C++ Classes

Εργίνα Καβαλλιεράτου

Σημερινό μάθημα

- Επικοινωνία με την κλάση
- Πρόσβαση σε στοιχεία κλάσης
- Απόδοση τιμής σε μεταβλητή
- Επιστροφή δεδομένων
- Αρχικοποίηση κλάσης
- Constructor και destructor
- Overloading συναρτήσεις-μέλη
- Overloading προκαθορισμένους τελεστές

Επικοινωνία με την κλάση

- Για να καλέσουμε μία συνάρτηση-μέλος από το κυρίως πρόβλημα γίνεται ΠΑΝΤΑ με αναφορά στο αντικείμενο που αντιστοιχεί:

<όνομα αντικειμένου>.<όνομα συνάρτησης-μέλους>

Παράδειγμα

```
#include…
class Cat {
        unsigned int itsAge;
        unsigned int itsWeight;
        Meow();
}; …
main()
{
unsigned int GrossWeight;
Cat Frisky;
…}
```

C++ Class

```
class Classic_Example {

public:

// Δεδομένα και διαδικασίες προσβάσιμα από παντού

protected:
//Δεδομένα και διαδικασίες προσβάσιμα από την κλάση
//  τις παραγόμενες κλάσεις και  friends κλάσεις

private:

// Δεδομένα και διαδικασίες προσβάσιμα από την κλάση

// και  friends κλάσεις

};
```

Ότι είναι public, έχει πρόσβαση από οπουδήποτε στο πρόγραμμα

```
#include <iostream.h>
class Cat
{ public:
      int itsAge;
      int itsWeight; };
void main() {
         Cat Frisky;
         Frisky.itsAge = 5;
         cout << "Frisky is a cat who is " ;
         cout << Frisky.itsAge << " years old.\n"; }
```

Πρόσβαση σε στοιχεία κλάσης

- Αν δεν διευκρινιστεί ο τύπος των δεδομένων στην κλάση, ο εξ’ ορισμού τύπος είναι private
- Στα private δεδομένα δεν μπορούμε να έχουμε πρόσβαση εκτός κλάσης.
- Αυτό συμβαίνει για την ασφάλεια των δεδομένων
- Για το λόγο αυτό πρέπει να δημιουργήσουμε κατάλληλες συναρτήσεις για εργασίες όπως
- Απόδοση τιμής σε μεταβλητή
- Επιστροφή δεδομένων κλπ.

Τελεστής Εμβέλειας ::

- “Όταν δηλώνουμε πρότυπο συνάρτησης μέσα σε μία κλάση και θέλουμε να ορίσουμε τη συνάρτηση εκτός κλάσης πρέπει να χρησιμοποιήσουμε τον Τελεστής Εμβέλειας ::

Τελεστής Εμβέλειας ::

```
class foo {
  ...
            int sum(void);
};
int foo::sum(void) {
  int sum=0;
  for (int i=0;i<foo::i;i++) {
       sum += a[i];
  }
  return(sum);
}
```

Απόδοση τιμής σε μεταβλητή

- Υπάρχουν συναρτήσεις που χρησιμοποιούνται για να αποδώσουν τιμή σε private μεταβλητή μιας κλάσης
- Συνήθως ονομάζονται

Set_<όνομα μεταβλητής>

Απόδοση τιμής σε μεταβλητή

```
class Cat {
public:
        void SetAge (int age);
        void Meow();
private:
        int itsAge;
};
…
void Cat::SetAge(int age) {
        itsAge = age;
}
…
```

Επιστροφή δεδομένων

- Υπάρχουν συναρτήσεις που χρησιμοποιούνται για να μεταφέρουν τιμή private μεταβλητής εκτός κλάσης
- Συνήθως ονομάζονται

Get_<όνομα μεταβλητής>

Παράδειγμα

```
class Cat {
public:
        void SetAge (int age);
        int GetAge();
        void Meow();
private:
        int itsAge;
}; …
int Cat::GetAge()  {
    return itsAge;
 }
…
```

Πρόγραμμα/1

```
#include <iostream.h>
class Cat {
    public:
        int GetAge();
        void SetAge (int age);
        void Meow();
    private:
        int itsAge;
};
```

Πρόγραμμα/2

```
int Cat::GetAge()  {
   return itsAge;
}
void Cat::SetAge(int age) {
   itsAge = age;
}
void Cat::Meow() {
   cout << "Meow.\n";
}
```

Πρόγραμμα/3

```
int main() {
   Cat Frisky;
   Frisky.SetAge(5);
   Frisky.Meow();
   cout << "Frisky is a cat who is " ;
   cout << Frisky.GetAge() << " years old.\n";  Frisky.Meow();
   return 0;
}
```

Output

Meow.

Frisky is a cat who is 5 years old.

Meow.

Αρχικοποίηση κλάσης

- Οι classes περιλαμβάνουν μια ιδιαίτερη συνάρτηση-μέλος που ονομάζεται constructor (δομητής).
- Ο constructor έχει το ίδιο όνομα με την κλάση
- Μπορεί να παίρνει παράμετρο αν χρειάζεται αλλά δεν έχει τύπο επιστροφής, ούτε καν void

Constructor (Δομητής)

```
class Date
{
public:
        Date(int d, int m, int y);
        // . . .
};
Date::Date(int d, int m, int y)
: day(d),
  month(m),
  year(y)
{ }
```

Constructor (Δομητής)

- Μεγάλο πλεονέκτημα των ΔΟΜΗΤΩΝ είναι το ότι η ύπαρξή τους απομακρύνει την πιθανότητα μη αρχικοποιημένων μεταβλητών. Αν έχει δηλωθεί η Date(int, int, int)

  τότε η

  ```
  Date d;
  ```

  είναι λάθος

- Στις κλάσεις μπορεί να ορίζονται περισσότεροι από ένας δομητές αλλά μόνο ένας από αυτούς θα είναι ο προκαθορισμένος (default)

Constructor (Δομητής)

```
class Date
{
 public:
        Date();
        Date(int d, int m, int y);
        Date(String);
        // . . .   };
Date::Date(int d, int m, int y): day(d),   month(m),   year(y)
{ if (year == 0) year = current year;}
```

Παράδειγμα/1.1

```
#include <iostream>
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area() {return width*height;}
};
void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}
```

Παράδειγμα/1.2

```
int main () {
  Rectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```

Παράδειγμα/2.1

```
#include <iostream>
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area () {return width*height;}
};
void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}
```

Παράδειγμα/2.2

```
int main () {
  Rectangle rect, rectb;
  rect.set_values (3,4);
  rectb.set_values (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

Παράδειγμα/3.1

```
#include <iostream>
class Rectangle {
    int width, height;
  public:
    Rectangle (int,int);
    int area () {return (width*height);}
};
Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}
```

Παράδειγμα/3.2

```
int main () {
  Rectangle rect (3,4);
  Rectangle rectb (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

Constructor και destructor

- Αν δεν δηλώσουμε constructor ή/και destructor ο compiler χρησιμοποιεί προκαθορισμένους.
- Οι προκαθορισμένοι constructor και destructor δεν κάνουν τίποτα και δεν επιστρέφουν τίποτα, ισοδυναμούν με:

cat() & ~cat()

- Όταν δηλώσουμε ένα αντικείμενο κλάσης καλούμε τον constructor
- Αν ο constructor δεν παίρνει παραμέτρους, μπορούμε να γράψουμε:

Cat Frisky;

- Συνίσταται όταν δηλώνουμε constructor να δηλώνουμε και destructor

Constructor και destructor/1

```
#include <iostream.h>
class Cat {
    public:
        Cat(int initialAge);       // constructor
       ~Cat();                             // destructor
        int GetAge();
        void SetAge (int age);
        void Meow();
    private:
        int itsAge;
};
```

Constructor και destructor/2

```
Cat::Cat(int initialAge) {
itsAge = initialAge;}
Cat::~Cat(){}
int Cat::GetAge()  {
   return itsAge; }
void Cat::SetAge(int age) {
   itsAge = age; }
void Cat::Meow() {
   cout << "Meow.\n"; }
```

Constructor και destructor/3

```
int main() {
   Cat Frisky(5);
    Frisky.Meow();
    cout << "Frisky is a cat who is " ;
    cout << Frisky.GetAge() << " years old.\n";
    Frisky.Meow();
    Frisky.SetAge(7);
    cout << "Now Frisky is " ;
    cout << Frisky.GetAge() << " years old.\n";
    return 0;}
```

Overloading συναρτήσεις-μέλη /1

```
#include <iostream.h>
typedef unsigned short int USHORT;
enum BOOL { FALSE, TRUE};
class Rectangle {
public:
    Rectangle(USHORT width, USHORT height);
    ~Rectangle(){}
    void DrawShape() const;
    void DrawShape(USHORT aWidth, USHORT aHeight) const;
private:
    USHORT itsWidth;
    USHORT itsHeight;
     };
```

Overloading συναρτήσεις-μέλη /2

```
Rectangle::Rectangle(USHORT width, USHORT height)  {
    itsWidth = width;
    itsHeight = height;  }
void Rectangle::DrawShape() const  {
    DrawShape( itsWidth, itsHeight); }
void Rectangle::DrawShape(USHORT width, USHORT height) const {
    for (USHORT i = 0; i<height; i++)  {
                for (USHORT j = 0; j< width; j++) {
                            cout << "*"; }
               cout << "\n"; }
}
```

Overloading συναρτήσεις-μέλη /3

```
int main() {
    Rectangle theRect(30,5);
    cout << "DrawShape(): \n";
    theRect.DrawShape();
    cout << "\nDrawShape(40,2): \n";
    theRect.DrawShape(40,2);
    return 0;
}
```

Overloading συναρτήσεις-μέλη /έξοδος

DrawShape():

****

****

****

****

****

DrawShape(40,2):

****

****

Overloading προκαθορισμένους τελεστές

- Μπορούμε να κάνουμε overloading τους προκαθορισμένους τελεστές δηλώνοντας συναρτήσεις της μορφής

returnType Operator op (parameters)

- όπου, op ο τελεστής για overloading ή γράφοντας τον τέλεστη

void operator++ ()

Overloading ++/1

```
#include <iostream.h>
typedef unsigned short USHORT;
class Counter  {
public:
    Counter();      ~Counter(){}
    USHORT GetItsVal()const { return itsVal; }
    void SetItsVal(USHORT x) {itsVal = x; }
    void Increment() { ++itsVal; }
    void operator++ () { ++itsVal; }
private:
    USHORT itsVal; };
Counter::Counter(): itsVal(0)  {};
```

Overloading ++/2

```
int main() {
    Counter i;
    cout << "The value of i is " << i.GetItsVal() << endl;
    i.Increment();
    cout << "The value of i is " << i.GetItsVal() << endl;
    ++i;
    cout << "The value of i is " << i.GetItsVal() << endl;
    return 0;
}
```

```
The value of i is 0
The value of i is 1
The value of i is 2
```

Προκαθορισμένες Τιμές/1

```
#include <iostream.h>
typedef unsigned short int USHORT;
enum BOOL { FALSE, TRUE};
class Rectangle {
public:
   Rectangle(USHORT width, USHORT height);
    ~Rectangle(){}
   void DrawShape(USHORT aWidth, USHORT aHeight, BOOL UseCurrentVals = FALSE) const;
private:
   USHORT itsWidth, itsHeight; };
Rectangle::Rectangle(USHORT width, USHORT height): itsWidth(width), itsHeight(height) {}
```

Προκαθορισμένες Τιμές/2

```
void Rectangle::DrawShape(USHORT width, USHORT height, BOOL UseCurrentValue ) const  {
    int printWidth, printHeight;
    if (UseCurrentValue == TRUE) {
            printWidth = itsWidth;  printHeight = itsHeight; }
    else {
            printWidth = width;
            printHeight = height; }
    for (int i = 0; i<printHeight; i++) {
            for (int j = 0; j< printWidth; j++) {
                    cout << "*"; }
            cout << "\n"; }  }
```

Προκαθορισμένες Τιμές/3

```
int main()  {
   Rectangle theRect(30,5);
   cout << "DrawShape(0,0,TRUE)...\n";
   theRect.DrawShape(0,0,TRUE);
   cout <<"DrawShape(40,2)...\n";
   theRect.DrawShape(40,2);
   return 0;
}
```

Προκαθορισμένες Τιμές/έξοδος

DrawShape(0,0,TRUE)...

****

****

****

****

****

DrawShape(40,2)...

****

****

Overloading vs Default

Χρησιμοποιήστε Overloading συναρτήσεις αντί Προκαθορισμένες τιμές, όταν:

- Δεν υπάρχει λογική προκαθορισμένη τιμή
- Χρειάζεστε διαφορετικές εκδοχές των διαδικασιών (αλγορίθμων)
- Χρειάζεστε διαφορετικές παραμέτρους εισόδου

Default Constructor

- Αν δεν δηλώσουμε constructor, ο compiler χρησιμοποιεί ένα default constructor, χωρίς παραμέτρους, που δεν κάνει τίποτα.
- Όταν ορίσουμε έναν οποιοδήποτε constructor ο default του compiler παύει να υπάρχει.
- Για αυτό αν θέλουμε να υπάρχει και constructor χωρίς παραμέτρους θα πρέπει να τον ορίσουμε εμείς.
- Εξ’ορισμού ένας constructor χωρίς παραμέτρους, ονομάζεται default constructor.
- Ο default constructor που ορίζουμε μπορεί να λειτουργεί όπως επιθυμούμε.

Constructor overloading/1

```
#include <iostream.h>
class Rectangle {
public:
    Rectangle();
    Rectangle(int width, int length);
    ~Rectangle() {}
    int GetWidth() const { return itsWidth; }
    int GetLength() const { return itsLength; }
private:
    int itsWidth;
    int itsLength; };
```

Constructor overloading/2

```
Rectangle::Rectangle() { itsWidth = 5;  itsLength = 10;  }
Rectangle::Rectangle (int width, int length)  {
    itsWidth = width; itsLength = length;  }

int main()  {
    Rectangle Rect1;
    cout << "Rect1 width: " << Rect1.GetWidth() << endl;
    cout << "Rect1 length: " << Rect1.GetLength() << endl;
    int aWidth, aLength;
    cout << "Enter a width: ";
    cin >> aWidth;
```

Constructor overloading/3

```
 cout << "\nEnter a length: ";
   cin >> aLength;
   Rectangle Rect2(aWidth, aLength);
   cout << "\nRect2 width: " << Rect2.GetWidth() << endl;
   cout << "Rect2 length: " << Rect2.GetLength() << endl;
   return 0
}
```

Constructor overloading/output

Rect1 width: 5

Rect1 length: 10

Enter a width: 20

Enter a length: 50

Rect2 width: 20

Rect2 length: 50