Puntatori

Classi e struct

* Differenza

Operators overloading

Binding dinamico (Lezione 4)

* Metto “virtual” al metodo che voglio abilitare al binding dinamico
* Specifico “override” nella sottoclasse che lo overrida void f() override{}
* Facendo Classemadre::f() si richiama la funzione non sovrascritta della classe madre

Distruttore ~

* Se è una subclass dopo aver chiamato il proprio chiama anche quello della classe madre

Costruttore

* Oggetto.var =1
* Oggetto.var2=3;
* Posso scriverlo come
* Oggetto={1,3}

Puntatori (l1)

Reference (l2)

* Si passa   
  TipoOggetto\* a = new TipoOggetto();  
  function(a);  
  a  
  void function(TipoOggetto& temp){}

Classi e struct (+ differenza) (l2 e l3)

* Class tutto private (struct public)

Operators overloading (l3 e l4)

* Float2D operator-(Float2D other){} float2d – float2d
* MyInt operator++() myint++

Ereditarietà:

* Per usare i metodi della superclasse si deve inserire public:  
  class sottoclasse : public classemadre{};

Poi istanziarli via puntatore:  
classemadre\* a = new sottoclasse();  
E richiamare il metodo via:  
a->metodoMadre();

Binding dinamico (Lezione 4)

* Metto “virtual” al metodo che voglio abilitare al binding dinamico
* Specifico “override” nella sottoclasse che lo overrida void f() override{}
* Facendo Classemadre::f() si richiama la funzione non sovrascritta della classe madre

CopyConstructors (l4)

* Arma(const Arma& a){}
* Si richiama con  
  Arma\* a = new Arma();  
  Arma\* b = new Arma(\*a);

CopyAssignment

* Personaggio a= new Personaggio();  
  Personaggio b = a;
* Personaggio& operator=(const Personaggio& pg)

MoveConstructors (l5)

* Per classi:  
  Personaggio(Personaggio&& pg){}
* Per sottoclassi:

Richiamo il move constructor di Personaggio, ma devo castare g

che gia' dalla initialization list risulta essere un lvalue, in un rvalue

se passassi soltanto Personaggio(g) verrebbe richiamato il COPY Constructor

di Personaggio invece del suo MOVE Constructor!

Questo cast viene fatto dalla funzione std::move()

Guerriero(Guerriero&& g) : Personaggio( std::move( g ) ){}

MoveAssignment

* E' come il MOVE constructor ma PRIMA di prendere le risorse altrui

si devono liberare le risorse proprie dell'oggetto

* Personaggio& operator=(Personaggio&& pg){}

Strings (l5)

Functors (l5)

* sono oggetti che si richiamano come fossero funzioni

sfruttando l'overload dell'operatore ()

* class LessThan{  
  bool operator()(const int &x){}  
  };
* VANNO ISTANZIATI
* Si posso usare anche i puntatori su di loro

SmartPointers (l5)

* Unique  
  unique\_ptr<Guerriero> pinotto (new Guerriero);

amilcare.reset(); // Un delete esplicito di un unique\_ptr  
  
void perPassareUnoUniquePointer( unique\_ptr<Guerriero>& byReference );

* Shared  
  hanno un contatore che capisce quanti shared pointers puntano all’oggetto  
  shared\_ptr<int> lol;

foo = bar;// copy

lol = std::move(bar);// move

Containers (l6)

* VECTORS  
  sono tipo arraylist e sono random access  
    
  std::vector<NomeClasse> = {{nome, cognome},{nome2, cognome2} };

auto it = vec.begin(); //restituisce l’iteratore in posizione 0

* LIST  
  sono bidirectional, quindi necessitano degli iterator  
  std::list<int> listaInt = {9, 2, 5}
* MAP  
  (sono ordinate come alberi binari, le unordered map usano hash e bucket)  
  Std::map<string, int> mappa { {“Primo”, 1}, {“Secondo”, 2}

Ostreams (l7)

* OUTPUT FILE  
  Ofstream ofs(“nomefile\_da\_scrivere.txt”, ios::app);
* INPUT FILE  
  Ifstream ifs(“nomefile.txt”, ios::in);

Exceptions (l7)

* Try{

Throw variabile; //Lo manda nel catch

//throw “a”;

//throw;

}catch(int variabile){cout<<variabile;}

Catch(char e){}

Catch(…){cout<<”Eccezione generica”;}

Templates (l7)

* Template <typename T>   
    
  T GetMin(T a, U b){ //Fare attenzione, ritornerà una variabile del tipo del primo parametro passato

T result;

result = (a<b)? a: b;

return result;

}

Libraries (l7)