Session 3: C++ Introduction (cont.)

Data Structures and Algorithm 1 - Lab

Yahya Tawil
8 Oct 2021

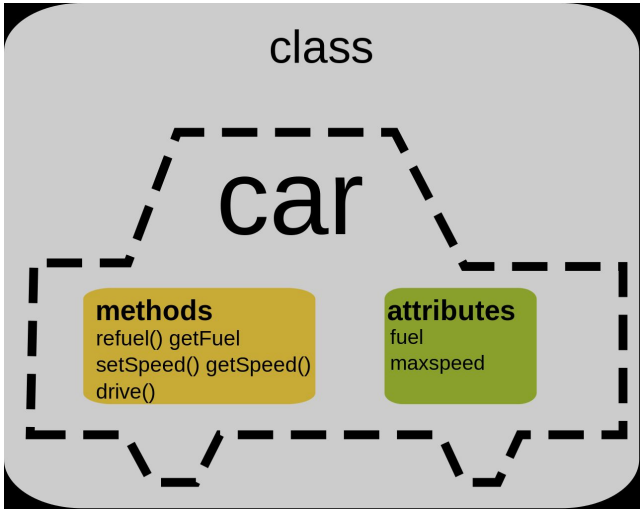
Last Session

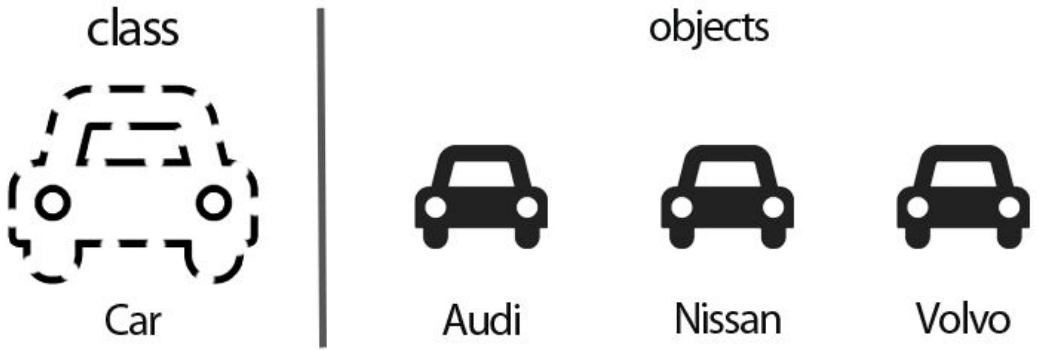
- The main references for our revision of C++ is:
  The C++ Language, Libraries, Tools, and Other Topics
  http://www.ece.uvic.ca/~mdadams/cppbook .

- Last session we reviewed the following: History of C++ - Hello World in C++ - Identifiers and variables types - Inclusion - ASCII representation - functions declaration - Arrays - Pointers - Auto - Typedef - Operations and precedence - Control Flow - Overriding - namespace.

- Today we will continue revision more advanced concepts in C++ and introduce linked list data structures.

C++ Basics (Class)





C++ Basics (Class)

- Class members access level:
  - **Public**: Can be accessed freely.
  - **Private**: Can be accessed by members only.
- members accessed by member-selection operator (.i.e `p.x`)

```
class point {
public:
double x; // The x component
double y; // The y component};
void func() {
point p;
p.x = 1.0; // Set data member x
to 1.0
p.y = 2.0; // Set data member y
to 2.0 }
```

C++ Basics (Class)

- to refer to member of class outside of class body must use scope-resolution operator (i.e., `point::initialize`).

```
class point {
public:
void initialize(double newX, double newY);
double x; // The x component
double y; // The y component
};

void point::initialize(double newX, double
newY) {
x = newX; // "x" means "this->x"
y = newY; // "y" means "this->y"
}
```

C++ Basics (Class)

- Sometimes we need to use `this` explicitly.
- `this` is pointer to object for which member function is being invoked. That is why we used member-selection operator `->` instead of `.`

```
#include <iostream>
class point {
public:
void initialize(double newX, double newY);
double x; // The x component
double y; // The y component
};
void point::initialize(double x, double newY) {
this->x = x;
y = newY; // "y" means "this->y"}
using namespace std;
int main(){
    point p1;
    p1.initialize(1,2);
    cout<<p1.x<<endl;  return 0;}
```

C++ Basics (Class)

- member function whose definition is provided in body of class is implicitly `inline` .

```
class MyInteger {
public:
    // Set the value of the integer and return the old value.
    int setValue(int newValue) {
        int oldValue = value;
        value = newValue;
        return oldValue;
    }
private:
    int value;
};
```

```
class MyInteger {
public:
    // Set the value of the integer and return the old value.
    int setValue(int newValue);
private:
    int value;
};

inline int MyInteger::setValue(int newValue) {
    int oldValue = value;
    value = newValue;
    return oldValue;
}
```

C++ Basics (Class)

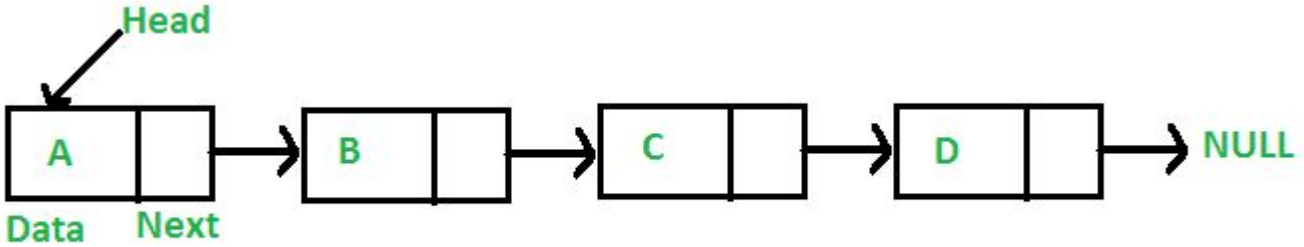
- constructor is member function that is called automatically when object created in order to initialize its value.
- Constructor has same name as class.
- constructor has no return type.
- constructor cannot be called directly.
- constructor can be overloaded

```
class point {
public:
point(double v1,double v2)
{// constructor 1
    x= v1;
    y=v2;
}
point()
{// constructor 2
    x= 1;
    y= 1;
}
double x; double y; };
```

```
using namespace std;
int main(){
    point p1;
    point p2(2,2);
cout<<p1.x<<endl<<p2.x<<
endl;
    return 0;
}
```

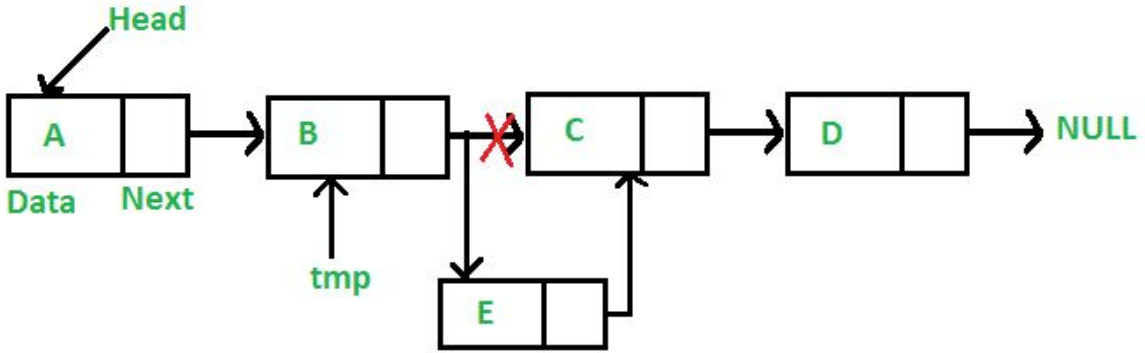
Single Linked List in C++

- The linked list stores data in sequential storage data structure, like arrays.
- The memory locations in the linked list are not contiguous.
- The linked list can store data of different data types.
- Operations: **Insert node** and **delete node**.
- **Example Pros:** insert an element in array it needs lot of shifting. But in linked list its very easy.

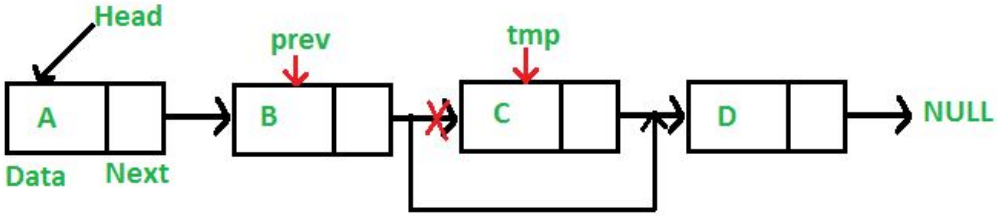


Single Linked List in C++

Insertion:



Deletion:



Single Linked List in C++

```
class Node {
public:
    int data;
    Node* next;

    Node()
    {
        data = 0;
        next = NULL;
    }

    Node(int data)
    {
        this->data = data;
        this->next = NULL;
    }
};
```

Single Linked List in C++

```
class Linkedlist {
    Node* head;

public:
    // Default constructor
    Linkedlist() { head = NULL; }

    // Insert a node at the end of the linked list.
    void insertNode(int);

   // print the linked list.
    void printList();

    // delete node at given position
    void deleteNode(int);
};
```

Single Linked List in C++

```
void Linkedlist::insertNode(int data)
{
    // Create the new Node.
    Node* newNode = new Node(data);

    // Assign to head if the linked list is empty
    if (head == NULL) {
        head = newNode;
        return;
    }

    // Traverse till end of list
    Node* temp = head;
    while (temp->next != NULL) {

        // Update temp
        temp = temp->next;
    }

    // Insert at the last.
    temp->next = newNode;
}
```

Single Linked List in C++

```
void Linkedlist::deleteNode(int nodeOffset)
{
    Node *temp1 = head, *temp2 = NULL;
    int ListLen = 0;

    if (head == NULL) {
        cout << "List empty." << endl;
        return;
    }

    // Find length of the linked-list.
    while (temp1 != NULL) {
        temp1 = temp1->next;
        ListLen++;
    }
    // Check if the position to be
    // deleted is less than the length
    // of the linked list.
    if (ListLen < nodeOffset) {
        cout << "Index out of range"
             << endl;
        return;
    }

    // Declare temp1
    temp1 = head;

    // Deleting the head.
    if (nodeOffset == 1) {

        // Update head
        head = head->next;
        delete temp1;
        return;
    }

    // Traverse the list to
    // find the node to be deleted.
    while (nodeOffset-- > 1) {

        // Update temp2
        temp2 = temp1;

        // Update temp1
        temp1 = temp1->next;
    }

    // Change the next pointer
    // of the previous node.
    temp2->next = temp1->next;

    // Delete the node
    delete temp1;
}
```

Single Linked List in C++

```
void Linkedlist::printList()
{
    Node* temp = head;

    // Check for empty list.
    if (head == NULL) {
        cout << "List empty" << endl;
        return;
    }

    // Traverse the list.
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
}
```

Assignment 2: Exercise 1

Starting from the code provided for linked list in this session, add the following methods:

- `isEmpty` Check if the list is empty.
- `Reset` clear the list.
- `FindAndReplace(int Val,int newVal)` iterate over the linked list and replace any node that has data with value `Val` to `newVal` .