

Κληρονομικότητα – Άσκηση 1

Δημιουργία της κλάσης Ήρωα (Hero)

- Δημιουργία Κλάσης
- Διαχωρισμός Δήλωσης από Υλοποίηση
- Private μέλη και μέθοδοι get, set
- Χρήση const
- Δημιουργία αντικειμένων
- Δυναμική διαχείριση μνήμης με new, delete

Άσκηση 1

- Δημιουργήστε μια κλάση με όνομα **Hero** που να αναπαριστά ήρωες. Κάθε ήρωας έχει όνομα, **level** και **life** ως private μεταβλητές. Δημιουργήστε αντίστοιχες public μεθόδους `get` και `set` για την προσπέλαση/τροποποίηση τους και μια μέθοδο που να εκτυπώνει τα στοιχεία ενός ήρωα.
- Στην `main` δημιουργήστε 3 αντικείμενα της Κλάσης:
 - Στο πρώτο δώστε τιμές άμεσα μέσω του δημιουργού.
 - Στο δεύτερο μὴν δώσετε τιμές ώστε να πάρει τις προκαθορισμένες που θα έχετε ορίσει στον δημιουργό
 - Στο τρίτο χρησιμοποιήστε δυναμική ανάθεση μνήμης

Άσκηση 1

hero.h

```
#ifndef HERO_H
#define HERO
class Hero{
public:
    Hero (char* ="Unknown", int =1, int =100);
    ~Hero();
    char* getName() const;
    int getLevel() const;
    int getLife() const;
    void print() const;
    void setName ( char* );
    void setLevel ( int );
    void setLife ( int );
private:
    char* Name;
    int Level;
    int Life;
};
#endif
```

Χρήση const για τις συναρτήσεις
προσπέλασης και εκτύπωσης

Δήλωση των μεταβλητών ως private και
προσπέλαση/τροποποίηση με τις
μεθόδους get/set

Άσκηση 1

hero.cpp

```
#include <iostream>
#include <iomanip>
using std::cout;
using std::endl;
using std::setw;
#include "hero.h"

Hero::Hero (char* N, int LVL, int L)
    :Name(N),Level(LVL),Life(L) {}

Hero::~Hero() { cout <<"Destruction of: " <<getName() <<endl; };

char*   Hero::getName() const { return Name; }
int     Hero::getLevel() const { return Level; }
int     Hero::getLife() const { return Life; }
void    Hero::setName( char* N)      { Name=N; }
void    Hero::setLevel( int LVL)     { Level=LVL; }
void    Hero::setLife( int L)        { Life=L; }

void    Hero::print() {
    cout <<setw(10)<<Name<<setw(5)
        <<Level<<setw(5)<<Life<<endl;
}
```

Άσκηση 1

```
#include "hero.h"
```

main.cpp

```
int main() {
```

```
    Hero hero1("Stan", 3, 80);  
    hero1.print();
```

```
    Hero hero2;  
    hero2.setName("John");  
    hero2.print();
```

```
    Hero* hero3ptr= new Hero("Peter", 0, 30);  
    hero3ptr ->print();
```

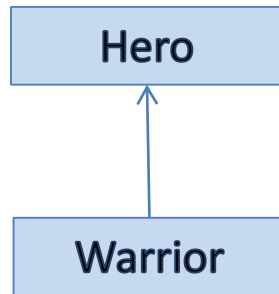
```
    delete(hero3ptr);  
    system("PAUSE");
```

```
return 0; }
```

Δυναμική ανάθεση
μνήμης με new και
delete

Κληρονομικότητα – Άσκηση 2

Δημιουργία της κλάσης Πολεμιστή (Warrior)



- Άμεση κληρονομικότητα
- Υπερφόρτωση (Method Overloading)
- Πέρασμα αντικειμένων κλάσης σε μέθοδο
- Χρήση της `static_cast`

Άσκηση 2

- Προσθέστε μια κλάση με όνομα **Warrior** που θα αναπαριστά Πολεμιστές. Ένας πολεμιστής κληρονομεί όλες τις ιδιότητες του ήρωα και έχει επιπλέον μια μεταβλητή *Armor* (θωράκιση) και μια μέθοδο *attack* με την οποία επιτίθεται σε άλλα αντικείμενα *Hero* ή *Warrior* αφαιρώντας μονάδες απο την ζωή τους. Στην περίπτωση που επιτίθεται σε *Warrior* οι μονάδες ζωής που αφαιρεί διαιρούνται με τις μονάδες θωράκισης:
- Η συνάρτηση *attack* θα οριστεί δυο φορές:
 - Στην πρώτη παίρνει ως όρισμα ένα αντικείμενο *Hero X* και έναν ακέραιο *D* και αφαιρεί απο την ζωή του ήρωα *X* το *D*.
 - Στην δεύτερη παίρνει ως όρισμα ένα αντικείμενο *Warrior X* και έναν ακέραιο *D* και αφαιρεί απο την ζωή του ήρωα *X* το *D* δια την θωράκιση *Armor* του *X*.
- Στην *main* δημιουργήστε αντικείμενα *Hero* και *Warrior* και δοκιμάστε να επιτεθείτε με έναν *Warrior* διαδοχικά σε έναν *Hero* και έναν *Warrior*.
- Παρατηρείστε οτι η ζημιά σε αντικείμενα *Warrior* είναι μειωμένη. Μπορείτε να εκμεταλευθείτε την συνάρτηση *static_cast* ώστε να παραλείψετε την θωράκιση ενός *Warrior*;

Άσκηση 2

```
#ifndef WARRIOR_H
#define WARRIOR_H
#include "hero.h"

class Warrior : public Hero {

public:
    Warrior (char* ="Unknown", int =1, int =100, int=20);

    void attack ( Hero*,int );
    void attack ( Warrior*,int );

    int getArmor() const ;
    void setArmor ( int );

private:
    int Armor; };

#endif
```

warrior.h

Άσκηση 2

warrior.cpp

```
#include "hero.h"
#include "warrior.h"
```

```
Warrior::Warrior(char* N, int LVL, int L, int A)
    :Hero(N,LVL,L),Armor(A) {}
```

Άμεση κλήση του δημιουργού της κλάσης βάσης

```
void Warrior::attack( Hero* X, int Damage){
    int new_X_life= X->getLife() -Damage;
    X->setLife(new_X_life);
    cout <<getName() <<" deals " << Damage<< " damage to hero "
    << X->getName() <<endl;
}
```

Πέρασμα αντικειμένων της κλάσης σε συνάρτηση

```
void Warrior::attack( Warrior* X, int Damage){
    int new_X_life= X->getLife() - (Damage/Armor);
    X->setLife(new_X_life);
    cout <<getName() <<" deals "<< Damage/Armor<< " damage to warrior "
    << X->getName() <<endl;
}
```

Δεν έχουμε άμεση πρόσβαση στις private μεταβλητές της κλάσης βάσης Hero. Θα μπορούσαμε να τις δηλώσουμε protected.

```
int Warrior::getArmor() const { return Armor; }
void Warrior::setArmor ( int A ){ Armor=A; }
```

Άσκηση 2

main.cpp

```
#include "hero.h"
#include "warrior.h"
int main() {

    Hero* herol= new Hero("Kostas",0,100);
    herol ->print();
    Warrior* warrior1= new Warrior("Mitsos",0,100,4);
    warrior1 ->print();
    Warrior* warrior2= new Warrior("Gianni",0,100,4);
    warrior2 ->print();

    warrior1 ->attack(herol,20);
    herol ->print();
    warrior1 ->attack(warrior2,20);
    warrior2 ->print();
    warrior1 ->attack(static_cast<Hero*>(warrior2),20);
    warrior2 ->print();

    ...
}
```

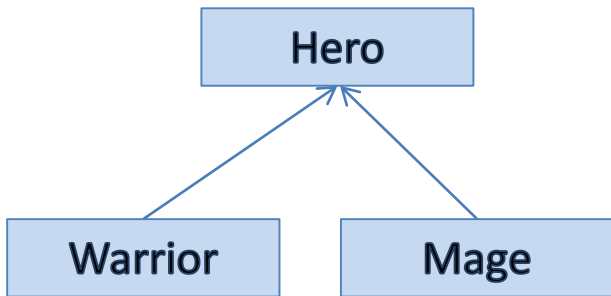
Οι δυο συναρτήσεις attack που ορίσαμε διαφέρουν μόνο στα ορίσματα και απο αυτά εξαρτάται ποια θα κληθεί

warrior2 : Δείκτης σε αντικείμενο Warrior
(static_cast<Hero*>(warrior2)) : Δείκτης σε αντικείμενο Hero

```
Kostas      0    100
Mitsos      0    100
Gianni      0    100
Mitsos deals 20 damage to hero Kostas
Kostas      0     80
Mitsos deals 5 damage to warrior Gianni
Gianni      0     95
Mitsos deals 20 damage to hero Gianni
Gianni      0     75
```

Κληρονομικότητα – Άσκηση 3

Δημιουργία της κλάσης Μάγου (Mage)



- Άμεση κληρονομικότητα
- Έλεγχος τιμής στις set μεθόδους
- Οι μέθοδοι που παίρνουν σαν παράμετρο αντικείμενα κλάσης, μπορούν να δεχτούν και αντικείμενα παραγόμενων κλάσεων αυτής

Άσκηση 3

- Προσθέστε μια κλάση με όνομα **Mage** που θα αναπαριστά Μάγους. Ένας μάγος κληρονομεί όλες τις ιδιότητες ενός Hero και διαθέτει επιπλέον μια μεταβλητή Mana (Ενέργεια) και μια συνάρτηση επίθεσης με μαγεία castSpell η οποία παίρνει ως ορίσματα έναν ήρωα X και έναν ακέραιο D και αν το D είναι μικρότερο του Mana τότε αφαιρεί από την ζωή του X το D. Επίσης αφαιρεί D από το δικό του Mana.
- Στην main δημιουργήστε αντικείμενο Mage και δοκιμάστε να καλέσετε από αυτό την συνάρτηση castSpell δίνοντας ως όρισμα αντικείμενο Hero και αντικείμενο Warrior.

Άσκηση 3

```
class Mage : public Hero {                                     mage.h
public:
    Mage (char* ="Unknown", int =1, int =100, int=100);

    void castSpell( Hero*,int );

    int getMana() const;
    void setMana ( int );

private:
    int Mana;
};
```

Άσκηση 3

```
Mage::Mage(char* N, int LVL, int L, int M)
    :Hero(N,LVL,L),Mana(M) {}

void Mage::castSpell( Hero* X, int Damage){
    if (Mana>Damage){
        Mana-=Damage;
        int new_X_life= X->getLife() -Damage;
        X->setLife(new_X_life);
        cout <<getName() <<" deals "
             << Damage << " damage to "
             << X->getName() <<endl;
    }
    else cout <<getName() <<" has not enough mana "<<endl;
}

int Mage::getMana() const { return Mana; }
void Mage::setMana ( int M ) { Mana=( M<0 ? 0 : M ); }
```

mage.cpp

Άμεση πρόσβαση
μόνο στο Mana

Έλεγχος της τιμής

Άσκηση 3

```
Hero* herol= new Hero("Kostas",0,30);
Warrior* warrior1= new Warrior("Mitsos",0,100,4);
Mage* mage1= new Mage ("Thanasis",0,80,100);

herol->print();
warrior1 ->print();
mage1 ->castSpell(herol,40);
herol ->print();
mage1 ->castSpell(warrior1,70);
mage1 ->castSpell(warrior1,50);
warrior1 ->print();
```

main.cpp

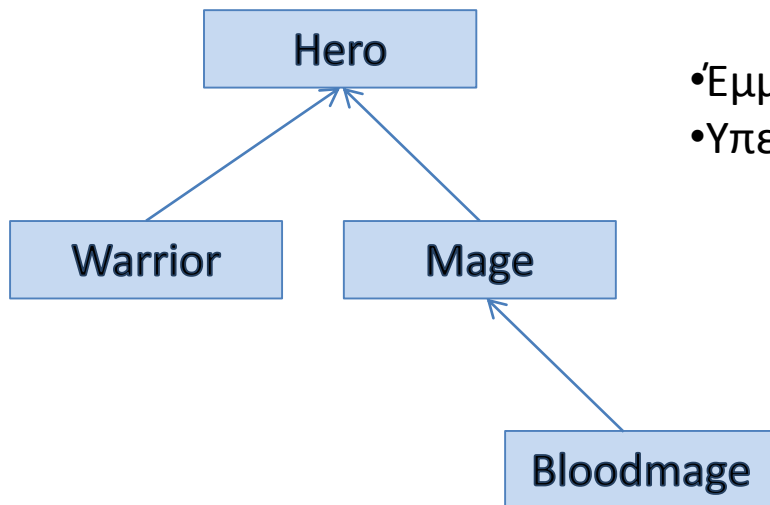
Η castSpell έχει οριστεί να παίρνει
όρισμα αντικείμενο Hero.

Μπορούμε να χρησιμοποιήσουμε σαν
όρισμα και αντικείμενα παραγόμενων
κλάσεων της Hero

```
Kostas      0    30
Mitsos      0   100
Thanasis deals 40 damage to Kostas
Kostas      0   -10
Thanasis has not enough mana
Thanasis deals 50 damage to Mitsos
Mitsos      0    50
```

Indirect Inheritance – Άσκηση 4

Δημιουργία της κλάσης Bloodmage



- Έμμεση Κληρονομικότητα
- Υπερ κάλυψη (Method Overriding)

Άσκηση 4

- Προσθέστε μια κλάση με όνομα **Bloodmage** η οποία κληρονομεί όλες τις ιδιότητες της κλάσης Mage αλλά **ορίζει την δική της συνάρτηση** castSpell με ίδια ορίσματα.
- Η castSpell ελέγχει αν το Mana είναι αρκετό ($\text{Mana} > \text{Damage}$).
 - Αν είναι αρκετό τότε αφαιρεί D απο την ζωή του X.
 - Αν δεν είναι αρκετό τότε πάλι αφαιρεί D απο τη ζωή του X αλλά αφαιρεί και απο τη δική του ζωή την διαφορά $D - M$
- Στην Main δημιουργήστε αντικείμενο τύπου Bloodmage και δοκιμάστε την συνάρτηση castSpell για διάφορες περιπτώσεις ώστε να βεβαιωθείτε οτι καλείτε η castSpell της κλάσης Bloodmage.
- Να καλέσετε απο αντικείμενο Bloodmage την castSpell της κλάσης βάσης Mage χρησιμοποιώντας την static_cast

Άσκηση 4

```
#include "Mage.h"
```

bloodmage.h

```
class Bloodmage : public Mage{  
    public:  
        Bloodmage (char* ="Unknown", int =1, int =100, int=100);  
  
        void castSpell( Hero*,int );  
};
```

Method Overriding:
Ξαναορίζουμε την μέθοδο
castSpell της κλάσης βάσης

Άσκηση 4

bloodmage.cpp

```
Bloodmage::Bloodmage(char* N, int LVL, int L, int M)
    :Mage(N,LVL,L,M) {}
```

```
void Bloodmage::castSpell( Hero* X, int Damage){
    if (getMana()>Damage){
        int newMana=getMana()-Damage;
        setMana(newMana);
    }
    else{ //Mana is not enough
        int Rest= Damage-getMana();
        setMana(0);
        int new_THIS_life= getLife() -Rest;
        setLife( new_THIS_life );
    }
    int new_X_life= X->getLife() -Damage;
    X->setLife(new_X_life);
    cout <<getName() <<" deals "<< Damage<< " damage to "
        << X->getName() <<endl;
}
```

Δεν έχουμε άμεση πρόσβαση σε καμία μεταβλητή αφού όλες είναι private μεταβλητές που έχουν κληρονομηθεί

Άσκηση 4

main.cpp

```
Hero* h1= new Hero("Kostas",0,30);
Warrior* w1= new Warrior("Mitsos",0,100,4);
Mage* m1= new Mage ("Thanasis",0,80,100);
Bloodmage* bm1= new Bloodmage ("Vrasid",0,80,100);
```

```
bm1->castSpell(h1,40);
```

```
Mage* mage_ptr= bm1;
mage_ptr ->castSpell(w1,120);
```

```
bm1->Mage::castSpell(w1,120);
```

```
static_cast<Mage>(* bm1).castSpell(w1,120);
```

```
static_cast<Mage*>(bm1)->castSpell(w1,120);
```

```
bm1->castSpell(w1,120);
```

```
delete(h1);
```

Κλήση απο αντικείμενο Bloodmage

Κλήση απο αντικείμενο Bloodmage

Δημιουργία δείκτη τύπου Mage και ανάθεση αντικειμένου Bloodmage
Κλήση απο mage

Κλήση της μεθόδου της Mage με χρήση του τελεστή Scope Resolution ::

Χρήση της static_cast για κλήση απο αντικείμενο τύπου Mage. Προσέξτε οτι δημιουργείται νέο αντικείμενο το οποίο μετά την παραπάνω κλήση διαγράφεται.
Αν η κλήση άλλαζε κάποιο στοιχείο του αντικειμένου που την καλεί δεν θα είχαμε το επιθυμητό αποτέλεσμα
Ο σωστός τρόπος είναι casting σε δείκτη αντικειμένου!

```
Kostas    0    30
Mitsos    0   100
Vrasid    0    80
Kostas    0    30
Vrasid deals 40 damage to Kostas
Kostas    0   -10
Vrasid has not enough mana
Vrasid has not enough mana
Vrasid has not enough mana
Destruction of: Vrasid
Vrasid has not enough mana
Mitsos    0   100
Vrasid deals 120 damage to Mitsos
Mitsos    0   -20
Vrasid    0    20
Destruction of: Kostas
```

Η castSpell έχει οριστεί στην Mage και την Bloodmage.
Ποια μέθοδος θα κληθεί εξαρτάται απο το είδος του αντικειμένου που την καλεί.

Indirect Inheritance – Άσκηση 5

Operator Overloading in Hero Class

- Υπερφόρτωση Τελεστών (Operator Overloading)

Άσκηση 5

- Στην κλάση Hero κάντε overload τους παρακάτω operators
 - Operator ++ Θα ανεβάζει το Level ενός Hero κατά 1
 - Operator – Θα αφαιρεί μια τιμή από το Life του Hero
 - Operator > Θα επιστρέφει true αν ο Hero έχει μεγαλύτερη τιμή Level από κάποιον άλλον

Άσκηση 5

hero.h

```
class Hero{  
    public:  
        Hero (char* ="Unknown", int =1, int =100) ;  
        ~Hero () ;  
  
        ...  
  
        void operator++ (int) ;  
        void operator- (int) ;  
        bool operator> (Hero) ;  
};
```

Άσκηση 5

hero.cpp

```
void Hero::operator++ (int) {
    Level++;
    cout<<Name<< " has gained a level!"<<endl;
    Life+=100;
}

void Hero::operator- (int Damage) {
    cout<<Name<< " is dealt "<<Damage<<" damage"<<endl;
    Life-=Damage;
}

bool Hero::operator> (Hero X) {
    if( Level > X.getLevel() )
        return true;
    else
        return false;
}
```


Άσκηση 5

main.cpp

```
h1++;           // Overloaded operator ++
h1.print();

h1 -20;         // Overloaded operator -
h1.print();

if(h1 >w1)       // Overloaded operator >
    cout<< h1.getName() <<" higher level than "
    << w1.getName()<<endl;

h1.operator-(20);

if(h1.operator>(w1) )
    cout<< h1.getName() <<" higher level than "
    << w1.getName()<<endl;
```

Ισοδύναμο με: h1 -20;

Ισοδύναμο με: if(h1 > w1)

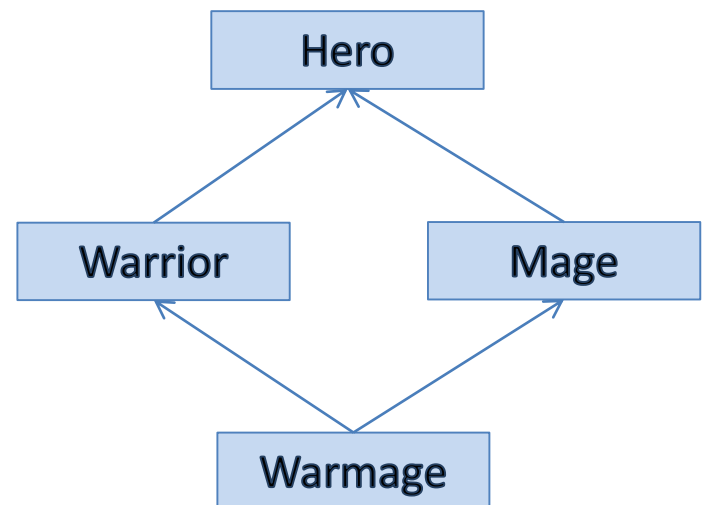
```
Dimitris    3    80
Giannis     1   100
Kostas      2    30
Kostas has gained a level!
Kostas      3   130
Kostas is dealt 20 damage
Kostas      3   110
Destruction of: Giannis
Kostas higher level than Giannis
Kostas is dealt 20 damage
Destruction of: Giannis
Kostas higher level than Giannis
```

Multiclassing – Άσκηση 6

Δημιουργία της κλάσης Warmage



- Πολλαπλή Κληρονομικότητα
- Diamond Problem
- Virtual Inheritance



Άσκηση 6

- Δημιουργείστε μια κλάση με όνομα **Warmage** που κληρονομεί τις κλάσεις Warrior και Mage (Multiclassing).
 - Δοκιμάστε στον δημιουργό της κλάσης Warmage να καλέσετε άμεσα τον δημιουργό της κλάσης Hero που κληρονομείτε έμμεσα.
- Στην Main δημιουργήστε αντικείμενο Warmage και δοκιμάστε να επιτεθείτε σε άλλα αντικείμενα με τις συναρτήσεις attack και castSpell
- Δοκιμάστε να εμφανίσετε τα στοιχεία ενός αντικειμένου Warmage μέσω της print συνάρτησης που είχατε δηλώσει στην Hero.
- Τι προβλήματα παρατηρήσατε και πως μπορούμε να τα αποφύγουμε.

Άσκηση 6

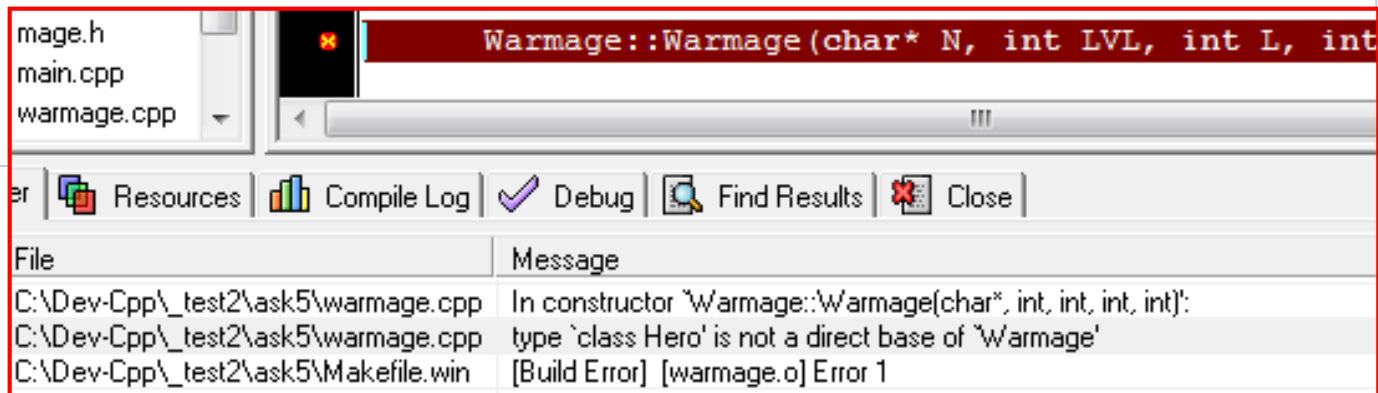
warmage.h

```
#include "Mage.h"
#include "Warrior.h"

class Warmage : public Warrior, public Mage{
public:
    Warmage (char* ="Unknown", int =1, int =100, int =2, int=100); };
```

warmage.cpp

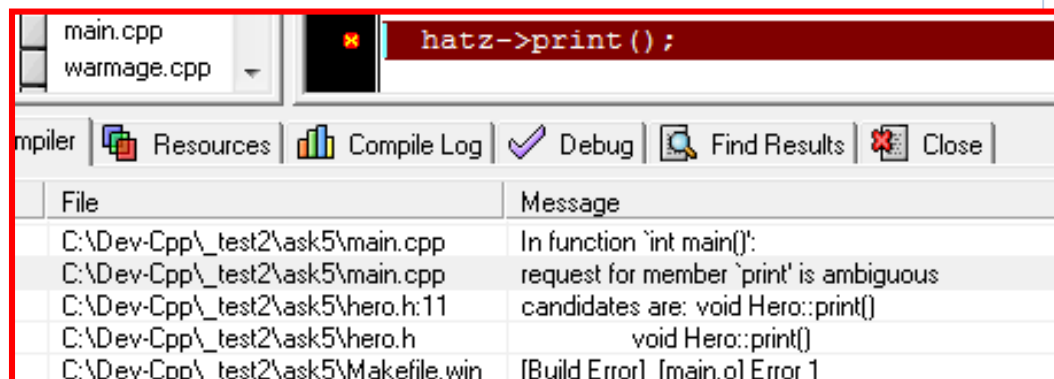
```
Warmage::Warmage(char* N, int LVL, int L, int A, int M)
:Hero(N,LVL,L) {
    setMana(M);
    setArmor(A);
}
```



Άσκηση 6

```
Hero* h1= new Hero("Kostas",0,30);
h1->print();
Warrior* w1= new Warrior("Gianni",0,100,4);
w1->print();
Mage* m1= new Mage ("Thanos",0,80,100);
m1->print();
Warmage* wm1= new Warmage ("Hatzis",5,120,4,180);
wm1 ->print();
wm1 ->attack(h1,10);
wm1 ->attack(m1,10);
wm1 ->attack(w1,10);
wm1 ->castSpell(w1,50);
h1->print();
w1->print();
m1->print();
```

main.cpp



Άσκηση 6

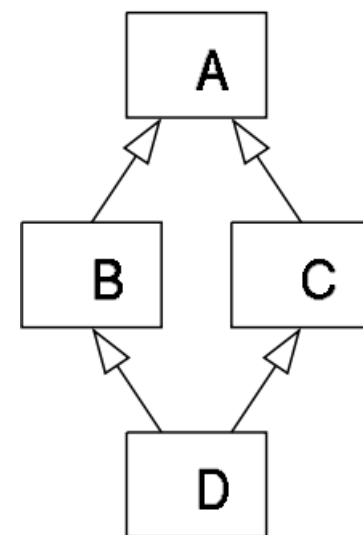
- 1^ο Πρόβλημα: Δεν μπορούμε να καλέσουμε τον δημιουργό έμμεσα κληροδοτούμενης κλάσης απευθείας
- 2^ο Πρόβλημα: Η κλάση Warmage κληρονομεί 2 φορές την print, μία λόγω της Mage και μία λόγω της Warrior. Ο compiler δεν γνωρίζει ποια απο τις δύο να καλέσει και αναφέρει error λόγω ambiguity. Το πρόβλημα είναι γνωστό ως Diamond problem.
- Λύση και για τα δυο προβλήματα είναι να ορίσουμε την κληρονομικότητα των κλάσεων Warrior και Mage ως virtual (Virtual Inheritance). Με αυτό τον τρόπο δηλώνεται μόνο μια φορά η κλάση Hero για την κλάση Warmage.

warrior.h

```
class Warrior : public virtual Hero {
```

mage.h

```
class Mage : public virtual Hero {
```



Προσοχή: η έννοια του Virtual Inheritance (χρήση virtual δήλωσης στην κληρονομικότητα) δεν σχετίζεται άμεσα με την έννοια του Πολυμορφισμού όπου ορίζουμε virtual μεθόδους.