Universitat de Barcelona

Tema 2 **TADS i C++**

Dra. Maria Salamó Llorente

Estructura de Dades

Grau d'Enginyeria Informàtica

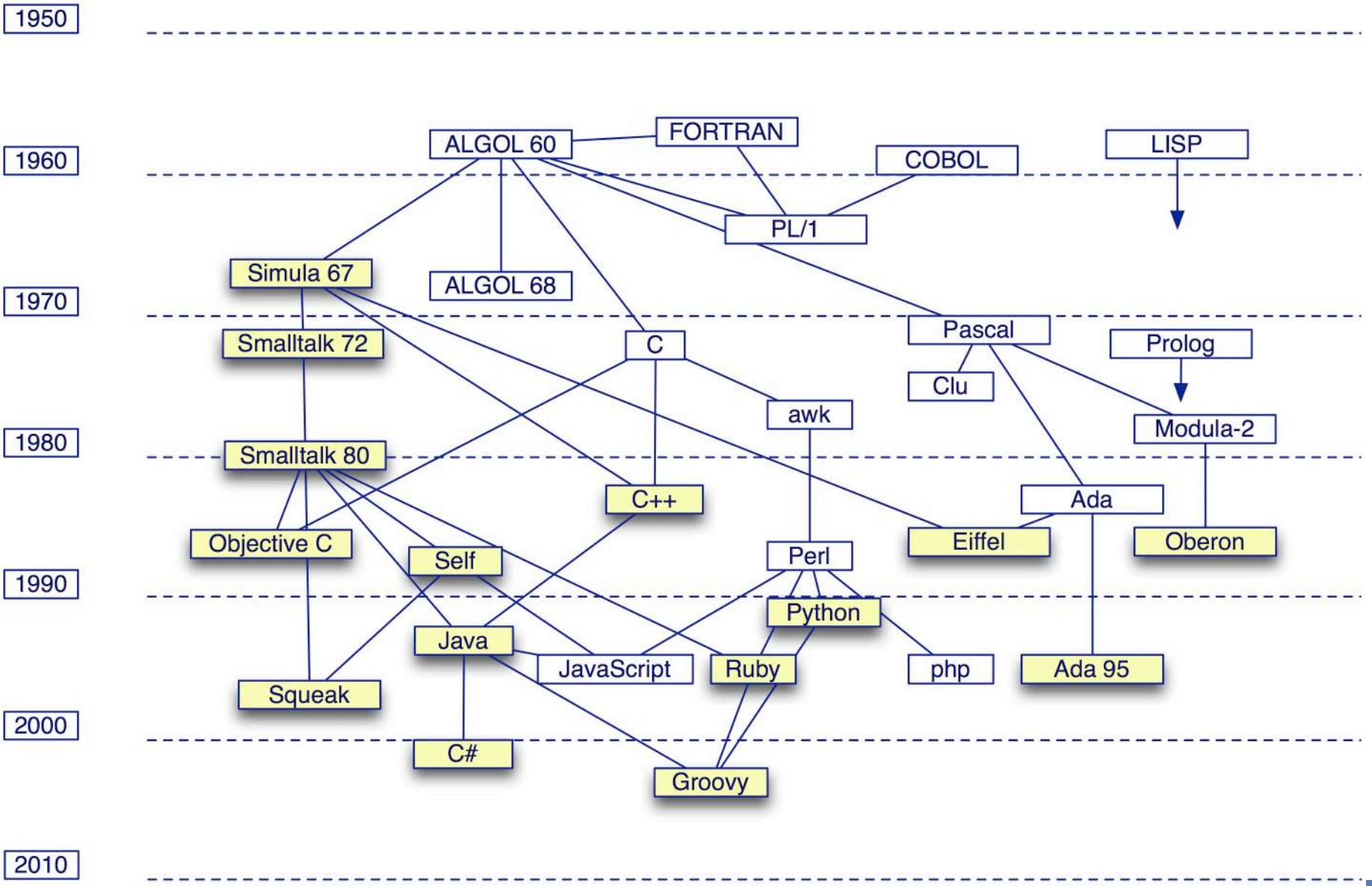
Facultat de Matemàtiques i Informàtica,
Universitat de Barcelona

Universitat de Barcelona

Contingut

2.1 Introducció bàsica al llenguatge C++

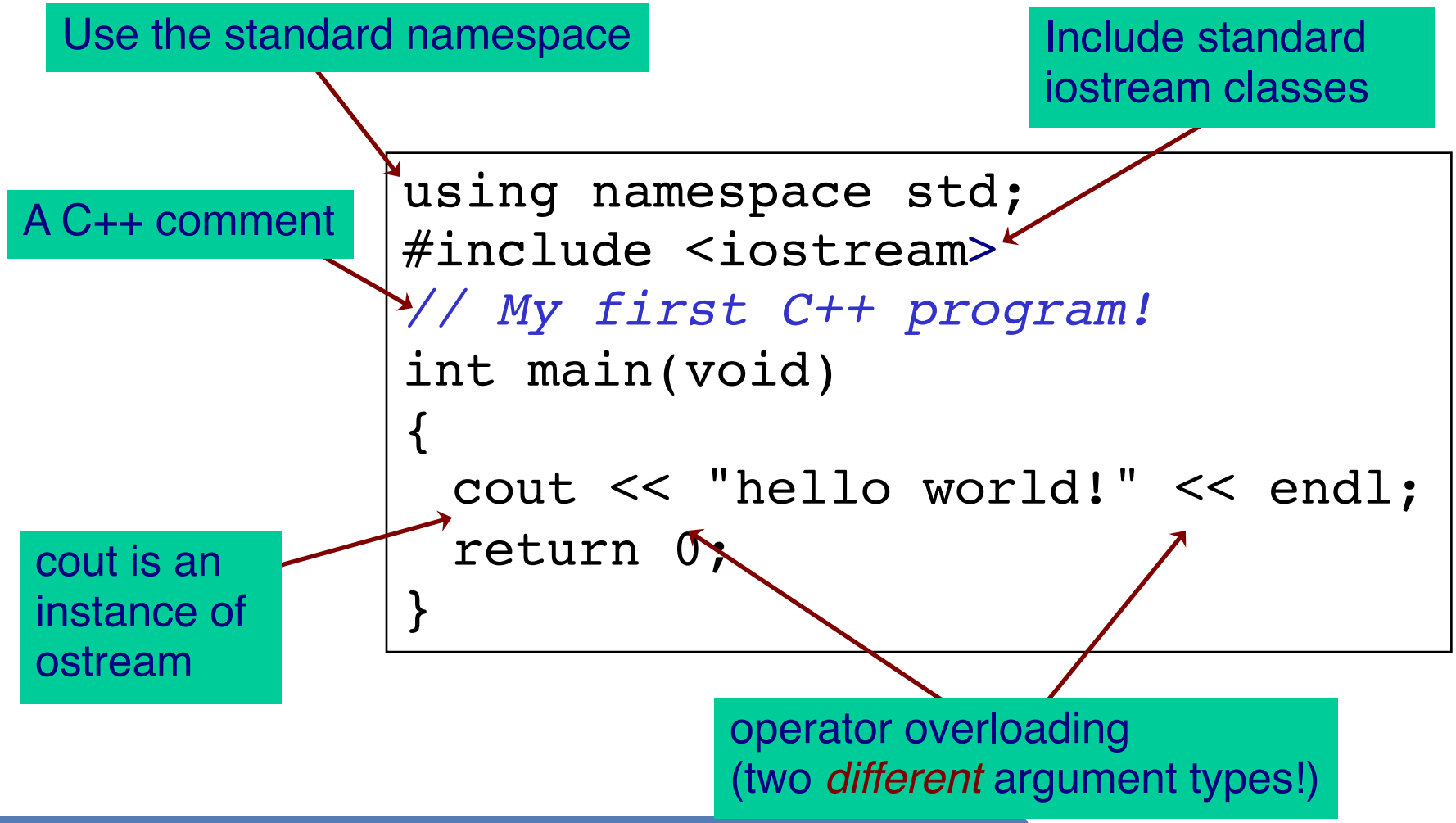
Genealogía de la Orientació a Objectes



"Hello World" en Java

```
package p2;
// My first Java program!
public class HelloMain {
    public static void main(String[] args) {
       System.out.println("hello world!");
       return 0;
    }
}
```

"Hello World" en C++



Compilar

Es pot compilar tot manualment:

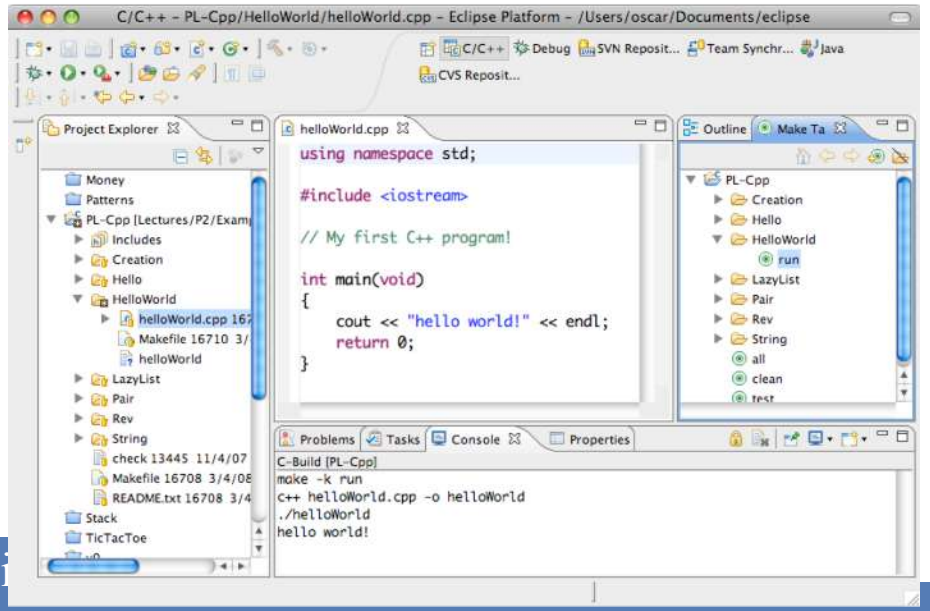
```
c++ helloWorld.cpp -o helloWorld
```

O es pot usar un *Makefile* per gestionar les dependències:

```
helloWorld : helloWorld.cpp
    c++ $@.cpp -o $@
```

```
make helloWorld
```

O es pot usar un IDE com *NetBeans o Eclipse* per crear un projecte i compilar el codi



Java i C++ — Similituds i extensions

Similituds:

- tipus de dades primitius (en Java, són independents de la plataforma)
- sintaxi: estructures de controls, excepcions ...
- classes, declaracions de visibilitat (public, private)
- multiples constructors, this, new
- tipus, casting de tipus (segur en Java, no en C++)
- comentaris

Algunes Extensions Java:

- garbage collector
- standard abstract machine
- standard classes
- packages (ara C++ té namespaces)
- final classes
- autoboxing
- Genèrics enlloc de templates

Simplificacions de Java del C++

- no punters — **només referències**
- no funcions — pots declarar mètodes **`static`**
- no variables globals — usa **`public static`** variables
- no destructors — **garbage collection** i **`finalize`**
- no linking — dynamic class loading
- no fitxers header — es poden definir **`interface`**
- no operadors de sobrecarrega — només es sobrecarrega mètodes
- no permeten llistes d'inicialització dels membres — crida al **`super`** constructor
- no preprocessor — **`static final`** **constants** i automatic inlining
- no herència múltiple — **implementa multiple interfaces**
- no structs, unions, enums — **normalment no són necessaris**

Noves paraules clau

Les paraules clau heretades de C, que C++ afegeix:

Exceptions	`catch, throw, try`
Declarations:	`bool, class, enum, explicit, export, friend, inline, mutable, namespace, operator, private, protected, public, template, typename, using, virtual, volatile, wchar_t`
Expressions:	`and, and_eq, bitand, bitor, compl, const_cast, delete, dynamic_cast, false, new, not, not_eq, or, or_eq, reinterpret_cast, static_cast, this, true, typeid, xor, xor_eq`

(vegeu** http://www.glenmccl.com/glos.htm**)

Un programa bàsic en C++

```
//include headers;  mòduls que inclouen funcions que feu servir
   al vostre codi
#include <iostream>
using namespace std;
int main() {
//declaració de variables
   string value;
//llegir valors d'entrada
   cin >> value;
//càlcul i imprimir sortida
   cout << " Hello World !!! " << value <<  endl;
return 0;
}
```

Després d'escriure el codi en C++ s'ha de compilar; és a dir, executes un programa que es diu **compilador** que comprova que se segueix la sintaxi de C++

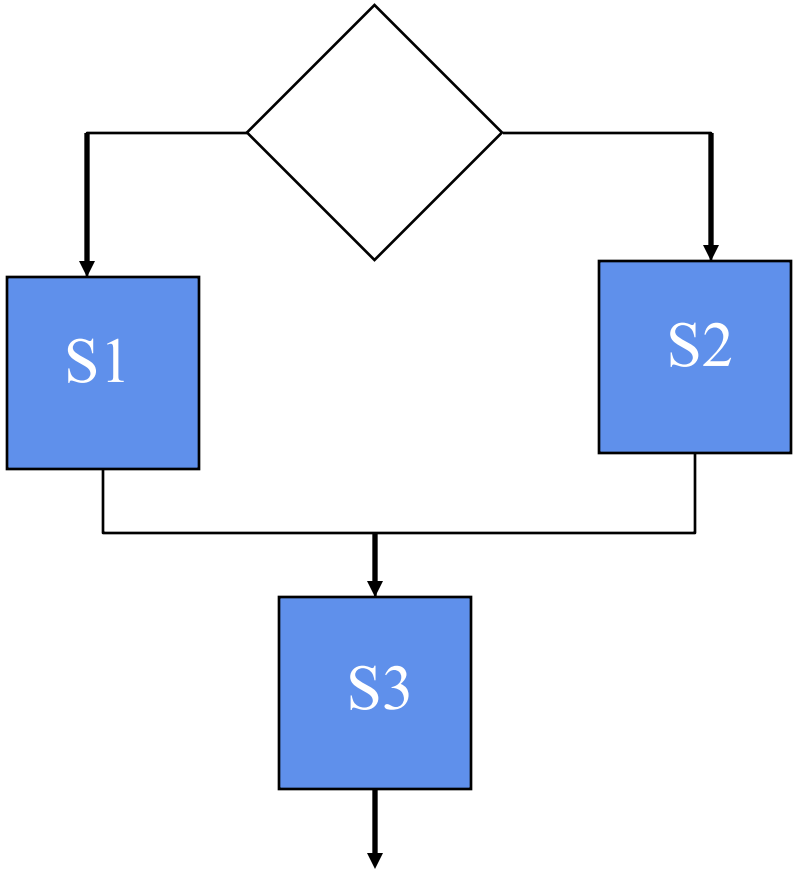
- Si hi ha errors, els llista
- Si no hi ha errors, tradueix el programa en C++ a un programa en codi màquina que es pugui executar

Notes

- El que vingui a continuació del **//** en la mateixa línia és un comentari
- Identació depèn del lector;
  - el compilador ignora tots els espais en blanc i els salts de línia, el delimitador per al compilador és el **;**
- Totes les instruccions finalitzen amb punt i coma
- **Majúscules i minúscules importen !!**
  - Void és diferent de void
  - Main és diferent de main

If statements

```
if (condicio_booleana) {
   S1;
}
else {
   S2;
}
S3;
```



Condicions booleanes

..es realitzen usant

- Operadors de comparació

  == igual

  != no igual

  < menor que

  > més gran que

  <= menor o igual que

  >= més gran o igual que

- Boolean operators

  && and

  || or

  ! not

Exemples

Assumeix que declarem les següents variables:

```
int a = 2, b=5, c=10;
```

Exemples de condicions booleanes:

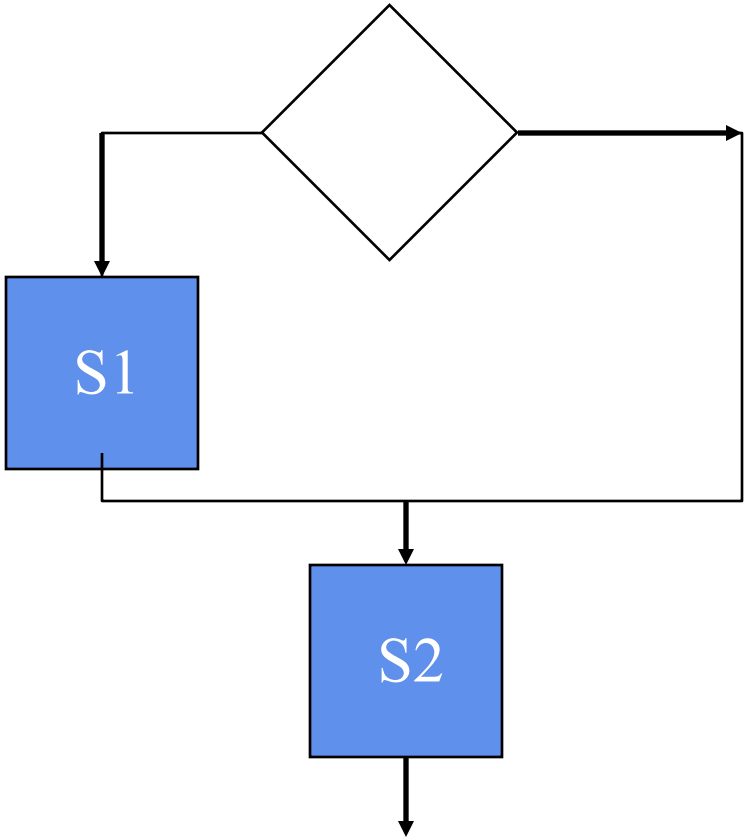
- `if (a == b) ...`
- `if (a != b) ...`
- `if (a <= b+c) ...`
- `If (a <= b && b <= c) ...`
- `if (!(a < b) && (b<c)) ...`

Exemple If

```
#include <iostream>
using namespace std;
int main() {
int a,b,c;
cin >> a >> b >> c;

if (a <=b) {  cout << "min is " << a << endl;
   }
else { cout << " min is " << b << endl;
}
cout << "happy now?" << endl;
return 0;
}
```

While

```
while (condicio) {
  S1;
}
S2;
```

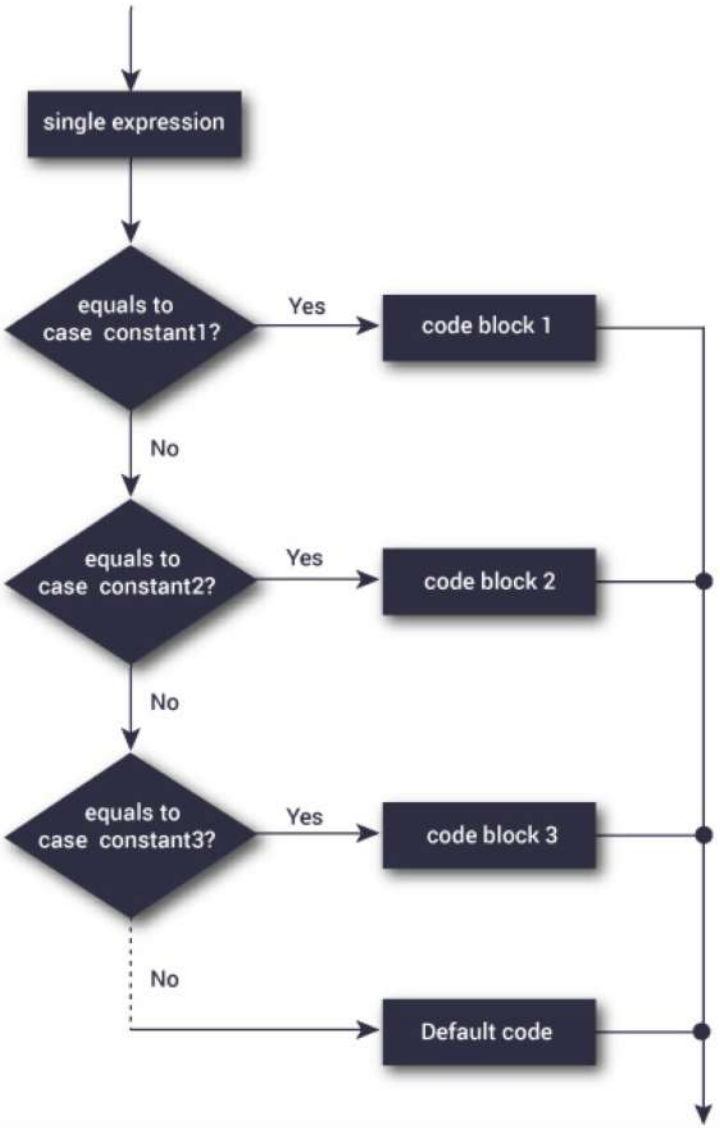


Exemple While

```
#include <iostream>
using namespace std;

int main() {
int i, sum, x;
sum=0;
i=1;
while (i <= 5) {
   cin >> x;
   sum = sum + x;
   i = i+1;
}
cout << "sum is" << sum << endl;
}
```

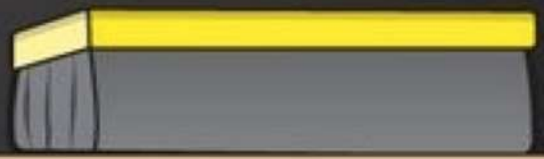
Switch

```
switch(expression){
case constant-expression :
        S1;  // code block 1
        break;
case constant-expression :
        S2;  // code block 2
        break;
// afegir tots els casos
default : //Opcional
        S3;  // default code
}
```



Exemple Switch

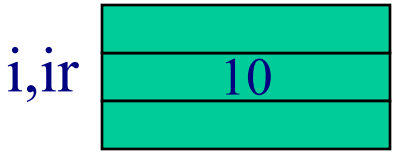
```
#include <iostream>
using namespace std;
int main () {
char grade = 'D';
switch(grade) {
    case 'A' : cout << "Excellent!" << endl; break;
    case 'B' : break;
    case 'C' : cout << "Well done" << endl; break;
    case 'D' : cout << "You passed" << endl; break;
    case 'F' : cout << "Better try again" << endl; break;
    default :
    cout << "Invalid grade" << endl;  break;
}
    cout << "Your grade is" << grade;
    return 0;
}
```



Referències en C++

Una referència és un ***alias*** per una altra variable:

```
int i = 10;
int &ir = i;   // referència (alias)
ir = ir + 1;   // incrementa i
```



Un cop inicialitzades, les referències no es poden canviar

Referències en C++

- **Què és una referència?**
  - *Un nom alternatiu (alias) per un OBJECTE*

GRAN diferència amb Java

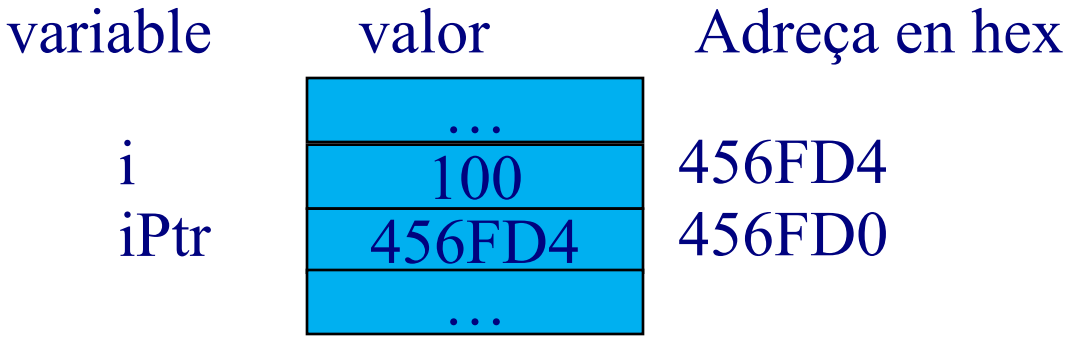
- Les referències només es creen en declaracions i paràmetres
- Una referència només pot aparèixer quan l'objecte ha pogut aparèixer

Referències simples

```
void f() {
  int a = 1;
  int &r = a;  //r i a referencien al mateix int
  int x = r;   // x ara val 1

  r = 2;       // a ara val 2
}

int k;
int &r1 = k;// OK: r1 s'inicialitza
int &r2;     // ERROR: no s'inicialitza
```

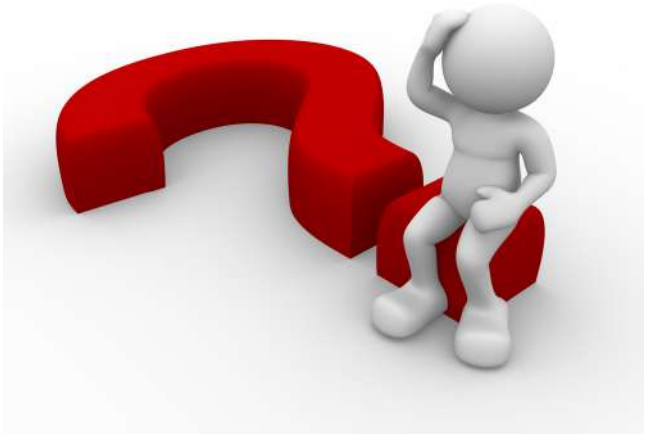
Punters en C++

```
int i;
int *iPtr; // un punter a un integer

iPtr = &i; // iPtr conté l'adreça de i
*iPtr = 100;
```



Referències simples

```
void g() {
  int ii = 0;
  int &rr = ii;
  rr++;

  int *pp = &rr;
}
```

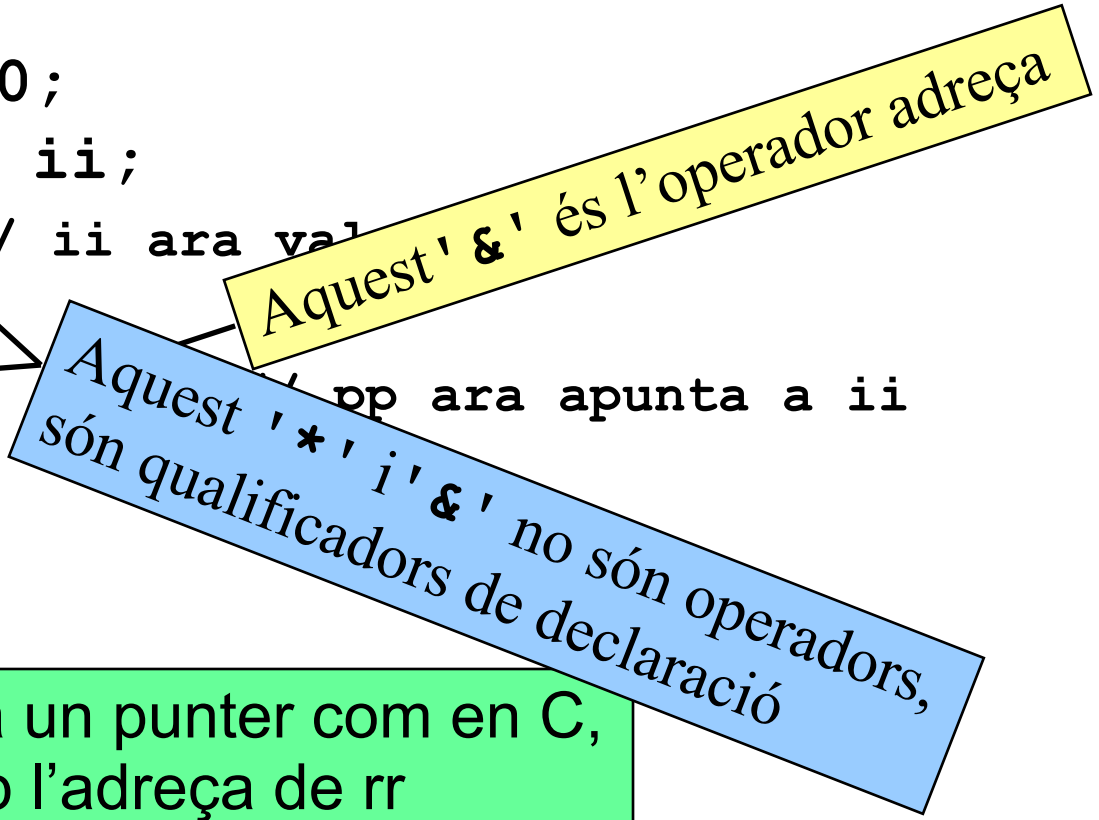


Pensem un moment …

Què creus que valen cadascuna de les variables?

Referències simples

```
void g() {
  int ii = 0;
  int &rr = ii;
  rr++;     // ii ara val
  int *pp =  pp ara apunta a ii
} //g
```



Nota: Es declara un punter com en C, i s'inicialitza amb l'adreça de rr (la qual és un altre nom de ii).

Exemple

Referència	Punter

int a = 10;
int b = 20;
int &c = a;
c = b;
	int a = 10;
	int b = 20;
	int *c = &a;
	c = &b;

Quin és el valor d'a?	Quin és el valor d'a?

Paràmetres per referència

- Es defineix un *alies* per l'argument de la funció que es crida
  - & situat després del tipus del paràmetre en el prototipus de la funció i en la capçalera de la funció
- Exemple
  - int &count a la capçalera d'una funció
    - Es diu que "count és una referència a un int"
- El nom del paràmetre a la funció que s'ha cridat és una referència a la variable original de la funció que ha fet la crida

Exemple de paràmetres per referència

```
#include <iostream>
using namespace std;
void swap (int &a , int & b){
   int temp = a;
   a = b;
   b = temp;
}
int main() {
   int a, b;
   cin >> a >> b;
   swap (a, b);
   cout << "A -> " << a << " B-> "  << b <<  endl;
return 0;
}
```

Referències Java *vs.* C++

- En *Java*, una referència és un tipus de dades
  - Es pot assignar, comparar, copiar, guardar, etc.
  - La mateixa referència pot referir-se a objectes diferents en moments diferents durant l'execució

- En *C++*, una referència és un *alies* per un objecte
  - No és pot assignar; l'assignació és a traves de la referència a l'objecte
  - La referència SEMPRE es refereix al mateix objecte mentre estigui actiu l'objecte.

Repeteix tres vegades

Una referència no és un punter, ...

Una referència no és un punter, ...

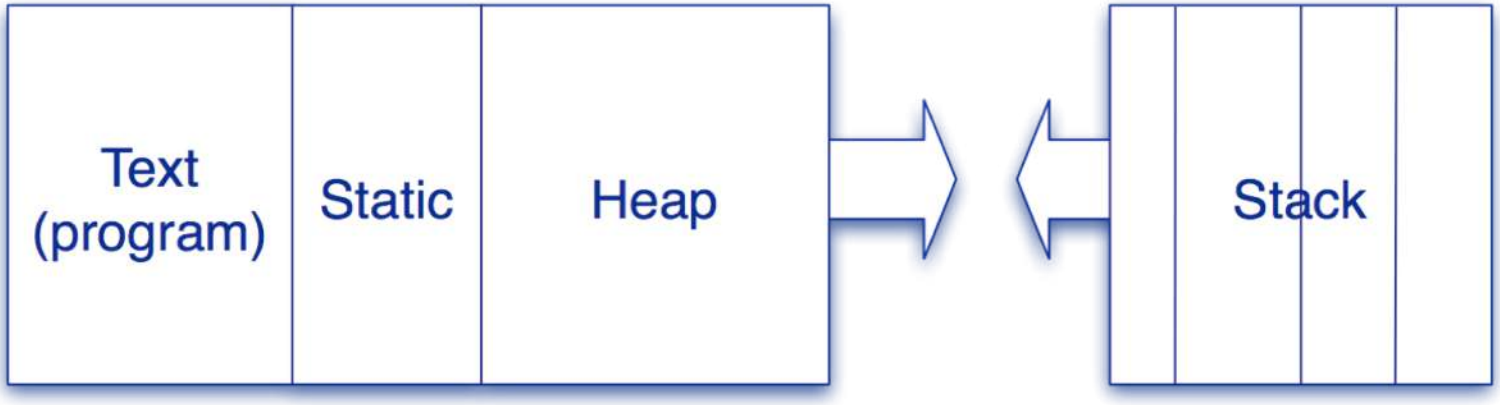
Una referència no és un punter, ...

I cap d'ells s'assembla a una referència de JAVA

Organització de la Memòria

L'espai d'adreces consisteix en (com a mínim):

Text:	Programa executable (no és pot sobreescriure)
Static:	Dades estàtiques
Heap:	Memòria global situada dinàmicament
Stack:	Memòria local per crides a funcions



2.2 Classes i objectes en C++

Definició de classes en C++

```
class classIdentifier{
   class members list
};
```

EXEMPLE

```
class Human
{
   // les dades són privades a les instàncies de la classe
   int height;
   char name[];
   int weight;

public:
   void  setHeight(int heightValue);
   int   getHeight();
};
```

La classe es defineix en Human.h

Definició de les funcions d'una classe

```
void Human::setHeight(int heightValue)
{
   if (heightValue > 0)
    height = heightValue;
   else
    height = 0;
}

int Human::getHeight()
{
   return(height);
}
```

La implementació
es defineix en el Human.cpp

Exemple d'ús

```
void main()
{    // first we define the variables
     int height = 72;
     int result = 0;

     Human hank;

     //set our human's height
     hank.setHeight(height);

     //get his height
     result = hank.getHeight();
     cout << "Hank is = " << result << "inches tall" << endl;
}
```

Sortida del programa

```
Hank is 72 inches tall
```

← Codi corresponent a main.cpp

Instanciant objectes

- La definició d'una classe NO crea cap objecte
- Com s'instancia un objecte?

```
className classObjectName;
```

- La instanciació es fa amb un **constructor**
  - Si la classe no té un constructor definit, llavors el compilador de C++ crea un constructor per defecte automàticament
  - El constructor per defecte no rep valors cap a les dades membres (i.e. les variables de la instància)
  - Instanciant un objecte amb un constructor amb paràmetres

```
className classObjectName(argument1, argument2, ...);
```

Instanciant objectes

- Quan es crea una instància, l'objecte es guarda a memòria
- Exemples:

```
Car myCar;

Elephant oneElephant, twoElephant;
```

- En cap dels objectes s'inicialitzen les dades membres (atributs)
- Cada objecte té la seva pròpia localització a memòria
  - `oneElephant` i `twoElephant` són objectes en diferents localitzacions de memòria
  - Cadascun es pot accedir per nom o localització

```
classObjectName.memberName
```

  - Cada atribut es pot accedir individualment usant el nom de l'objecte i el nom de l'atribut, per exemple:

```
oneElephant.age

twoElephant.name
```

Constructors i destructors

Una instància es pot crear *automàticament* (a l'stack):

```
MyClass oVal;     // es crida al constructor
                  // es destrueix quan finalitza el seu abast
```

o *dinàmicament* (en el heap)

```
MyClass *oPtr;          // punter no inicialitzat
oPtr = new MyClass;     // es crida al constructor
                        // s'ha d'esborrar explícitament
```

Constructors i destructors

```
#include <iostream>
using namespace std;
class Line{
public:
      void setLength( double len );
       double getLength( void );
       Line(double len);  // constructor
       Line (void);       // constructor per defecte
       ~Line(void);       // destructor
private:
    double length;
};
```

Inclou standard iostream

```
Line::Line( double len){
  cout << "Object is being created, length = " << len << endl;
  length = len;
}
Line::Line(void){ cout << "Object is being created" << endl;
}
Line::~Line(void){ cout << "Object is being deleted" << endl;
}

void Line::setLength( double len ){length = len;}
double Line::getLength( void ){ return length;}
```

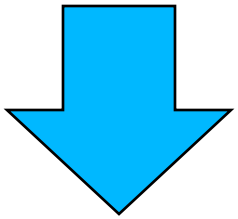
Constructors i destructors

```
int main( ){
    Line line(10.0);
    cout << "Length of line : " << line.getLength() <<endl;
    line.setLength(6.0);
    cout << "Length of line : " << line.getLength() <<endl;
    Line line2;
    line2.setLength(6.0);
    cout << "Length of line : " << line2.getLength() <<endl;
   return 0;
}
```

2.3 Abstracció

Abstracció

- El procés d'abstracció consisteix en separar els detalls del disseny i la implementació d'un codi del seu ús
- **L'abstracció de les dades** és el procés de separar les propietats lògiques de les dades de la seva implementació



Definim Tipus Abstractes de Dades

Tipus Abstractes de Dades

- Permeten separar les pròpietats lògiques de les dades dels detalls de la seva implementació
- Tot TAD està format per:
  - **Nom del tipus**
  - **Domini**, valors que pertanyen al TAD
  - **Operacions** associades amb les dades del TAD
- Per implementar un TAD en C++ es pot fer de dues maneres:
  - Structs
  - Classes → **En aquest curs usarem classes!!**

2.4 Encapsulament

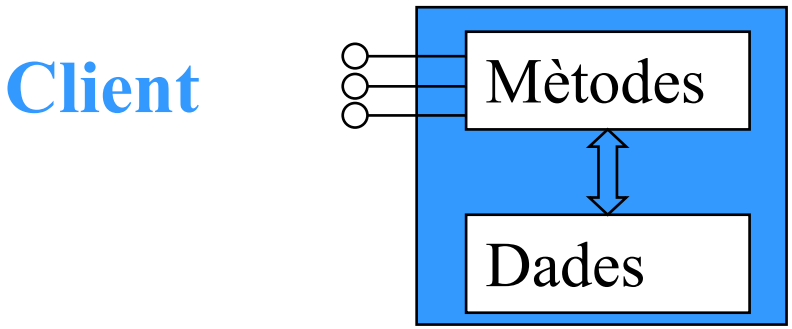
Encapsulament

Modificadors de visibilitat (accessibilitat)

	public	private
Variables	Trenca encapsulament	Força encapsulament
Mètodes	Serveis a clients	Suport a altres mètodes de la classe

Com es referencia a un objecte

- Cada objecte té un nom (o una localització), que s'assigna quan l'objecte s'instancia
- Dades **private** només són accessibles dins de la classe
  - Es necessari una funció membre pel seu accés
  - `myElephant.age = 72;` //no funciona si assumim `age` està declarat com a private
  - `myElephant.setAge(72);` // sí funciona



Visibilitat

- Existeixen 3 visibilitats: **`private`**, **`public`**, i **`protected`**
- **`private`**, **`public`**, i **`protected`** són paraules reservades en C++
- Per defecte, tots els membres (dades i mètodes) són **`private`**
- Tot membre **`private`** no és accessible des de fora de la classe
- Un membre **`public`** si és accessible des de fora de la classe
- Un membre **`protected`** és només accessible a les classes derivades

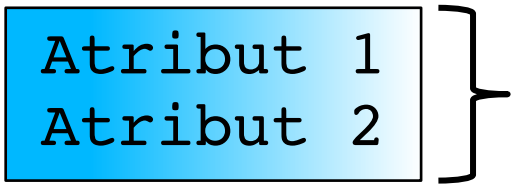
Visibilitat	private	protected	public
la mateixa classe	SI	SI	SI
classe derivada	NO	SI	SI
fora de la classe derivada	NO	NO	SI

2.5 Herència

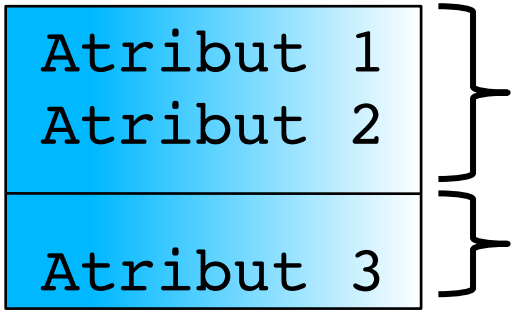
Herència

```
class className: memberAccessSpecifier baseClassName{
   members list
};
```

Classe Base



Classe Derivada (herència de la classe base)



Visibilitat amb herència

```
class circle: public shape {
. . .
};
```

Visibilitat Classe Base	Visibilitat Classe Derivada	Accés derivada?	Accés extern
public	public	SI	SI
private	private	NO	NO
protected	protected	SI	NO

Visibilitat amb herència

```
class circle: private shape {
. . .
};
```

Visibilitat Classe Base	Visibilitat Classe Derivada	Accés derivada?	Accés extern
public	private	SI	NO
private	private	NO	NO
protected	private	SI	NO

Visibilitat amb herència

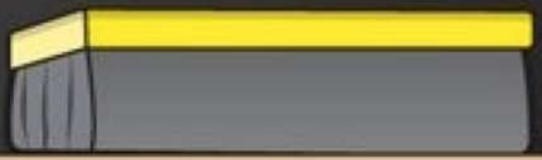
```
class circle: protected shape {
. . .
};
```

Visibilitat Classe Base	Visibilitat Classe Derivada	Accés derivada?	Accés extern
public	protected	SI	NO
private	private	NO	NO
protected	protected	SI	NO

Exemple

```
class Base{
  public:
  int m_nPublic;
  private:
  int m_nPrivate;
  protected:
  int m_nProtected;
};
```

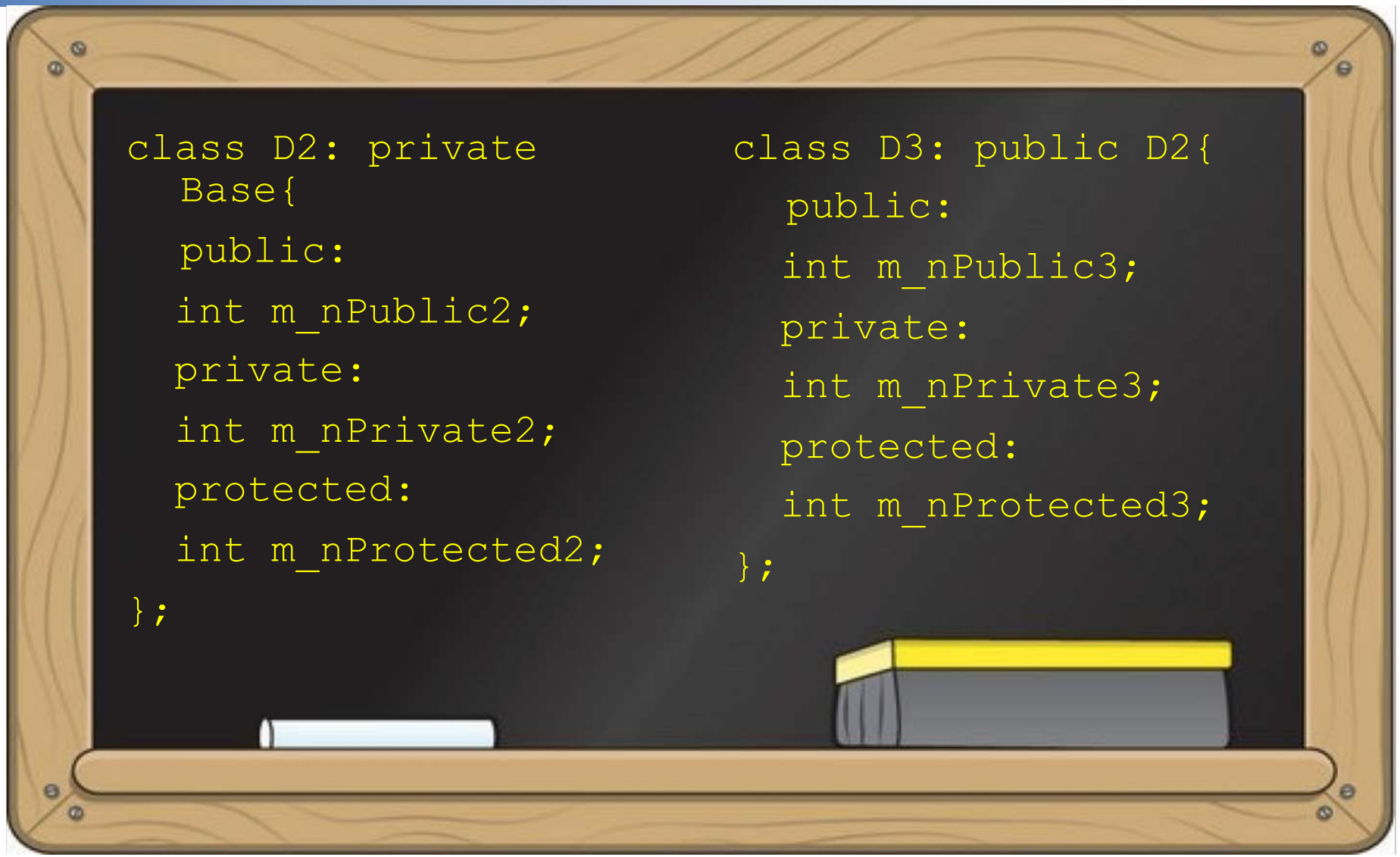
```
class D2: private
  Base{
  public:
  int m_nPublic2;
  private:
  int m_nPrivate2;
  protected:
  int m_nProtected2;
};
```



Exemple

```
class D2: private
   Base{
   public:
   int m_nPublic2;
   private:
   int m_nPrivate2;
   protected:
   int m_nProtected2;
};
```

```
class D3: public D2{
   public:
   int m_nPublic3;
   private:
   int m_nPrivate3;
   protected:
   int m_nProtected3;
};
```



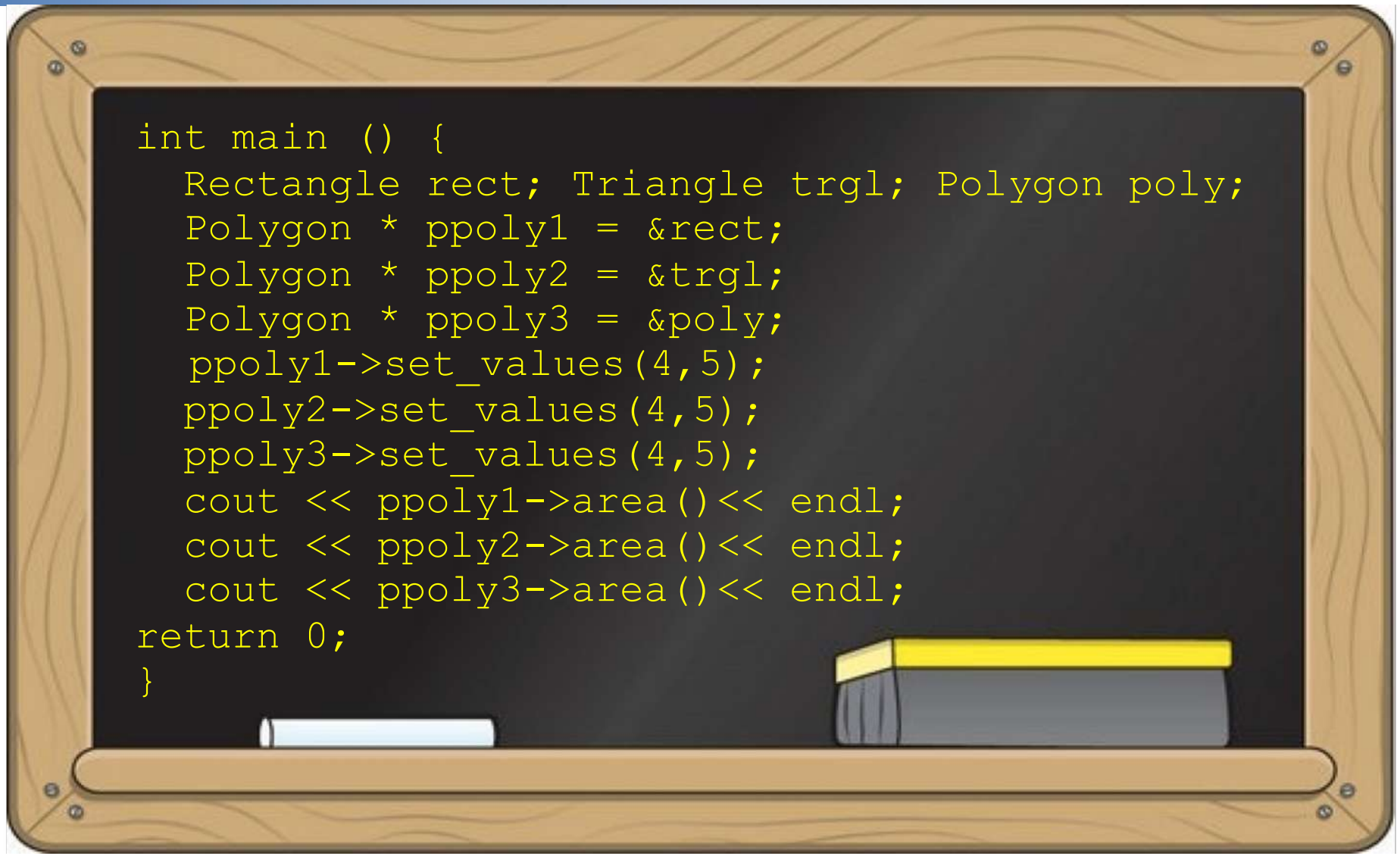
2.6 Polimorfisme

Polimorfisme

- Un punter a una classe derivada és compatible amb un punter a una classe base
- El polimorfisme usa aquesta característica per canviar el comportament
- Una funció **virtual** és una funció que pot redefinir-se a les classes derivades.

```
1  // virtual members
2  #include <iostream>
3  using namespace std;
4
5  class Polygon {
6    protected:
7      int width, height;
8    public:
9      void set_values (int a, int b)
10       { width=a; height=b; }
11     virtual int area ()
12       { return 0; }
13 };
14
15 class Rectangle: public Polygon {
16   public:
17     int area ()
18       { return width * height; }
19 };
20
21 class Triangle: public Polygon {
22   public:
23     int area ()
24       { return (width * height / 2); }
25 };
26
```

Exemple



```
int main () {
  Rectangle rect; Triangle trgl; Polygon poly;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  Polygon * ppoly3 = &poly;
  ppoly1->set_values(4,5);
  ppoly2->set_values(4,5);
  ppoly3->set_values(4,5);
  cout << ppoly1->area()<< endl;
  cout << ppoly2->area()<< endl;
  cout << ppoly3->area()<< endl;
return 0;
}
```

2.7 Templates

Curs 2021 - Els templates els veurem en detall al laboratori d'aquí a unes setmanes

Templates

- Els templates són útils per definir un únic codi en un conjunt de funcions relacionades (**function template**) o un conjunt de classes relacionades (**class template**)

```
template <class Type>
Type larger(Type x, Type y)
{
    if (x >= y)
        return x;
    else
        return y;
}
```

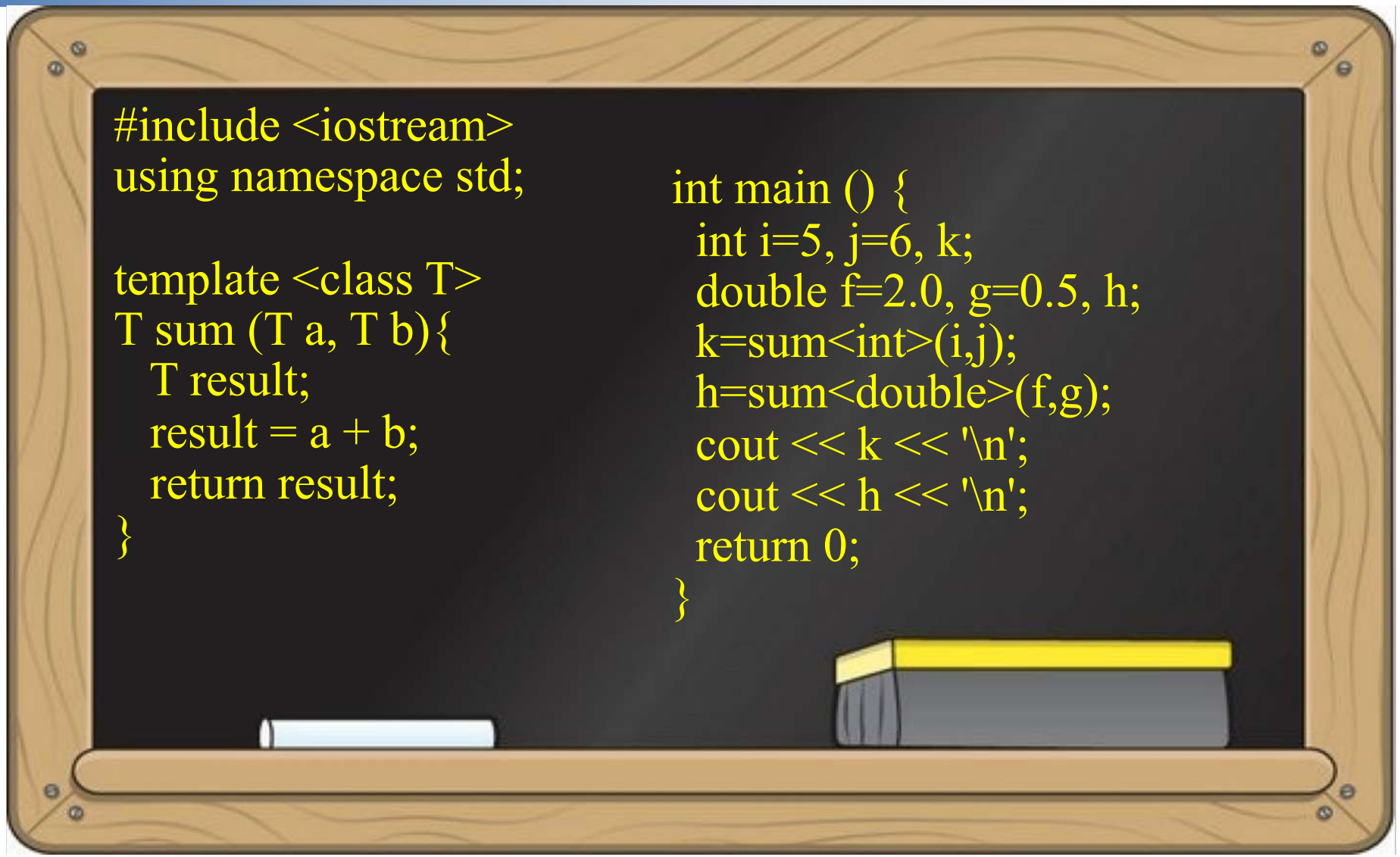
```
cout << larger(5,6) << endl
cout << larger ('A','B')<<endl;
```

Classe Template

```
template <class elemType>
class listType
{
public:
    bool isEmpty();
    bool isFull();
    void search(const elemType& searchItem, bool& found);
    void insert(const elemType& newElement);
    void remove(const elemType& removeElement);
    void destroyList();
    void printList();

    listType();

private:
    elemType list[100];   //array to hold the list elements
    int length;           //variable to store the number
                          //of elements in the list
};
```

Exemple funció template

```
#include <iostream>
using namespace std;

template <class T>
T sum (T a, T b){
  T result;
  result = a + b;
  return result;
}

int main () {
  int i=5, j=6, k;
  double f=2.0, g=0.5, h;
  k=sum<int>(i,j);
  h=sum<double>(f,g);
  cout << k << '\n';
  cout << h << '\n';
  return 0;
}
```

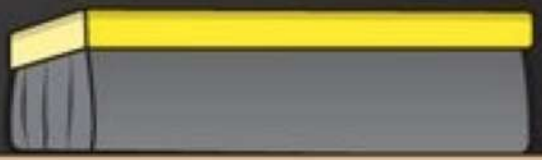


Exercicis

Exercici 1

```
char ch = 'Q';
char* p = &ch;
cout << *p;
ch = 'Z';
cout << *p;
*p = 'X';
cout << ch;
```

Quina és la sortida en cada pas?

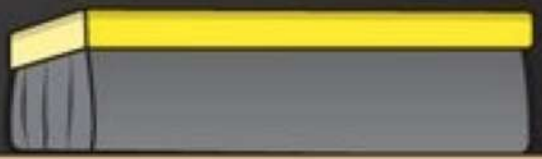


Exercici 2

```
int a[]={0,1,2,3}
int i = 2;
int j = i++;
int k = --i;
cout << a[k++];
```

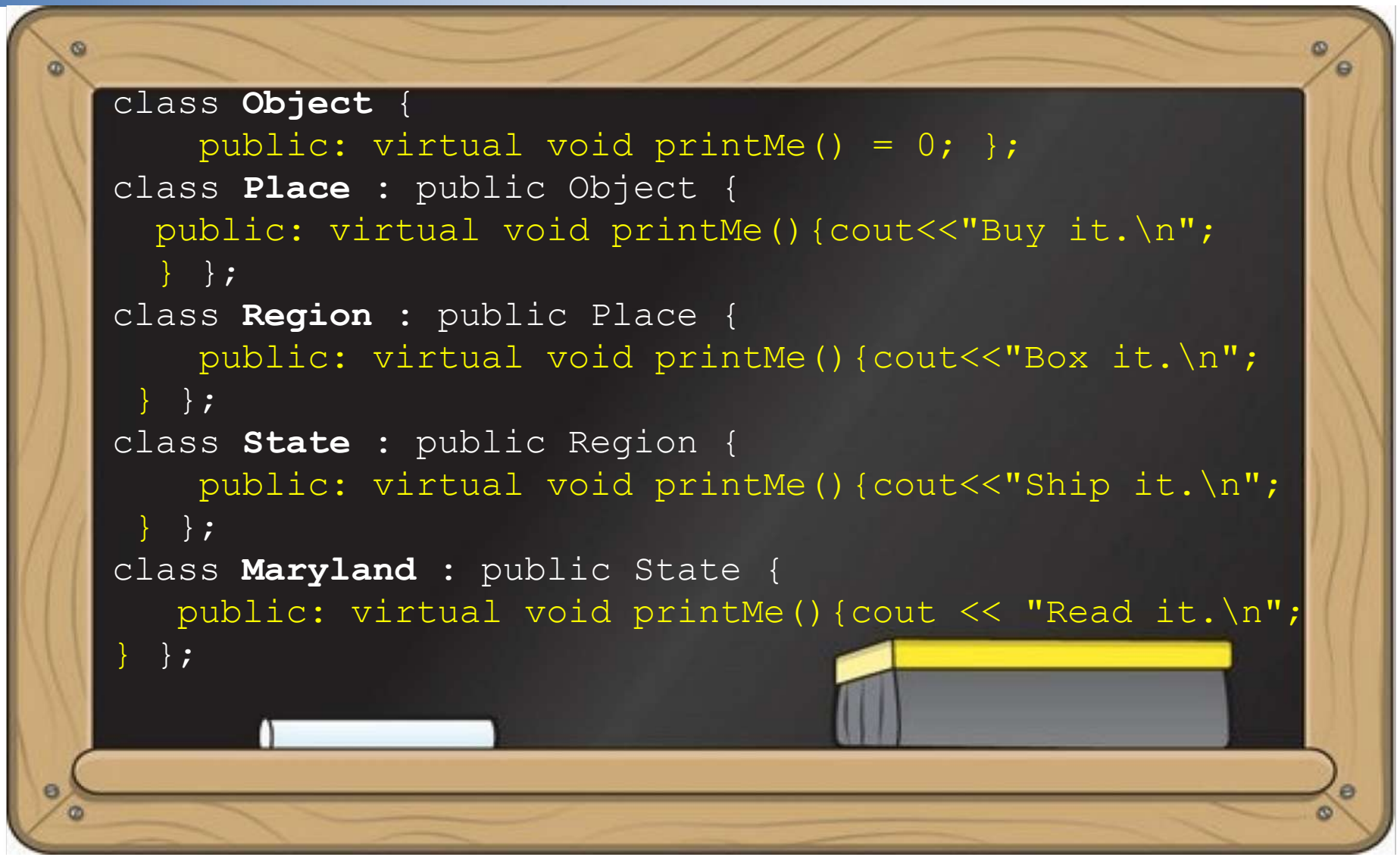
Quina és la sortida en cada pas?

Que val cada variable?



Exercici 3

```
class Object {
    public: virtual void printMe() = 0; };
class Place : public Object {
  public: virtual void printMe(){cout<<"Buy it.\n";
  } };
class Region : public Place {
    public: virtual void printMe(){cout<<"Box it.\n";
 } };
class State : public Region {
    public: virtual void printMe(){cout<<"Ship it.\n";
 } };
class Maryland : public State {
   public: virtual void printMe(){cout << "Read it.\n";
} };
```



Exercici 3 cont.

```
int main() {
  Region* mid = new State;
  State* md = new Maryland;
  Object* obj = new Place;
  Place* usa = new Region;
  md->printMe();
  mid->printMe();
  (dynamic_cast<Place*>(obj))->printMe();
  obj = md;
  (dynamic_cast<Maryland*>(obj))->printMe();
  obj = usa;
  (dynamic_cast<Place*>(obj))->printMe();
  usa = md;
  (dynamic_cast<Place*>(usa))->printMe();
return EXIT_SUCCESS;
}
```

Quina és la sortida?