person information\n";
    cout<<"\t 3. Insert a person information\n";
    cout<<"\t 4. Search a person based on IC\n";
    cout<<"\t 5. Display persons information\n";
    cout<<"\t 6. Exit\n";
    cout<<"\t Please enter your choice (1-6)\n";
}

int main()
{

    PersonInfo person; // person as object type
    LinkedList<PersonInfo> list; // list as object type class linked list
    string name, ic, gender; // local variables in main int choice;
    int choice;

    do{
        menu();
        cin>>choice;
        switch(choice) {
            case 1: //append a new person
                cout<< "Append a new person name, ic & gender :";
                cin>>name>>ic>>gender;
                person.setData(name, ic, gender); //set the person values
                person.findAge();
                list.appendNode(person); //append the person in linked list
                break;
        case 2: //delete a person by IC first occurance only
            cout<<"\t\n Delete a person name by IC \n";
            cin>>ic;
            person.setData(person.getName(), ic, person.getGender());
            list.deleteNode(person);
            break;
        case 3: //insert a new person
            cout<< "\t\n Insert a new person (name/ic/gender) \n";
            cin>>name>>ic>>gender;
            person.setData(name, ic, gender);
            list.insertNode(person);
            break;
        case 4: //search *** try to return
            cout<<"\t\n Search a person by IC: ";
```

```
                cin>>ic;
                person.setData(person.getName(), ic, person.getGender());
                person.findAge();
                list.searchNode(person);
                break;
            case 5: //display
                list.displayList(); //display the content of linked list
                system("pause");
                break;
            case 6: break;
            default: cout<< " Wrong choice \n"; }
        }while (choice!=6);

        cout<<"\n End of Program";
        return 0;
    }
```