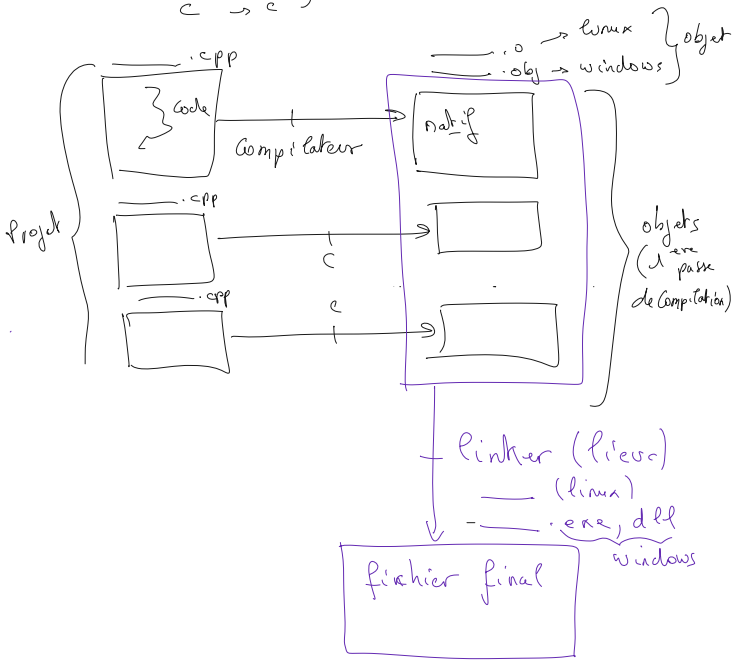
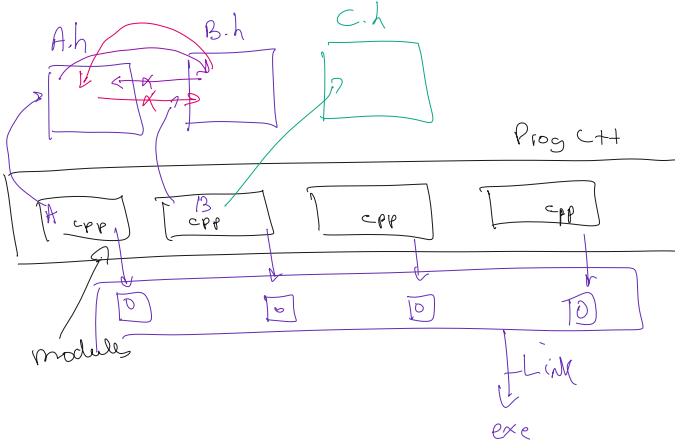
Processus de Compilation :

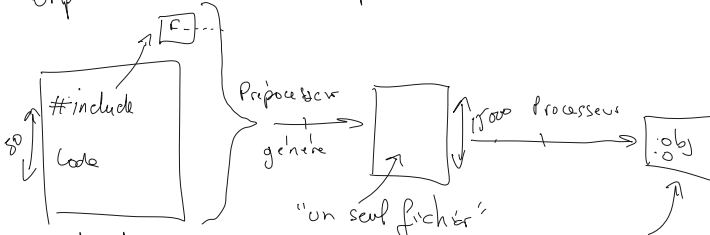
Niveau Code généré

Code source
cpp → c++ } compilateur utilisé
c → c





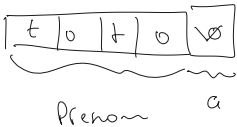
Compilation en 2 étapes



= directives

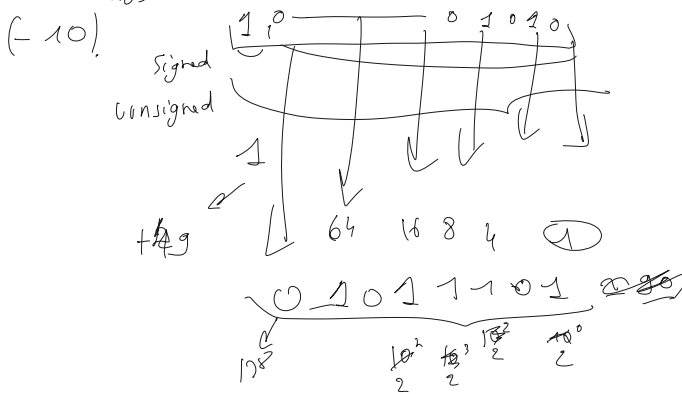
→ traitées par le préprocesseur

ex: #include #define #if
#pragma #ifdef etc...
#ifndef



"toto"

(-10)



base 2

10

16

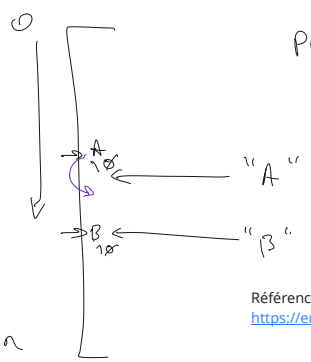
$108_{(10)}$

$8 \times 10^0 = 8$

$+ 0 \times 10^1 + 0$

$+ 1 \times 10^2 + 100$

108

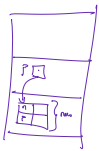


Référence vers la standard library C++ :
https://en.cppreference.com/w/cpp/string

```
class Personne {
  char nom[];
}
```

Java :

```
{
  Personne p = new Personne();
}
```



C++

```
class idem
{
  Personne p; // objet automatique (allocation, libération sur la pile)
}
```

P *p = new P();

P *p2 = p;

e p;

p = 0;

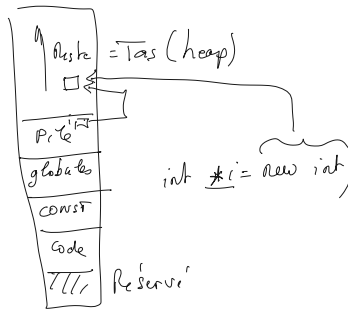
p2. ——— ();

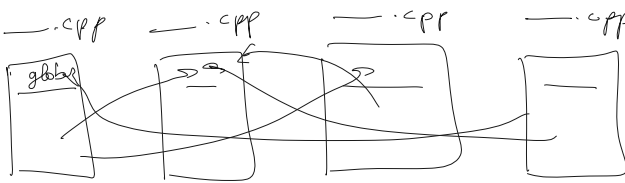
{

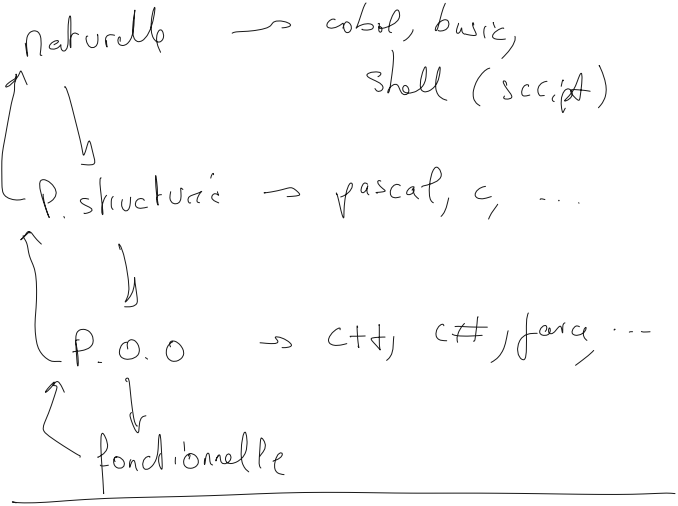
if( )
{ int i;

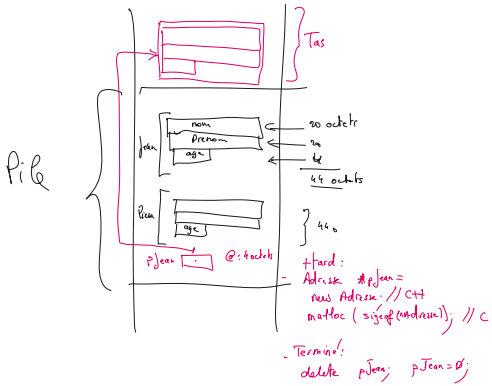
}

}

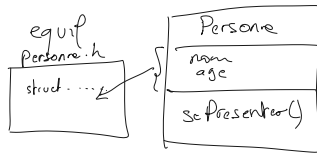





naturelle
→ cobol, basic, shell (script)
P. structuré
→ pascal, c, ...
P. O. O
→ c++, c#, java, ...
fonctionnelle


Pile
Tas
nom
Prenom
age
Jean
50 octets
4 octets
44 octets
Jean
age
44 o
pJean
@: 4 octets
- Hard :
Adresse *pJean =
new Adresse; // c++
malloc (sizeof(adresse)); // c
- Terminé :
delete pJean;
pJean = 0;

C :



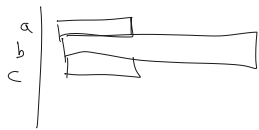
Personne.cpp

```
void sePresenter( Personne *this)
{
  Printf ( this -> nom      (*this).nom
           this -> age
```

---

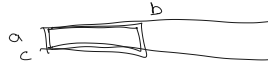
struct

int, double, int
a b c

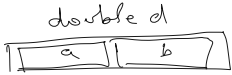


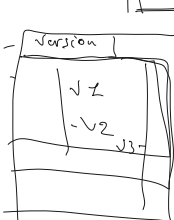
union

int a, double b, int c

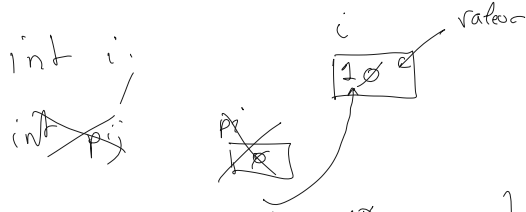


Union { double d
struct { int a, int b }






```
struct Flux {
  int version;
  union {
    struct fv1 { int a; }
    struct fv2 { int a, b; }
  }
}
```

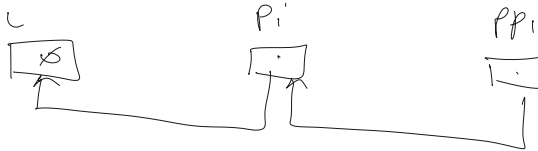


3 concepts :

- "pointeur sur ---" (type)
- "adresse de ---"
- "contenu de ---"

int i;	
int *pi;	
&i	
*pi	

pointeur ⟷ adresse (valeur "int")



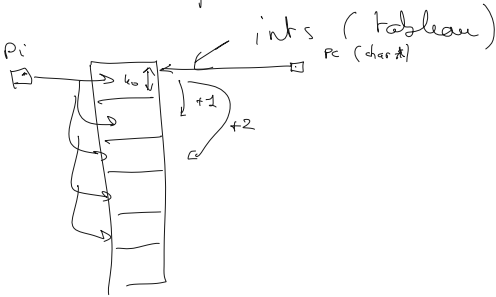
pi = &i;

ppi = π

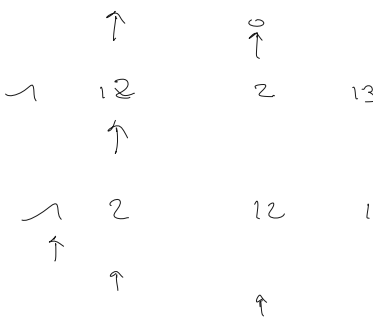
**ppi

```
int i = 10;
proc( ) .... // la proc doit incrémenter notre i local
// ne pas passer par une fonction qui renvoie la valeur modifiée
i doit valoir 11
```

Arithmétique de pointeurs



Trier a__ bottles

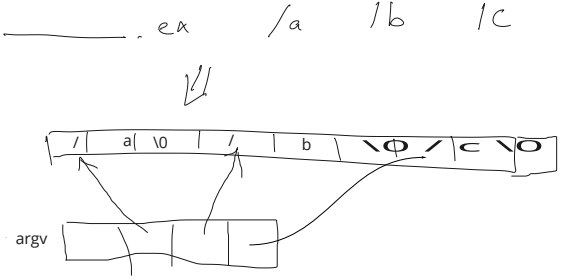


→ Demander à l'utilisateur — n nombres
→ Faites le tri
→ Afficher

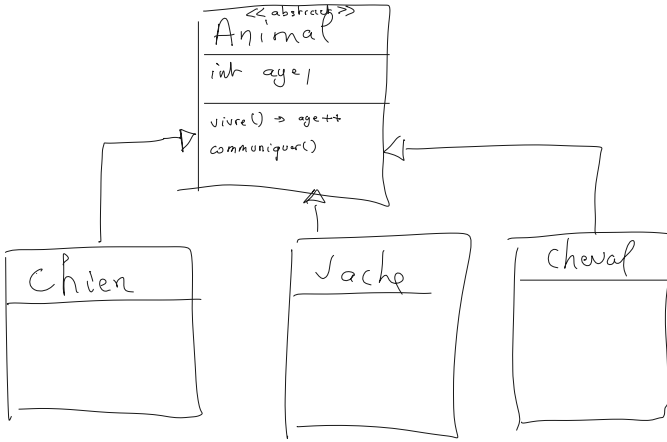
Tri a bulles !

```
void trier(int nombres[], int nbelements )
{
int pos = 0;
while (pos != nbelements-1)
if (nombres[pos] > nombres[pos + 1])
{
int tmp = nombres[pos];
nombres[pos] = nombres[pos + 1];
nombres[pos + 1] = tmp;
if (pos > 0)
pos--;
else
pos++;
}
else
pos++;
}
```

```
void trierpointeurs(int *nombres, int nbelements)
{
int pos = 0;

while (pos != nbelements - 1)
if (*(nombres+pos) > *(nombres+pos + 1))
{
int tmp = *(nombres+pos);
*(nombres+pos) = *(nombres+pos + 1);
*(nombres+pos + 1) = tmp;
if (pos > 0)
pos--;
else
pos++;
}
else
pos++;
}
```



Créez une classe Carre qui prends une arete en paramètre du constructeur et peut renvoyer la surface et son périmètre



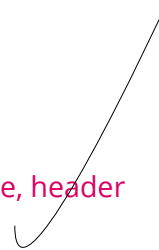
→ chaque animal communique à sa manière (juste renvoyer "cha'ne)

→ Creez un Tableau d'Animal et mettez y des instances variées.

→ parcourez votre tableau et affichez le résultat de communiquer

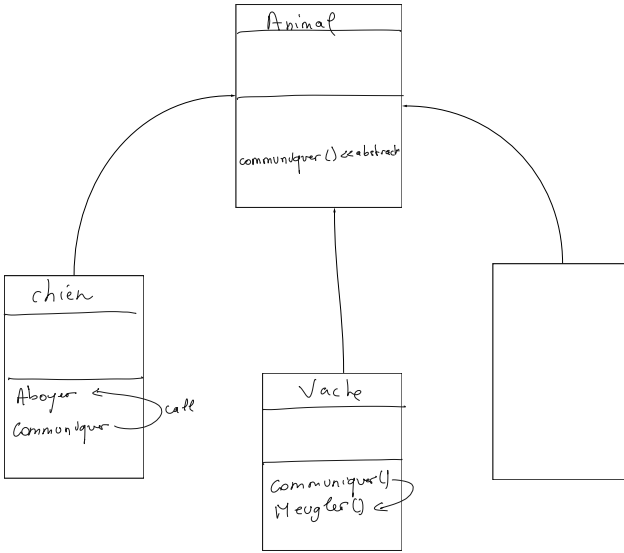
```
Animal *animaux = new Animal[n];
animaux[0] = new chien();
```

```
class Forme
{
public:
virtual int perimetre() = 0; // Virtual pure = abstract
virtual int surface() = 0;
virtual char* description();
};

class Carre : public Forme
{
public:
Carre(int);
~Carre();

virtual int perimetre();
char* decrire();
char* description();
protected:
int arete;
const char* msg_decrire = "Je suis un carré de %u
de coté.";
};

class Rectangle : public Carre
{
public:
int cote2;

Rectangle(int cote1, int cote2);

int perimetre();
};

class SRectangle : public Rectangle
{
public:
SRectangle(int cote1, int cote2) : Rectangle(cote1,
cote2) {}

int perimetre();
};
```



Héritage, implémentation

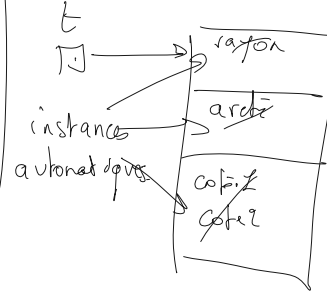
```
Carre::Carre(int arete)
{
this->arete = arete;
}

Carre::~Carre()
{
}

int Carre::perimetre()
{
return 4 * arete;
}

char* Carre::decrire()
{
const int size_msg = strlen(msg_decrire) + 20000;
char* buff = new char[size_msg];
sprintf_s(buff, size_msg, msg_decrire, arete);

return buff;
}

char* Carre::description()
{
return "Je suis un Carre";
}

Rectangle::Rectangle(int cote1, int cote2) : Carre(cote1)
{
this->cote2 = cote2;
}

int Rectangle::perimetre()
{
return 2 * arete + 2 * cote2;
}
int SRectangle::perimetre()
{
return 4 * arete + 4 * cote2;
}
```

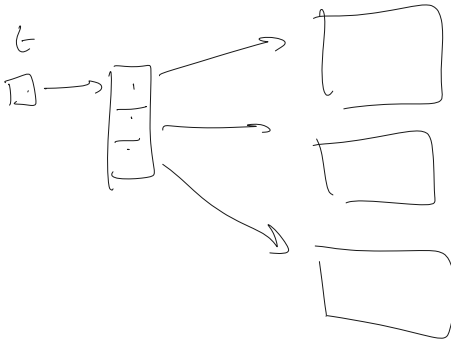


Animal [I?]

Java :

C++





Instancier un tableau d'élements concrets basés sur une classe abstraite :

```
Forme **formes = new Forme*[3];
formes[0] = new Carre(10);
formes[1] = new Rectangle(10, 2);
formes[2] = new Carre(15);

for (int c = 0; c < 3; c++)
printf("%d\n", formes[c]->perimetre());
```

Exercice : Faire une liste chaînée (de chaîne, ou entier, ou Template)