– Institut für Mechanik und Mechatronik / Abteilung 6
– Intelligente Handhabung und Robotertechnik

Einführung in C(++)

VO 318.060

Part 2 C++

o.Univ.-Prof. Dr. Dr.h.c.mult. Peter Kopacek

Jänner 2005

– Technische Universität Wien

Vienna University of Technology – E325 Institut für Mechanik und Mechatronik / A6 IHRT

Object-Oriented Programming(OOP)

Object oriented programming is a programming-method, which is very close to the way everybody solves the tasks of everyday live. To solve the complexity surrounding us, we use the possibility to generalize, to classify and to abstract. Many words we are using represent classes. The members of these classes have common attributes or behaviours. So we can talk about "dogs" without specifying a distinct animal. Object oriented programming uses this methods of abstractions and classifications

The three fundamental attributes of OOP:

- **Encapsulation**
  - The combination of data-structures and functions (methods), which manipulate these data-structures, results in an new type: an **OBJECT**.

- **Inheritance**
  - New objects are incorporated into an hierarchical object-tree. The new object takes over all data-structures and functions from their parents. Now this new object can be changed by adding new data-structures and/or functions.

- **Polymorphism**
  - One function has a unique name within the whole hierarchy. This function is implemented in each object of the hierarchical tree in a different way.

Encapsulation

Example: Font-manipulation
You have a data-structure (e.g.: array) and functions to show, rotate and scale the font.

TRADITIONAL PROGRAMMING	OBJECT ORIENTED PROGRAMMING
struct demo { .... } data { init(); get(..); rotate(..) print(..); ... } code	class { .... } elementary functions constructor (..) get(..); rotate(..) print(..);
Definition of data-structures and functions are divided. It is possible to manipulate the data without using the predefined functions.	Normally the data-structure is a private element and therefore can be manipulated only by functions within the same object. These functions are defined as public and can be used by others.

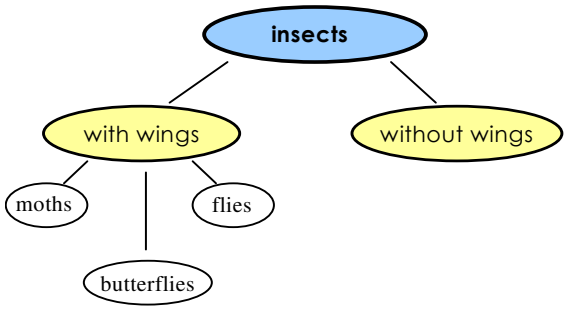
Now there is a need to change the data-structure (e.g. from the array to a list):

Traditional programming	OOP
All predefined functions must be rewritten. All Code which manipulates the data structure directly and was written by another programmer must be *searched* and *rewritten.*	Only the functions within the object must be rewritten. All users of this object even don't know that the object is reconstructed.

Vienna University of Technology – E325 Institut für Mechanik und Mechatronik / A6 IHRT
TU VIENNA

Inheritance

Researchers in the field of insects e.g. do some classifications similar to the figure above. If we have a new insect (or object) we have to answer some questions to classify them: What is the similarity and what is the difference to the members of the main class? Each class has its own characteristics and behaviour, which defines them and distinguishes them to other classes.



When inserting a new insect (object) we start at the top of the tree. First we have more generalized questions (Do they have wings or not ?) but coming down, the questions become more detailled. You automatically know that a insect has wings, if you decide a new insect is from class fly. If you define an attribute in a class all lower classes *"inherit"* these attributes (data-structures and functions !).

As you see in this example it is a very difficult task to find the optimal class-hierarchie for an application.

BEFORE WRITING ONLY ONE LINE OF CODE USING OOP IT IS VERY IMPORTANT TO DEFINE ALL CLASSES AT ALL LEVELS !

Nowadays there are a lot of Class-Libraries available (e.g MFC, TVISION and WINDOWS-Programming for user-interfaces and others for general lists sorting methods and so on).

POLYMORPHISM

In the traditional programming methods each function has to have a unique name. This name is used by the compiler to identify what function has to be used.

In OOP the Polymorphism is a new and very important feature compared to traditional programming. Within a class hierarchie you can define one function-name and implement this function in each class in a different way. At run-time the system decides what function has to be used (from class A or class B or C.....).

Vienna University of Technology – E325 Institut für Mechanik und Mechatronik / A6 IHRT
TU VIENNA

Example

```
// class Location describes screen locations in X and Y coordinates

// class Point describes whether a point is hidden or visible

// class Circle describes the radius of a circle


enum Bool {False, True};
```

A Classe Location is created.

```
class Location {

protected:          // allows derived class to access private data

   int X;                              // actual X-position

   int Y;                              // actual Y-position

public:                          //  these  functions  can  be  accessed  from
outside

   Location(int InitX, int InitY);     // Initialise

   int GetX();                         // returns actual X-Position

   int GetY();                         // returns actual Y-Position

};
```

Now we create a new class POINT based on LOCATION. Therefore Point inherits all data and functions from LOCATION. We only add a new data word VISIBLE which indicates if the point is lighted or not. One important new function is MOVETO which allows the class POINT to move to another place on the screen.

Furthermore POINT has two new functions SHOW and HIDE which are defined as VIRTUAL and therefore can be overwritten in derived classes (Polymorphism). We will see in the implementation that we need SHOW and HIDE to do MOVETO in a easy way. MOVETO is now available for all derived classes when they define their own SHOW and HIDE functions.

Vienna University of Technology – E325 Institut für Mechanik und Mechatronik / A6 IHRT

```
class Point : public Location {

 //derived from class Location

 //public derivation means that X and Y are protected within Point

protected:

 Bool Visible; //classes derived from Point will need access to Visible

 public:
   Point(int InitX, int InitY);     // constructor

   virtual void Show();             // Show the point on screen

   virtual void Hide();             // Hide the point

   Bool IsVisible();                // returns actual state (Visible or not)

   void MoveTo(int NewX, int NewY);       // Move Point to new location

};
```

The class CIRCLE is based on POINT. We add the new data word RADIUS and overwrite the function SHOW and HIDE because a circle must be drawn in a different way. Furthermore we add the new functions EXPAND and CONTRACT to manipulate the circle. Because of the INHERITANCE of OOP, CIRCLE is able to MOVETO another location and to be visible or not

```
class Circle : public Point {              // derived from class Point

                                           //   and   ultimately   from   class
Location

  protected:

    int Radius;                            // actual value for Radius

  public:

    Circle (int InitX, int InitY, int InitRadius);      // constructor

    virtual void Show();                   // Show the Circle on screen

    virtual void Hide();                   // Hide the Circle

    void Expand(int ExpandBy);             // Make Circle bigger

    void Contract(int ContractBy);         // Makes Circle smaller

};
```

Vienna University of Technology – E325 Institut für Mechanik und Mechatronik / A6 IHRT

In the described way more classes can be derived (e.g lines, rectangles etc.) to build a complex object tree. This can be used to build new application (e.g. CAD-Systems or simulation-programs).

In the following we describe the implementation of the Class LOCATION:

```
/*-------------------------------------------------------------------*/
Location::Location(int InitX, int InitY)   // The Constructor of LOCATION
{
  X=InitX; Y=InitY;                        // Set init-values
}
int Location::GetX()
{
  return (X);                              // X is accessable in the object
}
int Location::GetY()
{
  return (Y);                              // Y is accessable in the object
}
```

In the following we describe the implementation of the Class POINT:

```
/*-------------------------------------------------------------------*/
Point::Point(int InitX, int InitY) : Location (InitX,InitY) // The
Constructor
{
  Visible = False;                         // initially point is not visible
}
```

This is the implementation of the function SHOW for the class POINT. It must be overwritten for derived classes.

```
void Point::Show()
{
  Visible=True;                          // Point becomes now visible
  putpixel (X,Y,getcolor());             // draw point
}
```

This is the implementation of the function HIDE for the class POINT. It must be overwritten for derived classes.

```
void Point::Hide()
{
  Visible=False;                         // Point becomes now invisible
  putpixel (X,Y,getbkcolor());           // draw point with background color
}
```

This is the implementation of the function ISVISIBLE. This Function can be used in derived classes.

```
Bool Point::IsVisible()
{
  return (Visible);                      // return current state
}
void Circle::Hide()
{
```

Vienna University of Technology – E325 Institut für Mechanik und Mechatronik / A6 IHRT
TU VIENNA

```
  unsigned int TmpColor = getcolor();      // save current color

  setcolor(getbkcolor());            // new color is background

  Visible=False;                           // erase circle

  circle (X,Y,Radius);                     //  draw  circle  with  background-
color

  setcolor(TmpCOlor);                      // restore old color

}

void Circle::Expand (int ExpandBy)

{

  Hide();                                  // erase old circle

  Radius +=ExpandBy                        // new radius

  if (Radius < 0)

    Radius = 0;                            // avoid negative radius

  Show();

}

void Circle::Contract(int ContractBy)    // Makes Circle smaller

{

  Expand (-ContractBy);

}
```

Here is a short program segment which defines two Objects - one Point and one Circle. First both objects are shown. In a next step ACircle shall move to a new location. This is done by using the function POINT::MOVETO. POINT::MOVETO calls the virtual functions CIRCLE::SHOW and CIRCLE::HIDE to draw a correct circle. After this we test if ACircle is visible and if so we hide it.

```
 .....               // Here are some definitions

int main ()

{
```

```
    .....
  Circle ACircle (10,10,5);     // define a Circle at 10/10 with Radius 5
  Point APoint (120,110);       // and a point at 120/110
  ACircle.Show();               // show the circle on the screen
  APoint.Show();                // show the point on the screen
  ACircle.MoveTo (10,30);       // Move the circle to a new location
                                // CIRCLEW::SHOW and CIRCLE::HIDE are
                                // called when executing Moveto !!
  APoint.MoveTo (5,5);          // Move Point to 5/5
                                //(POINT::SHOW and POINT::HIDE are used)
                                //But both use POINT::MOVETO
  if (ACircle.IsVisible())      // if ACircle is visible we hide it.
    ACircle.Hide();
    ....
}
```

Vienna University of Technology – E325 Institut für Mechanik und Mechatronik / A6 IHRT

TU VIENNA

Constructor and destructor

There are two ways to define an integer variable.

1. You define the variable and then assign a value to it later in the program.

```
int Radius;          // define a variable
...                  // other code here
Radius = 5;          // assign it a value
```

2. you can define the integer and immediately initialize it.

```
int Radius = 5;     // define and initialize to 7
```

Initialization combines the definition of the variable with its initial assignment. You can change the value later. Initialization ensures that your variable has a distinct value.

How do you initialize the member data of a class?

Classes have a special member function called a constructor. The constructor can take parameters as needed, but it cannot have a return value--not even `void`. The constructor is a class method with the same name as the class itself.

Whenever you declare a constructor, you'll also want to declare a destructor.

- constructors create and initialize objects of your class,
- destructors clean up after your object and free any memory you might have allocated.

A destructor always has the name of the class, preceded by a tilde (~). Destructors take no arguments and have no return value.

Literature

Bjarne Stroustrup, "The C++ Programming Language", Second Edition, Addison-Wesley Publishing Company, 1991.