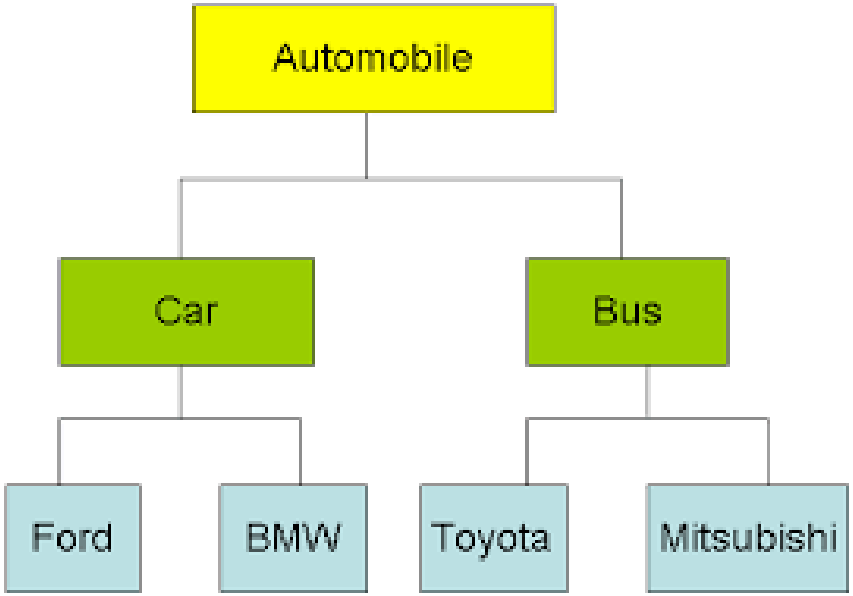
C/C++ Programming: C++ advan. (1/3)



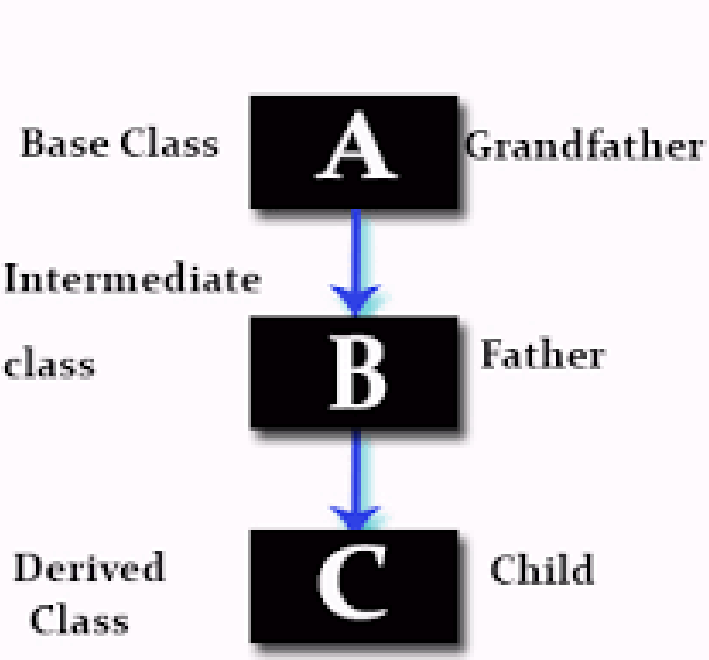
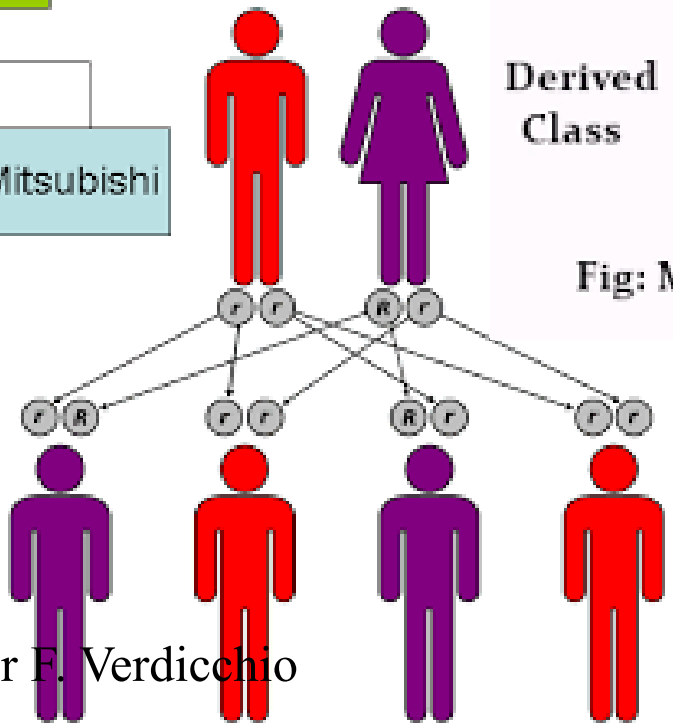


Fig: Multilevel Inheritance



Any question?

Relevant topics to ask questions:

- C++ Objects:

Inheritance

Start with the intArray class

```
class intArray{
public:
    static const int MAX_SIZE=20;
private:
    // holds the content
    int arrayItemContent[MAX_SIZE];
    // indicates if teh element is empty
    bool arrayItemEmpty[MAX_SIZE];
    // tot non-empty items
    int tot_items;

    void resetArray() { ... }

    void sortedAppend( int val) { ... }
public:
    // constructor; upon construction the array is "empty".
    intArray() { ... }
    //If position "pos" is a valid position and corresponding item is "empty", return true; false otherwise;
    bool isItemEmpty(int pos) { ... }
    //Check that position "pos" is a valid position for this array; if so return true; false otherwise;
    bool checkPos(int pos) { ... }
    //Insert "val" in position "pos" if this is a valid position and
    // if the corresponding item is "empty", then return true; false otherwise;
    bool insertItem(int val, int pos) { ... }
    int getItem(int pos) { ... }
    //Pint to screen position and value of all "non-empty" elements in the array.
    void printArrayContent() { ... }
    //The number of "non-empty" items in the array (inserted so far by the user);
    int getTotValidItems() {return tot_items;}
    //If a "non-empty" item is found in position "pos", its value is written to "val",
    // the item becomes "empty" and true is returned; false otherwise;
    bool removeItemVal(int pos, int& val) { ... }
    //If position "pos" is a valid position for this array, reset the corresponding item (so that it becomes "empty")
    bool resetItem(int pos) { ... }
    void sortArray(bool ascending=true) { ... }
};
```

Inheritance

Start with the intArray class

This class implements a fixed-size array that accommodates integer items; can print out array content; sort array content; etc.

Inheritance

Start with the intArray class

This class implements a fixed-size array that accommodates integer items; can print out array content; sort array content; etc.

What about a class that: implements a fixed-size array that accommodates integer items within set bounds (min, max); can print out array content; sort array content; etc.

Inheritance

Start with the intArray class

This class implements a fixed-size array that accommodates integer items; can print out array content; sort array content; etc.

What about a class that: implements a fixed-size array that accommodates integer items within set bounds (min, max); can print out array content; sort array content; etc.

This is a “variation/specialization” of the class above;
can we “borrow” from it ?!

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

Derived class (inheriting from the base class)

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

Base class

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

An object of the derived class comprises all the member variables and functions of the base class.

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

An object of the derived class comprises all the member variables and functions of the base class.
In addition, the derived class can introduce any member variable / functions of its own

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

In this case:
boundaries (min , max)
flag (boundaries set)

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

In this case:

constructor;

setBounds(…)

(overloaded) insetItem(…)

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

Constructor:

When the derived object is instantiated the following happens:

1. The constructor of the base class is called (automatically)
2. The constructor of the derived class is called

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

setBounds(…)
Verifies and accepts (only once) bounds (min. max) for valid inputs

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

(overloaded) insetItem(…)
same declaration as the function in the parent class (does not have to be for an overloaded function – inputs may differ);

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

(overloaded) insetItem(…)
same declaration as the function in the parent class (does not have to be for an overloaded function – inputs may differ);

Clips the input (val) to stay within the set bounds (min, max)

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

(overloaded) insetItem(…)
same declaration as the function in the parent class (does not have to be for an overloaded function – inputs may differ);

Clips the input (val) to stay within the set bounds (min, max);

Then call the base-class input function with the “restricted” input val

Inheritance

```
class intArray { ... };

class intArrayWithBounds : public intArray{
protected:
    int min_val, max_val;
    bool bounds_set;
public:
    intArrayWithBounds(){bounds_set=false;}
    bool setBounds(int inp_min_val, int inp_max_val)
    {
        bool result=false;
        if(!bounds_set)
        {
            if(inp_min_val <= inp_max_val)
            {
                min_val=inp_min_val;
                max_val=inp_max_val;
            }
            else
            {
                min_val=inp_max_val;
                max_val=inp_min_val;
            }
            bounds_set=true;
            result=true;
        }
        return result;
    }
    bool insertItem(int val, int pos)
    {
        if(bounds_set)
        {
            if(val<min_val)
                val=min_val;
            if(val>max_val)
                val=max_val;
        }
        return intArray::insertItem(val, pos);
    }
};
```

(overloaded) insetItem(…)
same declaration as the function in the parent class (does not have to be for an overloaded function – inputs may differ);

Clips the input (val) to stay within the set bounds (min, max);

Then call the base-class input function with the “restricted” input val

Inheritance: test

```
void test_intArray()
{
    intArray testarray;
    for (int pos = 0; pos < intArray::MAX_SIZE; pos++)
    {
        int val = rand() % 100;
        testarray.insertItem(val, pos);
    }
    testarray.printArrayContent();
    cout << endl;
    testarray.sortArray();
    testarray.printArrayContent();
}
```

```
void test_intArray()
{
    intArrayWithBounds testarray;
    testarray.setBounds(10, 50);
    for (int pos = 0; pos < intArray::MAX_SIZE; pos++)
    {
        int val = rand() % 100;
        testarray.insertItem(val, pos);
    }
    testarray.printArrayContent();
    cout << endl;
    testarray.sortArray();
    testarray.printArrayContent();
}
```

Any question?