# Tutorial for Week 2 Session 2: Linked List

## Contents

1. Tutorial a: Struct data type and pointer to struct
2. Tutorial b: Self-Referential structures
3. Tutorial c: A Simple Linked list
4. Tutorial d: Using Linked list ADT

## Tutorial w2s2a: Struct

### Learning outcomes:

1. create a struct data type
2. use struct data type

### Tasks:

1. Create a new File in Dev-C++ (File, New) or CTRL-N
2. Solve the following problem:
   1. Create a data type Student using struct with three members: integer to store id, string to store name and double to store gpa.
   2. Enter the data for the id, name and gpa from keyboard input
   3. Print out the contents of the structure
   4. Do the same for a Student pointer variable
3. You can start with the following code

|  |  |
| --- | --- |
| Line 1:  Line 2:  Line 3:  Line 4:  Line 5:  Line 6:  Line 7:  Line 8:  Line 9:  Line 10:  Line 11:  Line 12:  Line 13:  Line 14:  Line 15:  Line 16:  Line 17:  Line 18:  Line 19:  Line 20: | #include <iostream>  #include <string>  using namespace std;  struct Student {  int id;  string name;  double gpa;  };  int main(){  Student s;  Student\* sptr;    int sid;  string sname;  double sgpa;  // your code here |
|  |  |

1. Complete the code.
2. Compile the program, run it.

### Questions:

1. What struct keyword means and how it is useful?
2. When you declare a Student pointer variable, you will need to call new keyword to create the storage for the struct. Why does it so? Why it didn't happen when you declare the Student variable (not a pointer)?

### Summary:

In this tutorial, you have learned how to create user-defined data type using struct keyword. You also created a struct variable and pointer to struct. As opposed to Arrays which only can contain elements of the same data type, struct can contain elements of different data types. Thus, struct is useful in representing a group of related data, for example Student (id, name gpa) or Address (street number, street name, city, region, province, phone number, postal code).

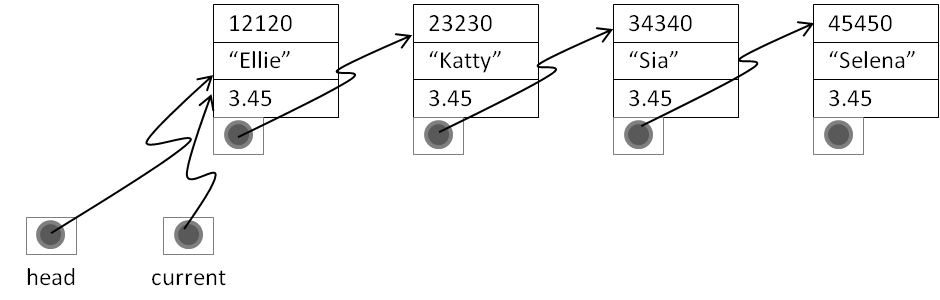
## Tutorial w2s2b: Self-referential structures

### Learning outcomes:

1. Create a self-referential structure
2. Traversal of (access) the element (Node) of the self-referential structure

### Tasks:

1. Create a new File in Dev-C++ (File, New) or CTRL-N
2. Solve the following problem:



* 1. Add a pointer to Student as the member of the Student struct
  2. Create a 4-node linked list manually
  3. Print all the nodes in the linked list manually
  4. Create another 4-node linked list automatically (using while loop)
  5. Print all the nodes in the linked list automatically (using while loop)

1. You can start with the following code

|  |  |
| --- | --- |
| Line 1:  Line 2:  Line 3:  Line 4:  Line 5:  Line 6:  Line 7:  Line 8:  Line 9:  Line 10:  Line 11:  Line 12:  Line 13:  Line 14:  Line 15:  Line 16:  Line 17:  Line 18:  Line 19:  Line 20:  Line 21:  Line 22:  Line 23:  Line 24:  Line 25:  Line 26:  Line 27:  Line 28:  Line 29  Line 30:  Line 31:  Line 32:  Line 33:  Line 34:  Line 35: | #include <iostream>  #include <string>  using namespace std;  struct Student {  int id;  string name;  double gpa;  Student\* link;  };  int main(){  Student\* head;  Student\* current;  Student\* newnode;  int sid;  string sname;  double sgpa;    cout << "Enter student id: ";  cin >> sid;  cout << "Enter student name: ";  cin >> sname;  cout << "Enter student gpa: ";  cin >> sgpa;  head = new Student;  current = head;  current->id = sid;  current->name = sname;  current->gpa = sgpa;  current->link = NULL;  // Your code here |

1. Complete the code.
2. Compile the program, run it.

### Questions:

1. Why there must be a pointer to the first node?
2. What current = current.link means?
3. Why should you have two pointers, which are head and current?
4. What the difference between Linked list and Arrays?
5. What does a traversal mean?
6. What the difference between random access and sequential access?
7. Were the memory for the nodes allocated contiguously?
8. How does the size of the linked list grow?
9. What information that needs to be stored to keep track of a linked list?

### Summary:

In this tutorial, you have learned how to create a self referential structure. A Linked list differs from an array in many aspects, including the memory location, the size and the access. While the storage of an array is allocated in a contiguous block of memory, the storage for a linked list is allocated in random memory location. That's why the structure need a pointer to link between nodes. The size of an array is fixed during compilation time whereas a linked list can grow at our need. An element in an array can be accessed randomly using [] operator, but the node in a linked list must be accessed sequentially.

## Tutorial w2s2c: A Simple Linked list

### Learning outcomes:

1. Use template for the node
2. Create a simple linked list class
3. Build (Insert a node into) the linked list
4. Print the nodes of the linked list
5. Search the linked list
6. Destroy (de-allocate the memory for) the linked list

### Tasks:

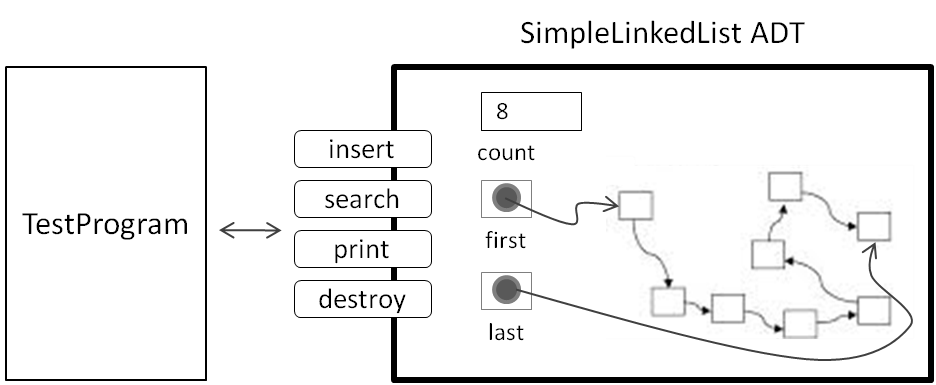
1. Create a new file (CTRL-N).
2. Solve the following problem:
3. Create a struct NODE template with two members: T data type named data and a link to the node.
4. Create public methods: insert,
5. Print all the nodes in the linked list manually
6. Create another 4-node linked list automatically (using while loop)
7. Print all the nodes in the linked list automatically (using while loop)
8. You can start with the following code

|  |  |
| --- | --- |
| Line 1:  Line 2:  Line 3:  Line 4:  Line 5:  Line 6:  Line 7:  Line 8:  Line 9:  Line 10:  Line 11:  Line 12:  Line 13:  Line 14:  Line 15:  Line 16:  Line 17:  Line 18:  Line 19:  Line 20:  Line 21:  Line 22:  Line 23:  Line 24:  Line 25:  Line 26:  Line 27: | #include <iostream>  #include <string>  using namespace std;  struct NODE {  int data;  NODE<int>\* link;  };  class SimpleLL {  public:  void insert(const int& item);  void print() const;  bool search(const int& item) const;  void destroy();  SimpleLL();  ~SimpleLL();    private:  int count;  NODE<int>\* first;  NODE<int>\* last;  };  // Your code here |

1. Complete the code and save the source file as "SimpleLL.h"
2. Create a Test Program which uses the SimpleLinkedList. You can start with the following code:

|  |  |
| --- | --- |
| Line 1:  Line 2:  Line 3:  Line 4:  Line 5:  Line 6:  Line 7:  Line 8:  Line 9:  Line 10:  Line 11:  Line 12:  Line 13:  Line 14:  Line 15:  Line 16: | #include <iostream>  #include <string>  #include "SimpleLinkedList.h"  using namespace std;  int main(){  SimpleLL list;  int num;    cout << "Enter a number. If you type -999, it will stop building the list." << endl;  cin >> name;  // Your code here |

1. Compile the source code (F9), then run it.
2. You can just created a simple abstract data type (ADT) named SimpleLL. Notice that the two test program show that you can use the same ADT for storing different data type. You implemented your ADT using linked list physical data structure. The diagram below illustrate how your ADT is used by the Application Program (Test program). It can be seen that the application program communicate with the ADT through the public interface (methods) of the ADT.



### Questions:

1. Why do you need to put const after the search and print methods?
2. What does & operator means in the parameter of search and insert methods?
3. What is ADT?
4. Why is ADT useful?
5. Why does an ADT use template?

### Summary:

In this tutorial, you have learned how to create and use a simple linked list ADT.

## Tutorial w2s2d (Optional): Using Linked list ADT

### Learning outcomes:

1. Understand the virtual function
2. Use the Singly LinkedList ADT
3. Create a simple Test Program for the Singly Linked List ADT

### Tasks:

1. Copy the Singly Linked List source code files: linkedList.h, unorderedLinkedList.h orderedLinkedList.h, testProgLinkedList.cpp
2. Create a new Project (File, New, Project).
3. Add the linkedList.h, unorderedLinkedList.h orderedLinkedList.h, testProgLinkedList.cpp into the project.
4. Build the Project (F12)
5. Run the Test Program.

### Questions:

1. What is virtual function?
2. Why do we need a virtual function?
3. What happens if you don't override virtual function?
4. What is iterator?
5. Why do we need an iterator?

### Summary:

In this tutorial, you have learned how use virtual functions and iterator.

---end of Tutorial Week 2---