* Antoine Jouglet
* TP/TD : 4h par semaine, un coup l’un un coup l’autre
* Examens
  + 20% TP + 40% M + 40% F
  + Poly annoté autorisé au final
* Compilateur : Penser à Qt creator

Compte LO21

* + lo21p035
  + wejo8DUP
* TD/TP
  + Thibaut Arribe
  + thibaut.arribe@utc.fr
* Objet
  + Ce n’est forcément quelque chose qui se touche
  + C’est un truc
  + Nous avons la faculté de percevoir des objets grâce à leurs attributs. Ils nous permettent de définir différents objets.

Attributs

* + Ce ne sont pas des objets, ils servent à les caractériser
  + Se retrouve dans un certain nombre d’attributs
  + Objet au croisement de différents attributs
  + Ensemble d’objets qui partagent des attributs communs : classe
* Classe
  + Abstraction
* Modélisation orientée objet est toujours contextuelle : on choisit des attributs pertinents p/r l’utilisation qu’on va faire de cette modélisation.
* Approche objet
  + Utilisation d’objets.
  + Ils interagissent entre eux pour réaliser une fonction => système
  + Système décomposé sur ce qu’il est et fait
* Attributs
  + Chaque objet est responsable de ses attributs
* Méthode
  + Fonction qui s’exécute sur un objet particulier
* Avantages approche OO (espèce d’arc-en-ciel)
  + Stabilité
* Analyse et conception orientées objet
  + Découpage naturel et intuitif contre le complexité d’un problème.
  + Découpage horizontal
    - Exemple : (polygones (quadrilatères (parallélogrammes (losanges rectangles))triangles pentagones))
* Outils
  + UML
    - Langage graphique
    - On l’utilise comme on veut
    - Degré de détails pertinent à ce qu’on veut en faire
  + Design Pattern
* Préprocesseur
  + Plus de .h.
  + iostream : entrée sortie
  + string : bibliothèque pour générer les caractères
  + namespace
    - portée globale/portée locale d’une fonction
    - portée globale divisée en plusieurs sous espaces
      * tous les éléments des bibliothèques standard sont déclarés dans std.
* cout
  + objet qui appartient à la classe ostream, qui représente la sortie standard
  + Pendant de printf
  + cout<<“Bonjour ! \n“ ; = cout.operator<<(“Bonjour ! \n“) ;
* cin
  + Pendant de scanf.
* Définition et déclaration d’une variable
  + Définir une variable : réserver un espace mémoire.
  + Déclarer une variable : dire que quelque part il y aura une variable (mot clé : extern)
  + Initialiser les variables, c’est bien sinon on n’est pas sûr de sa valeur
  + Une définition est aussi une déclaration
  + Identificateur de fonction ou de variable doit être déclaré avant utilisation
* Données
  + Portée
    - Où est par qui la donnée est-elle visible ?
    - Visible qu’à partir de la ligne où elle a été déclarée

|  |
| --- |
| int i  for (i=0 ; i<10 ; i++)  cout << i << “ \n ” ;  int i = 5 ;  for (int i=0 ; i<10 ; i++)  {  cout << i << “\n” ; \\ affiche i, le compteur de la boucle, ce n’est pas le même i que celui qui avait été déclaré avant la boucle for.  cout << ::i << “\n” ; \\ affiche 5  } |

* + Durée de vie
    - Quand est-ce que la donnée existe ? = quand est-ce que la données a un espace mémoire qui lui a été réservé ?
  + Segments mémoire
    - Segment statique
      * Variables globales et statiques
      * Durée de vie = celle du programme
      * static
      * Quand on a besoin qu’une variable soit toujours là
    - Pile
      * stack
      * variables automatiques
      * vie dans le bloc à partir de leur déclaration
    - Tas
      * Heap

Taille des objets

* + Sous forme de multiple de la taille de char.
  + Taille de char = 1 (une fois moins 8 bits, mais ça peut être 16, 32, …)
* Initialisation et affection
  + Initialisation : int i=4 ;
  + Affectation : i=5 ;
* Constantes
  + const
  + Super important
* Structures
  + struct
  + Forme simple de class
  + struct id  
    {  
     type1 id1 ;  
     …  
     typen idn ;  
    }

Objet

* + Région contiguë de mémoire
  + Au plus un identificateur : pas forcément besoin de connaître son nom
* Adressage indirect
  + Pointeurs
    - A une valeur entière
    - Numéro d’un emplacement mémoire
      * Peut être valide ou non
      * Peut correspondre ou non à une adresse allouée
    - type\* : pointeur de type
    - pointeur nul : 0
    - Conversion implicite en bool :
      * Nul => False
      * Non nul => True
    - Pointeurs const
      * const type\* pt
      * Il pointe sur une zone qu’on ne peut pas modifier par l’intermédiaire du pointeur
      * En revanche on peut la modifier par l’intermédiaire d’autre chose
  + Références
    - Doit être initialisée
    - Ne pas être réassignée
    - Jamais nulle
* lvalue
  + A l’origine, expression qu’on peut être à gauche de l’opérateur d’affection
  + Exp = … ; Exp est une lvalue
  + Maintenant, elle désigne un emplacement mémoire