Preview

- Structured Data Type: Linked List
  - Functions for Linked List
    - Search a element from a List
    - Insert a New Node in a List
    - Delete a Node from a List
- Separating Interface from Implementation
  - makefile and makefile Utilities

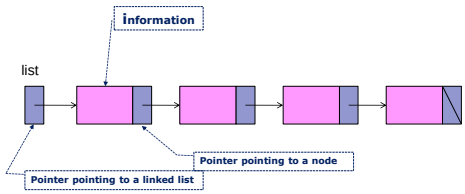
Structured Data Type

- Why we need learn data structure?
  - Any types of software need deal with data.
  - Input data from anywhere (from a file, keyboard,…) must saved in a temporary space.
  - Software engineer need decide which data structure is the best one to deal with specific data set.
    Ex) Array, Linked List, Queue, Stack, Priority Queue, Binary Search Tree, Balanced Binary Search Tree, Heap, Hash Table,…
  - Main operation in any data structures are
    - Search a data in the data structure
    - Insert new data in the data structure
    - Delete a data from the data structure

Structured Data Type: Linked List

- A **linked list** is one of the fundamental data structures, and can be used to implement other data structures (i.e. stack queue..).
- It consists of a sequence of nodes.
- Each node contains arbitrary data fields and one or more pointers (single or double linked list) which can point to the next and/or previous nodes.

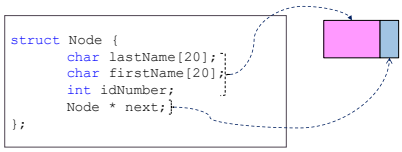
Structured Data Type Linked List



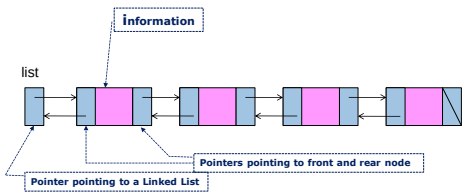
Single linked list

Structured Data Type Linked List

A Node Structure for a Single Linked List



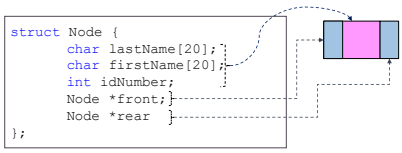
Structured Data Type Linked List



Double Linked List

Structured Data Type Linked List

Node Structure for Double Linked List



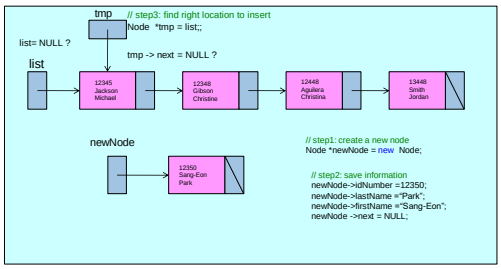
Functions for a Linked List

- Search data from the list
  - There is a data in the list
  - Data not found in the list
- Insert new data in the list
  - Make Sorted list without duplication
  - Insert a new node at the end of the list
- Delete data from the list
  1. Search the data in the list
  2. If there is, delete from the list

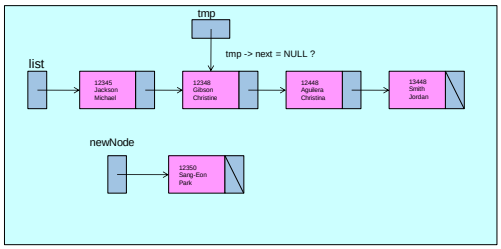
Insert a New Node in a List

- Steps for inserting a new data in the list.
  1. Create a new node
  2. Save information in the new node
  3. Find right location in the list
  4. Insert the new node in the list
- Two possible linked list
  - New node is always insert as the first or last node
  - Insert node in sorted order by one of value in a node (no duplicated ID used for sorting)

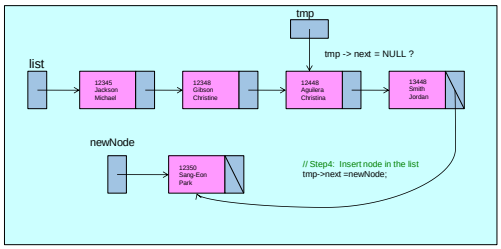
Insert a New Node in a List
(At the End of List)



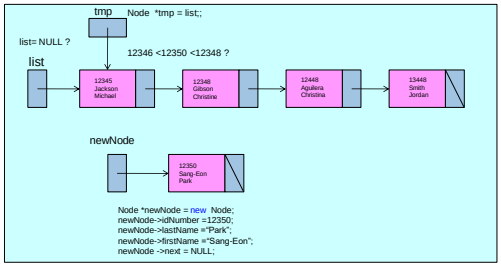
Insert a New Node in a List
(At the End of List)



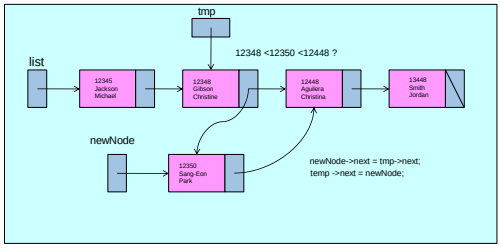
Insert a New Node in a List
(At the End of List)



Insert a New Node in a List
(Sorted by Student ID)



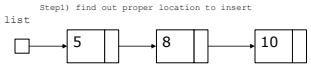
Insert a New Node in a List
(Sorted by Student ID)

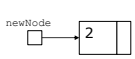


Insert a node
(The First)

```
struct Node {
        int ID;
        Node * next;
};
```

Step1) find out proper location to insert





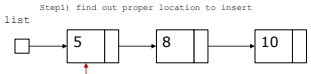
To insert a node
1. Find out proper location to insert
2. Connect pointers with proper sequence

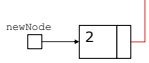
```
Step2)
1. newNode->next=list
2. list = newNode;
```


Insert a node
(The First)

```
struct Node {
        int ID;
        Node * next;
};
```

Step1) find out proper location to insert





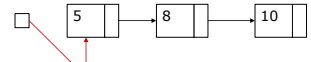
To insert a node
1. Find out proper location to insert
2. Connect pointers with proper sequence

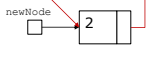
```
Step2)
1. newNode->next=list
2. list = newNode;
```


Insert a node
(The First)

```
struct Node {
        int ID;
        Node * next;
};
```

Step1) find out proper location to insert





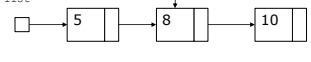
To insert a node
1. Find out proper location to insert
2. Connect pointers with proper sequence

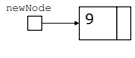
```
Step2)
1. newNode->next=list
2. list = newNode;
```


Insert a node
(between)

```
struct Node {
        int ID;
        Node * next;
};
```

Step1) find out proper location to insert

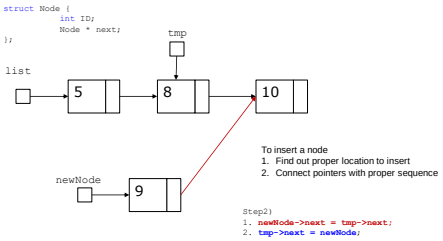




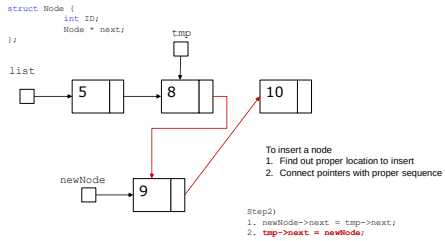
To insert a node
1. Find out proper location to insert
2. Connect pointers with proper sequence

```
Step2)
1. newNode->next = tmp->next;
2. tmp->next = newNode;
```

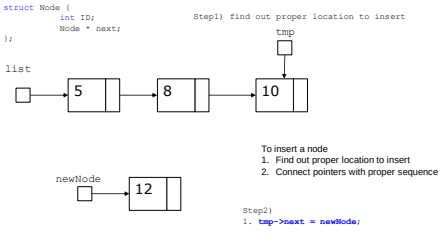
Insert a node
(between)



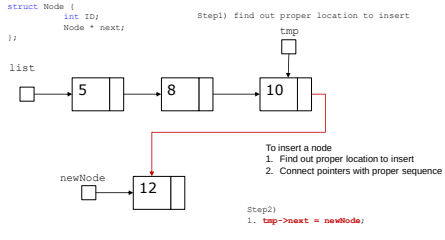
Insert a node
(between)



Insert a node
(The End)



Insert a node
(The End)



Insert a New Node in a List
(Sorted by Student ID)

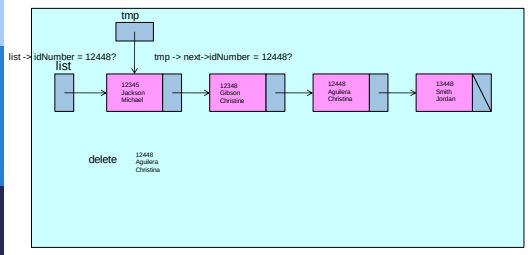
- To insert a node in the sorted list (ex by ID), need consider four different cases.
  - When list is empty
  - Insert as the first node (new node ID is smallest)
  - Insert between two nodes
  - Insert as the last (new node ID is currently biggest)
- Our challenge!
  - Insert with only one temporally pointer to the node

Delete a Node From a List

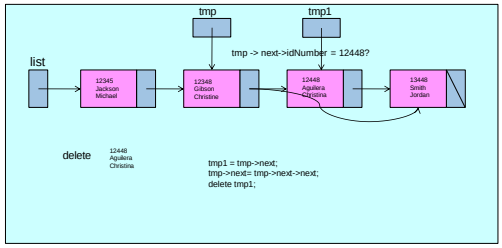
Step for deleting a node

- Search a Node in the List
  - If there is no such a node, display message and do noting
  - If there is such a node, Delete it from the list
    - Three Cases for delete
      - Delete the first node
      - Delete a node between two nodes
      - Delete the last node.

Delete a Node From a List



Delete a Node From a List

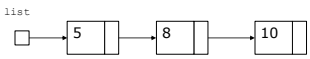


Delete a node
(The First Node)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

Step1) find out proper location to delete



Delete a node with 5

1. Find out the location of node to delete
2. Change pointers with proper sequence

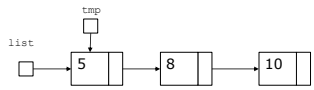
```
Step2)
1. tmp=list;
2. List=list->next
3. delete tmp;
```


Delete a node
(The First Node)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

Step1) find out proper location to delete



Delete a node with 5

1. Find out the location of node to delete
2. Change pointers with proper sequence

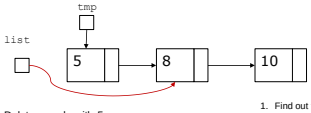
```
Step2)
1. tmp=list;
2. List=list->next
3. delete tmp;
```


Delete a node
(The First Node)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

Step1) find out proper location to delete



Delete a node with 5

1. Find out the location of node to delete
2. Change pointers with proper sequence

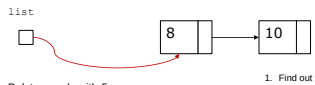
```
Step2)
1. tmp=list;
2. List=list->next
3. delete tmp;
```


Delete a node
(The First Node)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

Step1) find out proper location to delete



Delete a node with 5

1. Find out the location of node to delete
2. Change pointers with proper sequence

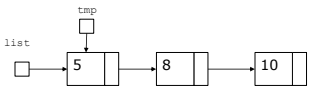
```
Step2)
1. tmp=list;
2. List=list->next
3. delete tmp;
```

Delete a node
(A Node Between)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

```
Step1) find out proper location to delete
```



Delete a node with 8

1. Find out the location of node to delete
2. Change pointers with proper sequence

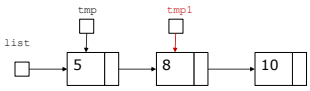
```
Step2)
1. tmp1=tmp->next;
2. tmp->next=tmp->next->next
3. delete tmp1;
```


Delete a node
(A Node Between)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

```
Step1) find out proper location to delete
```



Delete a node with 8

1. Find out the location of node to delete
2. Change pointers with proper sequence

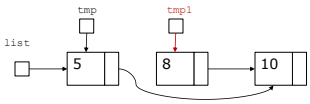
```
Step2)
1. tmp1=tmp->next;
2. tmp->next=tmp->next->next
3. delete tmp1;
```


Delete a node
(A Node Between)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

```
Step1) find out proper location to delete
```



Delete a node with 8

1. Find out the location of node to delete
2. Change pointers with proper sequence

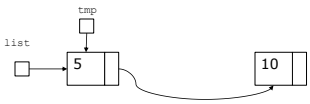
```
Step2)
1. tmp1=tmp->next;
2. tmp->next=tmp->next->next
3. delete tmp1;
```


Delete a node
(A Node Between)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

```
Step1) find out proper location to delete
```



Delete a node with 8

1. Find out the location of node to delete
2. Change pointers with proper sequence

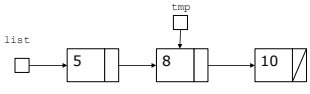
```
Step2)
1. tmp1=tmp->next;
2. tmp->next=tmp->next->next
3. delete tmp1;
```


Delete a node
(The Last Node)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

```
Step1) find out proper location to delete
```



Delete a node with 10

1. Find out the location of node to delete
2. Change pointers with proper sequence

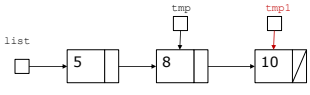
```
Step2)
1. tmp1=tmp->next;
2. tmp->next=NULL;
3. delete tmp1;
```


Delete a node
(The Last Node)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

```
Step1) find out proper location to delete
```



Delete a node with 10

1. Find out the location of node to delete
2. Change pointers with proper sequence

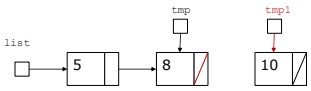
```
Step2)
1. tmp1=tmp->next;
2. tmp->next=NULL;
3. delete tmp1;
```

Delete a node
(The Last Node)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

```
Step1) find out proper location to delete
```



Delete a node with 10

1. Find out the location of node to delete
2. Change pointers with proper sequence

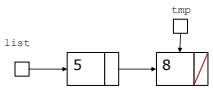
```
Step2)
1. tmp1=tmp->next;
2. tmp->next=NULL;
3. delete tmp1;
```


Delete a node
(The Last Node)

```
struct Node {
        int ID;
        Node * next;
};
```

To delete a node, we need two temp pointers
1. one pointer point to a previous node to delete
2. second pointer point to the node to delete

```
Step1) find out proper location to delete
```



Delete a node with 10

1. Find out the location of node to delete
2. Change pointers with proper sequence

```
Step2)
1. tmp1=tmp->next;
2. tmp->next=NULL;
3. delete tmp1;
```


Separating Interface from Implementation

- One of the fundamental principles of good software engineering is to separate interface from implementation.
- A program is divided into several files, in header files (.h) or program files (.cpp)
- We can compile each program file independently.
- If we modify a part of program in a program file, we only need compile the program file which is modified.

Separating Interface from Implementation

- Header file (extension with .h)– class definitions, structure type definitions, function prototypes
- Program file (extension with .cpp) – classes member functions, functions.

Separating Interface from Implementation

```
// time1.h
// Declaration of the Time class.
// Member functions are defined in time1.cpp

// prevent multiple inclusions of header file
#ifndef TIME1_H
#define TIME1_H

// Time abstract data type definition
class Time {
public:
   Time();                          // constructor
   void setTime( int, int, int );   // set hour, minute, second
   void printMilitary();            // print military time format
   void printStandard();            // print standard time format
private:
   int hour;     // 0 - 23
   int minute;   // 0 - 59
   int second;   // 0 - 59
};

#endif
```


Separating Interface from Implementation

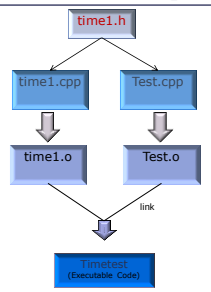
- Since a header file might possibly be included in several program files, by using **#ifndef** and **#endif** statement, we can prevent multiple inclusion.
- The preprocessor prevent the code between **#ifndef** and **#endif** from being included again, if it is already included by previous compile.

```cpp
// time1.cpp
// Member function definitions for Time class.
#include <iostream>
#include "time1.h"

using namespace std;
// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time() { hour = minute = second = 0; }

// Set a new Time value using military time. Perform validity
// checks on the data values. Set invalid values to zero.
void Time::setTime( int h, int m, int s )
{
   hour   = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
}
// Print Time in military format
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
}
// Print time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
}
```



```cpp
// Test.cpp
// Driver for Time class
// NOTE: Compile with time1.cpp
#include <iostream>
#include "time1.h"
using namespace std;

// Driver to test simple class Time
int main()
{
   Time t;  // instantiate object t of class time

   cout << "The initial military time is " <<endl;
   t.printMilitary();
   cout << " The initial standard time is " <<endl;
   t.printStandard();

   t.setTime( 13, 27, 6 );
   cout << "Military time after setTime is " <<endl;
   t.printMilitary();
   cout << "Standard time after setTime is " <<endl;
   t.printStandard();

   t.setTime( 99, 99, 99 );  // attempt invalid settings
   cout << "After attempting invalid settings: " <<endl;
   cout << "Military time: " <<endl;
   t.printMilitary();
   cout << "Standard time: " <<endl;
   t.printStandard();
   cout << endl;
   return 0;
}
```

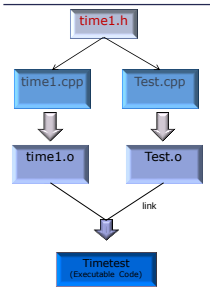

Separating Interface from Implementation



makefiles and make utility

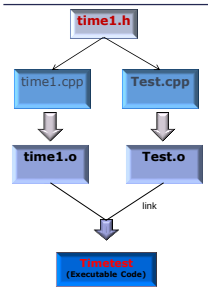
- A makefiles are special format files that together with the *make* utility will help to automatically build and manage the program with multiple source code.
- To use make utility, create a new directory and placing all the files for a project in there.
- If you call *make* utility, it will look for a file named *makefile* in the directory, and then execute it. If you have several a makefiles named MyMakefile, then you can execute them with the command:
  - make -f MyMakefile

makefiles and make utility



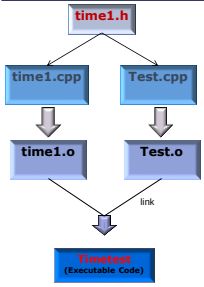

```
all: Timetest

Timetest: Test.o time1.o
    g++ Test.o time1.o  -o Timetest

Test.o: Test.cpp
    g++ -c Test.cpp

time1.o: time1.cpp
    g++ -c time1.cpp

clean:
    rm -rf *o Timetest
```


makefiles and make utility



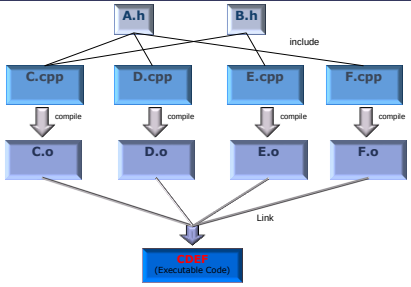

```
# CC is variable for compiler name
# CFLAGS will be the option to pass
CFLAGS = -c -Wall
CC=g++
all: Timetest

Timetest: Test.o  time1.o
      $(CC) Test.o time1.o -o Timetest

Test.o: Test.cpp
      $(CC) $(CFLAGS) Test.cpp

time1.o: time1.cpp
      $(CC) $(CFLAGS) time1.cpp

clean:
     rm -rf *o Timetest
```

makefiles and make utility




```
# CC is variable for compiler name
# CFLAGS will be the option to pass
CC=g++
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=Test.cpp time1.cpp
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=Timetest
all: $(SOURCES) $(EXECUTABLE)
$(EXECUTABLE): $(OBJECTS)
    $(CC) $(LDFLAGS) $(OBJECTS) -o $@
.cpp.o:
    $(CC) $(CFLAGS) $< -o $@
```


makefiles and make utility



makefiles and make utility

```
all: CDEF

CDEF: C.o D.o E.o F.o
    g++ C.o D.o E.o F.o -o CDEF

C.o: C.cpp
    g++ -c C.cpp
D.o: D.cpp
    g++ -c D.cpp
E.o: E.cpp
    g++ -c E.cpp
F.o: F.cpp
    g++ -c F.cpp
clean:
    rm -rf *o CDEF
```

```
# CC is variable for compiler name
# CFLAGS will be the option to pass
CFLAGS = -c -Wall
CC=g++
all: CDEF

CDEF: C.o  D.o E.o F.o
      $(CC) C.o D.o E.o F.o –o CDEF

C.o: C.cpp
      $(CC) $(CFLAGS) C.cpp

D.o: D.cpp
      $(CC) $(CFLAGS) D.cpp
E.o: E.cpp
      $(CC) $(CFLAGS) E.cpp
F.o: F.cpp
      $(CC) $(CFLAGS) F.cpp
clean:
      rm -rf *o CDEF
```


makefiles and make utility

```
# CC is variable for compiler name
# CFLAGS will be the option to pass
CC=g++
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=C.cpp D.cpp E.cpp F.cpp
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=CDEF
all: $(SOURCES) $(EXECUTABLE)
$(EXECUTABLE): $(OBJECTS)
    $(CC) $(LDFLAGS) $(OBJECTS) -o $@
.cpp.o:
    $(CC) $(CFLAGS) $< -o $@
```