C++資料結構與程式設計

C++與資料結構

NTU CSIE

大綱

使用類別(Class)建立資料結構

使用繼承(Inheritance)建立資料結構

C++物件導向

以物件為基礎的程式設計，將程式中互動的單元視為一個個的物件。

封裝（Encapsulation）

- 封裝物件資訊是第一步，您要瞭解如何使用類別定義物件，像是定義物件的屬性、方法(行為)等等，類別是建構物件時所依賴的規格書。

Example

- 物件: 狗
  - 屬性：顏色(黑、黃、白), 品種(黃金、博美…), 四肢, 尾巴
  - 行為：叫, 吃, 跑, 高興, 憤怒

類別（Class）

類別class是C++中用來封裝資料的關鍵字，當您使用類別來定義一個物件(Object)時，您考慮這個物件可能擁有的「屬性」(Property)與「方法」(Method)成員

- 屬性是物件的靜態描述
- 方法是可施加於物件上的動態操作

使用類別定義出這個物件的規格書，之後就可依這個規格書製作出一個個的物件實例，並在製作過程中設定個別物件的專屬特性資料。

要訣：

- 屬性(該類別的變數)→要存放的資料(每個物件有自己的屬性)
- 方法(該類別的函式)→寫要執行的程式(用方法控制屬性)

如何設計類別?

思考 (以功能角度)

- 每個物件需要什麼資料?
- 每個物件需要什麼方法來操作資料?

進階思考 (以使用者角度)

- 如何讓使用類別的人方便簡單使用
- 如何避免使用類別的人因資料操作不當而產生錯誤

類別（Class）

範例: 輸入兩個人資料(姓名, 身高, 體重)並印出

```
#include <iostream>
using namespace std;

class Person
{
   public:
      void input()
      {
         cin.getline(name, 80);    //gets(name);
         cin >> height;    //scanf("%d",&height);
         cin >> weight;    //scanf("%d",&weight);
         cin.ignore();     //fflush(stdin);
      }
      void output()
      {
         cout << "Name:" << name << endl;
         cout << "Height:" << height << " cm" << endl;
         cout << "Weight:" << weight << " kg" << endl;
      }
      char name[80];
      int  height;
      int  weight;
};
```

```
int main()
{
    Person p1;
    Person p2;

    p1. input();
    p1. output();
    p2. input();
    p2. output();
    return 0;
}
```

類別（Class）

宣告一個類別 *(類似定義一個結構struct)*

語法：

```
class 類別名稱
 {
      public:
         類別名稱();  //建構式, 用來做物件的初始化
         ~類別名稱(); //解構式, 用來做物件的善後工作
         公開的方法或屬性;
      protected:          // 只有在同一繼承架構中可以使用的資料
         受保護的方法或屬性;
      private:  // 只有在此類別中可以使用的資料
         私有的方法或屬性;
 };
```

資料的權限

最重要的是別忘了在最後加上分號，初學C++的新手很常犯這個錯誤

public這個關鍵字，它表示以下所定義的成員可以使用物件名稱直接被呼叫，稱之為「公開成員」

private關鍵字下的則是「私有成員」，不可以透過物件名稱直接呼叫

在類別封裝時，有一個基本原則是：資訊的最小化公開。如果屬性可以不公開就不公開，如果要取得或設定物件的某些屬性，也是儘量透過方法成員來進行

建構式與解構式

在定義類別時，您可以使用建構函式(Constructor)來進行物件的初始化

ex:

```
LinkedList::LinkedList()
{
        head = NULL;
}
```

而在物件釋放資源之前，您也可以使用「解構函式」(Destructor)來進行一些善後的工作

建構式與解構式

思考:

由上一個輸入兩個人資料(姓名, 身高, 體重)並印出的範例

你希望一開始姓名為No name, 身高與體重為0

該如何達到此功能？

類別的方法之描述

實作一個類別方法的內容(*類似寫一個函式*)

除了寫在類別定義中，也可拿到類別定義以外的地方描述

語法：

- 資料型態 類別名稱::方法名稱(參數1, 參數2, … ,參數n)
  {
  　　程式碼;
  }

Ex:

```
void LinkedList::insert_node ( node *ptr, int value)
{
        …
}
```

物件的產生與使用

使用類別定義物件 *(類似定義一個變數)*

語法：

- 類別名稱 物件名稱;
- 類別名稱 物件名稱(參數1, 參數2, … , 參數n);

物件
可透過 . 來使用或存取該方法或屬性
(類似C語言的結構struct)

若為物件指標，
可透過 -> 來使用或存取該方法或屬性

```
Person p1;
Person *p2;
p2  = &p1;

p1. input();
p1. output();
p2 -> input();
p2 ->output();
```

Friend

在定義類別成員時，私用成員只能被同一個類別定義的成員存取，不可以直接由外界進行存取

然而有些時候，您希望提供私用成員給某些外部函式或類別來存取，這時您可以設定類別的「好友」，這些好友才可以直接存取自家的私用成員

Friend類別

- 使用friend函式通常是基於效率的考量，以直接存取私用成員而不透過函式呼叫的方式，來省去函式呼叫的負擔

- 您也可以將某個類別宣告為friend類別，被宣告為friend的類別可以直接存取私用成員

Friend類別

思考：

由建立出兩個正方型物件

並算出其面積之範例

你需要提供一個尺類別讓使用者可以

比較兩正方型之大小

該如何達到此功能？

Friend類別

下面這個程式中使用friend關鍵字來設定一類別為另一類別的好友

```
#include <iostream>
using namespace std;
class Square
{
  public:
    Square(int n)     {
      len = n;
    }
    int getLen()     {
      return len;
    }
    int area()     {
      return len*len;
    }
    friend class Ruler;

  private:
    int len;
};
```

```
class Ruler
{
  public:
    Ruler(int n)     {
      len = n;
    }
    void compareSquare(Square &s1, Square &s2)     {
      // 可直接存取私用成員
      if( (len < s1.len) || (len < s2.len) )
        cout << "尺太短, 無法量測" << endl;
      else       {
        if(s1.len > s2.len)
          cout << "s1較大" << endl;
        else if(s1.len == s2.len)
          cout << "s1跟s2一樣大" << endl;
        else
          cout << "s2較大" << endl;
      }
    }
  private:
    int len;
};
```

Friend類別

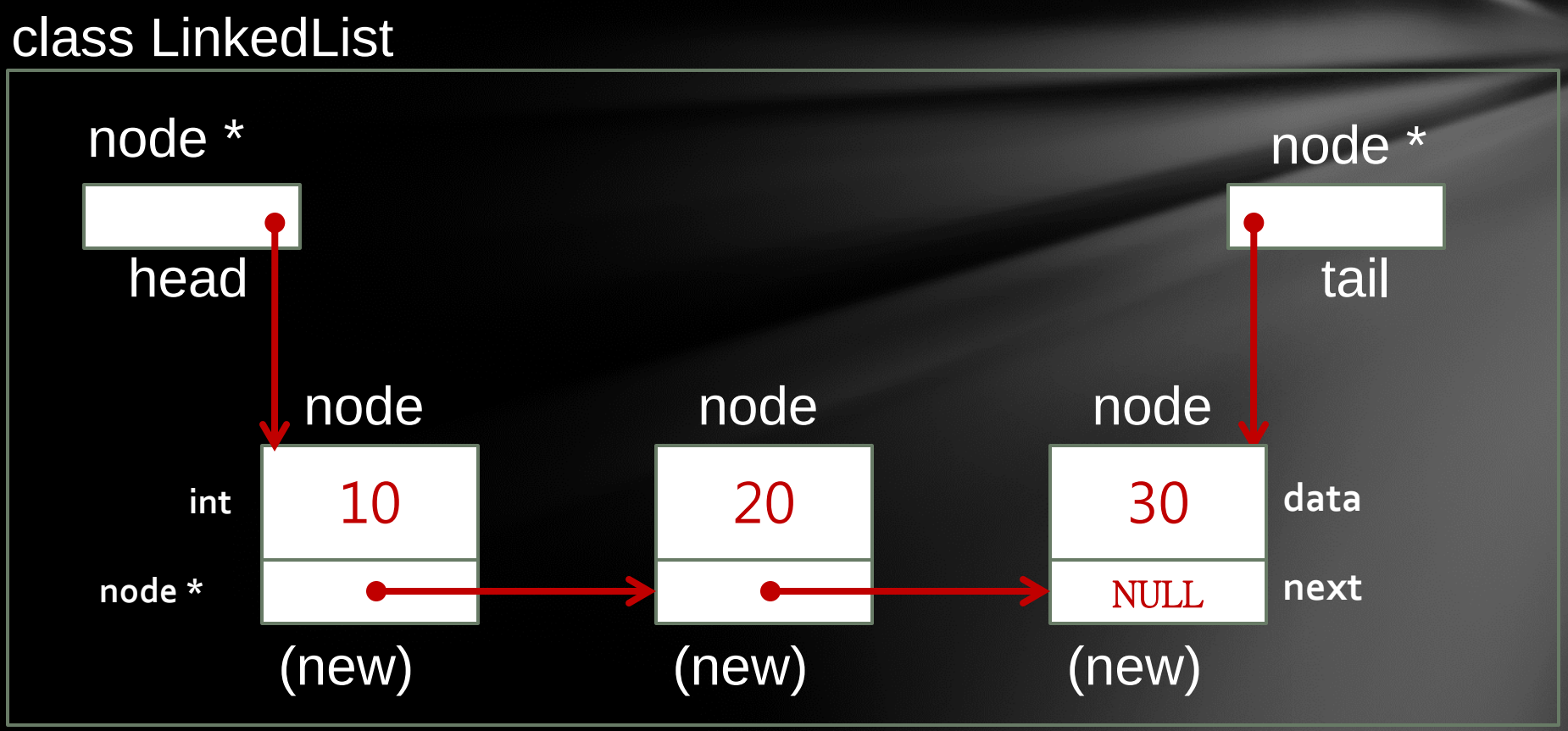
- 主程式可建立Ruler物件比較兩方型大小

```
int main()
{
    Square s1(10);
    Square s2(20);
    Ruler r(30);

    cout << "s1: len = " << s1.getLen() << ", area = " << s1.area() << endl;
    cout << "s2: len = " << s2.getLen() << ", area = " << s2.area() << endl;

    r.compareSquare(s1, s2);

    return 0;
}
```

範例:C++鏈結串列

將第二章之鏈結串列範例改用C++之類別實作

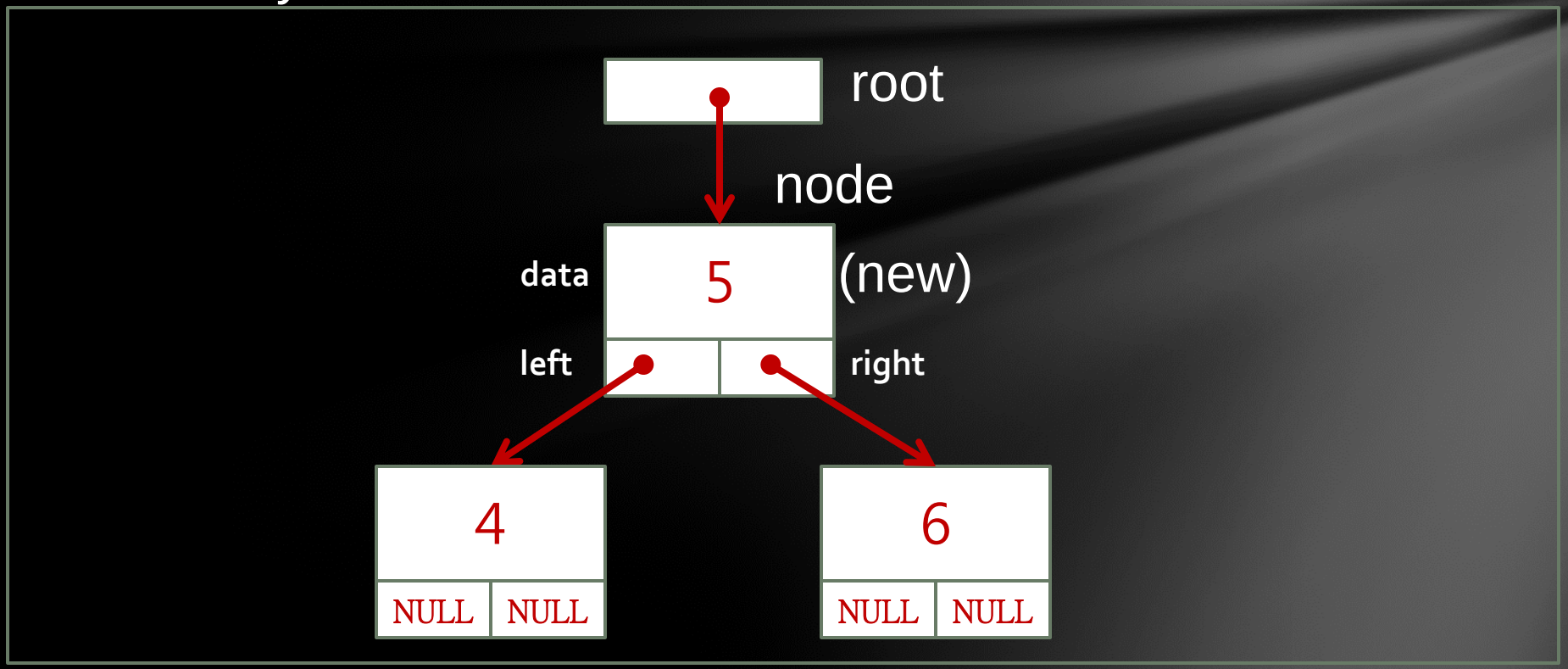
(c++_linked_list.cpp)



回家作業:C++樹狀結構

將第六章之二元搜尋樹範例改用C++之類別實作

class BinarySearchTree



大綱

使用類別(Class)建立資料結構

使用繼承(Inheritance)建立資料結構

繼承（Inheritance）

「繼承」（Inheritance）是物件導向程式設計的一種進階觀念

繼承就是物件的再利用，當定義好一個類別後

其他類別可以繼承這個類別的成員資料和函數

語法:

```
class 子類別名稱: 繼承權限 父類別名稱
{
    …
};
```

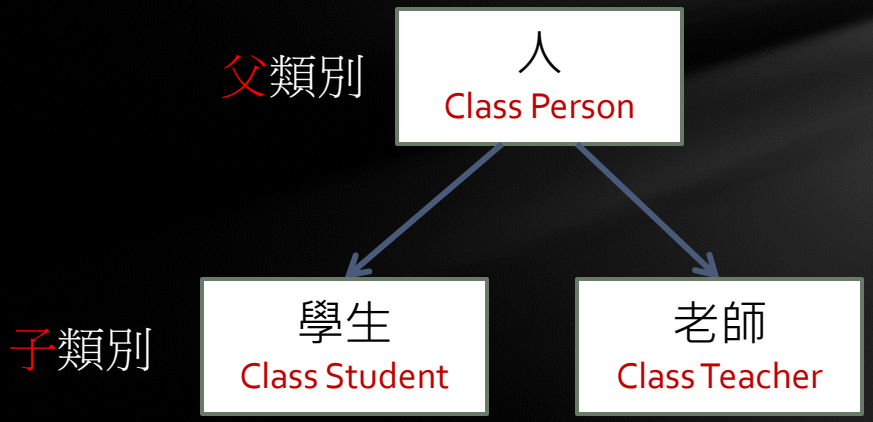
在繼承的關係中

- 被繼承的類別：
  - 「父類別」（Parent class）或「基礎類別」（Base class）
- 繼承父類別的類別：
  - 「子類別」（Child class）或「衍生類別」（Derived class）

繼承（Inheritance）

範例:

- 類別繼承也是在模擬真實世界，例如：學生和老師都是人，我們可以先定義Person類別來模擬人類，然後擴充Person類別建立Student類別來模擬學生，Teacher類別來模擬老師
- (Inheritance01.cpp)



繼承（Inheritance）

範例:


```
class Person
{
   public:
      void inputPerson()
      {
         char str[128];
         cout << "<輸入個人資料>" << endl;
         cout << "姓名: ";
         fflush(stdin);
         cin.getline(str, 128);
         Name = str;
         cout << "電話: ";
         cin >> Phone;
         cout << "Email: ";
         cin >> Email;
      }
      void outputPerson()
      {
         cout << "<印出個人資料>" << endl;
         cout << "姓名: " << Name <<endl;
         cout << "電話: " << Phone <<endl;
         cout << "Email: " << Email <<endl;
      }
   private:
      string Name;
      string Phone;
      string Email;
};
```

Student

```
Student s1;
s1.inputPerson();
s1.inputStudent();
cout << endl;
s1.outputPerson();
s1.outputStudent();
```

```
class Student: public Person {
        public:
                void inputStudent() {
                        cout << "<輸入學生資料>" << endl;
                        cout << "學號: ";
                        cin >> StudentID;
                        cout << "系所: ";
                        cin >> Department;
                }
                void outputStudent() {
                        cout << "<印出學生資料>" << endl;
                        cout << "學號: " << StudentID <<endl;
                        cout << "系所: " << Department <<endl;
                }
        private:
                string StudentID;
                string Department;
};
```

Teacher

```
Teacher t1;
t1.inputPerson();
t1.inputTeacher();
cout << endl;
t1.outputPerson();
t1.outputTeacher();
```

```
class Teacher: public Person {
        public:
                void inputTeacher() {
                        cout << "<輸入老師資料>" << endl;
                        cout << "職稱: ";
                        cin >> Title;
                        cout << "系所: ";
                        cin >> Department;
                }
                void outputTeacher() {
                        cout << "<印出老師資料>" << endl;
                        cout << "職稱: " << Title <<endl;
                        cout << "系所: " << Department <<endl;
                }
        private:
                string Title;
                string Department;
};
```

範例(C++堆疊與佇列)

使用C++鏈結串列以繼承方式實作堆疊與佇列

- (c++_stack_queue.cpp)