Chapter 11

Inheritance

Chapter 11 Topics(part 2)

- Constructor and Destructor for Subclass
  - Declarations
  - Invoking Sequence
- Subtype, Subclass and Substitution（替换）
  - Some Basic Concepts
  - The Substitutability Paradox (可替换性悖论)
  - The Problem with Substitution

Declaration for Constructor of Subclass

SYNTAX

```
ChildClassName(ParameterList):ParentClassName(ParameterList0)
{
   // Initialize new data members in child class
}
```

Example

```
class Base{
  public:
    Base(int n)
   { i=n; }
  private:
    int i;
};
```

```
class Derive:public Base{
  public:
    Derive(int m, int n):Base(m)
   { j=n; }
  private:
    int j;
};
```

Declaration for Destructor of Subclass

SYNTAX

```
~ChildClassName( )
{
   ...
}
```

Invoking Sequence

When a new instance of child class is created,

- the constructor of parent class is invoked first,
- then the constructor of child class is invoked

When an instance of child class is destroyed,

- the destructor of child class is invoked first,
- then the destructor of parent class is invoked

Example

```
class Base{
  public:
    Base(int n)
    { i=n;
      cout<<“Constructing base class,i=”<<i<<endl;}
    ~Base( )
    { cout<<“Destructing base class ,i=”<<i <<endl; }
  private:
    int i;
};
```

```
#include<iostream>
using namespace std;
int main( )
{
    Derive obj(10, 20);
    return 0;
}
```

```
class Derive:public Base{
  public:
    Derive(int m, int n):Base(m)
    { j=n;
      cout<<“Constructing derive class, j=”<<j<<endl; }
   ~Derive( )
    {cout<<“Destructing derive class , j=”<<j <<endl;}
  private:
    int j;
};
```

Declaration for Constructor of Subclass with Instance Fields

SYNTAX

```
ChildClassName(ParameterList):ParentClassName(ParameterList0),
SubObject1(ParameterList1), … , SubObjectN(ParameterListN)
{
   // Initialize new data members in child class
}
```

Example

```
class Base{
  public:
    Base(int n)
   { i=n; }
  private:
    int i;
};
```

```
class Derive:public Base{
  public:
    Derive(int m, int n, int k):Base(m),d(k)
   { j=n; }
  private:
    int j;
    Base d;
};
```

Invoking Sequence

When a new instance of child class is created,

- the constructor of parent class is invoked first,
- then the constructors of subobjects are invoked,
- and then the constructor of child class is invoked

When an instance of child class is destroyed,

- the destructor of child class is invoked first,
- then the destructors of subobjects are invoked,
- and then the destructor of parent class is invoked

Example

```
class Base{
  public:
    Base(int n)
    { i=n;
      cout<<"Constructing base class,i="<<i<<endl;}
    ~Base( )
    { cout<<"Destructing base class ,i="<<i<<endl; }
  private:
    int i;
};
```

```
#include<iostream>
using namespace std;
int main( )
{
    Derive obj(10, 20,30);
    return 0;
}
```

```
class Derive:public Base{
  public:
    Derive(int m, int n , int k):Base(m),d(k)
   { j=n;  cout<<"Constructing derive class, j="<<j<<endl; }
  ~Derive( )
   {cout<<"Destructing derive class , j="<<j <<endl;}
  private:
    int j;
    Base d;
};
```

Chapter 11 Topics(part 2)

- Constructor and Destructor for Subclass
  - Declarations
  - Invoking Sequence
- Subtype, Subclass and Substitution
  - Some Basic Concepts
  - The Substitutability Paradox
  - The Problem with Substitution

Concepts of Subclass and Subtype

- To assert that one class is a *subclass* of another is to simply say it was built using inheritance. It is a statement concerning how it was constructed.

- To assert that one class is a *subtype* of another is to say that it preserves the purpose of the original. It is a statement concerning meaning.

An Argument for Substitution

- Instances of the child class must **possess**(拥有) all data members associated with the parent class.
- Instances of the child class must implement, through inheritance at least (if not explicitly overridden), all **functionality**(功能) defined for the parent class.
- Thus, an instance of a child class can **mimic**(模仿) the behavior of the parent class and should be ***indistinguishable***(不能区别的) from an instance of the parent class if substituted in a similar situation.

The Substitutability Paradox

- Substitution is permitted, based on subclass. That is, a variable declared as the parent type is allowed to hold a value derived from a child type.
- Yet from a semantic point of view, substitution only makes sense if the expression value is a subtype of the target variable.

The Undecidability of the Subtype Relationship

One of the classic corollaries(推论) of the halting problem(停机问题) is that there is no procedure that can determine, in general, if two programs have equivalent behavior.

It is extremely difficult to define meaning, and even if you can it is almost always impossible to determine if one class preserves the meaning of another.

The Problem with Substitution

```
class Window {
  public:
  ...
  private:
    int height;
    int width;
};
```

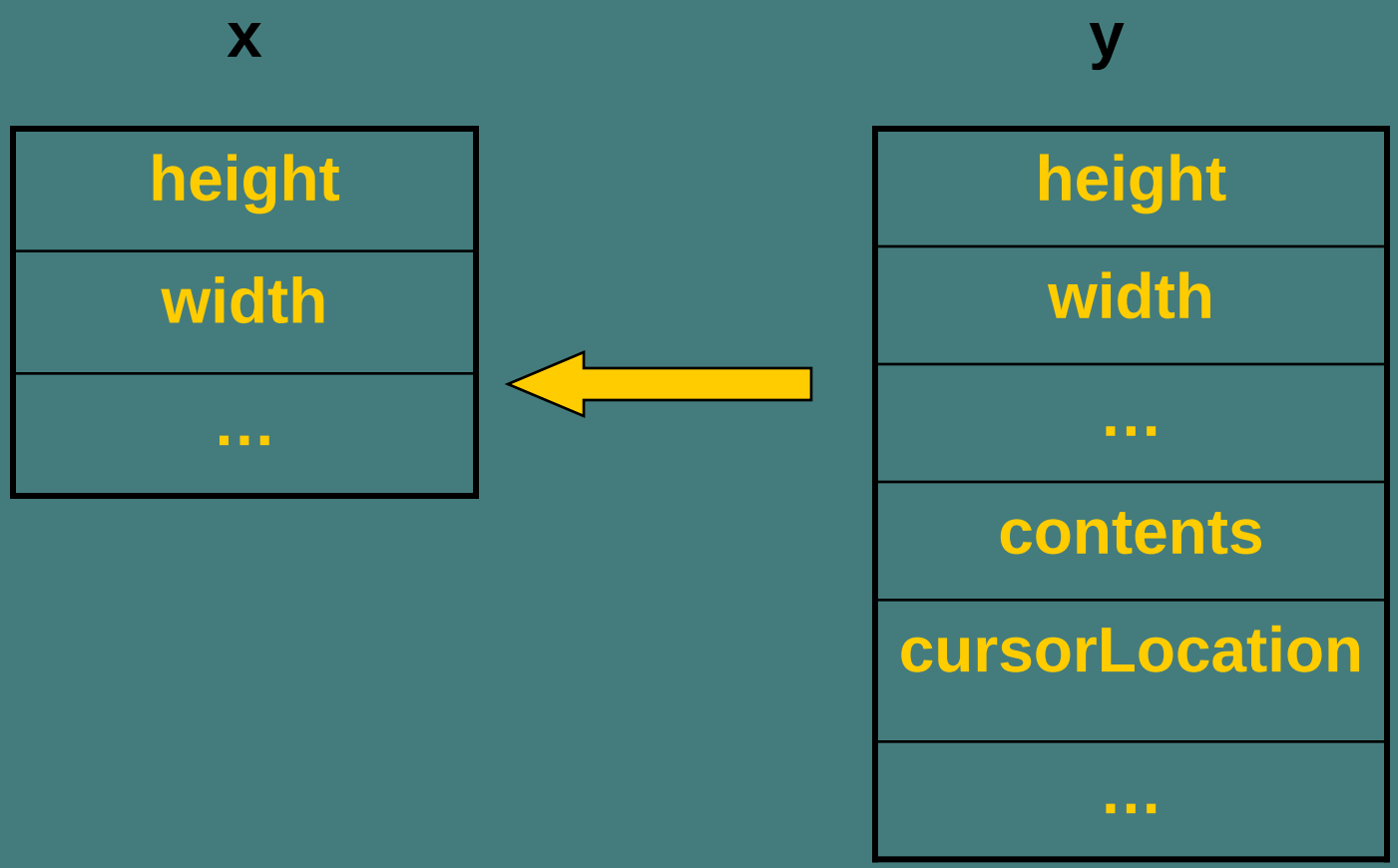
```
class TextWindow : public Window {
  public:
  ...
  private:
    char * contents;
    int cursorLocation;
};
```

```
Window x; //
TextWindow y;
x = y; //
```

Memory Allocation Strategies (内存分配策略)

- ***Minimum Static Space Allocation*— Allocate the amount of space necessary for the base class only. (C++)**
- ***Maximum Static Space Allocation* — Allocate the amount of space for the largest legal value.(not used in practice)**
- ***Dynamic Space Allocation* — Allocate for x only the amount of space required to hold a pointer. (Smalltalk, Java)**

Assigning a Larger Value to a Smaller Box



The Slicing(切片) Problem

The problem is you are trying to take a large box and squeeze(挤) it into a small space. Clearly this won't work. Thus, the extra fields are simply sliced off.