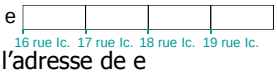
Les données de type « pointeur »

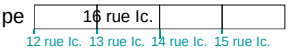
- Découpage de la mémoire vive de l'ordinateur en unités de base (les octets)
- Variable :
  désigne un emplacement mémoire occupant 1 ou plusieurs unités de base
- Une variable est caractérisée par une **adresse**
- Possibilité de stocker des **valeurs d'adresses** dans des **variables de type pointeur**

Norme Algorithmique

- **Opérateur & :**
  Accès à l'adresse d'une variable x de type T
  Ex : **variable**
  e : **entier**



  &e retourne la valeur de l'adresse de e
- Définition d'une variable de type pointeur



  Ex : **Variable**
  pe : **pointeur sur entier**
  Dans pe on peut ranger l'adresse d'une variable de type entier
  pe ← &e

- **Opérateur ↑** (de déréférencement)
  Accès à l'emplacement de type T situé à l'adresse fournie en opérande de l'opérateur

  Ex :
  pe↑ désigne la variable dont l'adresse est écrite dans pe
  pe↑ désigne donc ici la variable e
  On dit que e est la variable pointée par pe

  pe↑ devient un alias de e
  (seulement tant que pe pointe sur e …)
- Remarques :
  Il y aura autant de type pointeur que de types pointés

  Le déréférencement n'est possible que si l'adresse est valide (précondition). Penser à initialiser les variables de type pointeur!

- **Variable**
  e_1 : **entier** ← 1
  e_2 : **entier** ← 2
  pe : **pointeur sur entier** ← nil
- **Début**
  pe ← **&** e_1
  pe↑ ← 3
  pe ← **&** e_2
  pe↑ ← 4
  e_2 ← 5
  …
  Fin

Pointeurs et adressage en C/C++

- **Opérateur &**
  Accès à la valeur de l'adresse d'une variable
  ex : int a;
  &a renvoie la valeur de l'adresse de a
  &a peut être stockée dans une variable de type **int ***

  Attention : Ne pas confondre avec le & permettant de définir des références en C++

- **Définition de variables de types pointeur**
  Nom_Type *nom_variable;

  La variable pointée par **nom_variable** est de type **Nom_Type**

  Ex : **int ***pa;

  pa destinée à contenir des adresses de variables de type int

  Ex : pa = **&**a ;
  // ou a est une variable de type int

- **Opérateur *** (de déréférencement)
  Pour accéder à une variable à partir d'une valeur d'adresse

  Ex : ici ***pa** désigne a

  Remarque : ***pa** identique à ***(&a)**
  identique à a

- Remarque : Pour déréférencer un pointeur, on peut aussi utiliser l'opérateur [] avec argument 0

  ***pa** identique à pa[0]

  (cf transparents ultérieurs sur les pointeurs et les tableaux C/C++)

- **void *pg;**
  Permet de définir une variable pg de type adresse générique (pouvant contenir l'adresse d'une variable de n'importe quel type).
  AUCUN DEREFERENCEMENT POSSIBLE D'UN void *
- **Conversion implicite possible d'un pointeur sur type T en void ***
  int i;
  int *pi=&i;
  pg=pi;
- **Conversion d'un void * en un pointeur sur type T**
  - Implicitement possible en C
    double *pdb;
    pdb=pg;
  - Mais pas en C++ ISO (nécessité d'un *cast* **explicite**)
    pdb=(double *)pg;
    ou plutôt
    pdb=reinterpret_cast<double *>(pg);

Pour les conversions dangereuses :
Entre 2 types pointeurs sans rapport
Entre un type pointeur
et un type entier

Pourquoi les pointeurs ?

- N'utiliser que la mémoire nécessaire
  - La mémoire d'un ordinateur est limitée en taille.
  - Ne réserver de la mémoire qu'en cas de nécessité.
  - Libérer cette mémoire une fois le traitement terminé.
  - Cette place libérée est ainsi disponible pour une autre utilisation dans le même programme.
  - Possibilité d'allouer de la mémoire qui survivra au bloc dans lequel est réalisé son allocation
- Construire des structures de données dynamiques
  - Dont la structure peut évoluer au cours de l'exécution d'un programme
  - Avec partage possible d'informations entre certaines entités (éviter certaines duplications d'information)

- Une utilisation des pointeurs en C :
  - Mise en œuvre du passage de paramètres résultat ou donnée résultat à une procédure
    (pas de références en C,
    seulement en C++)
  - Ex :
    ```
    void swap(int a, int b)
    { int c;
      c=a;
      a=b;
      b=c;
    }
    ```
  - l'appel :

    ```
    int aa=5, bb=3;
    swap(aa,bb);
    ```

    Que se passe-t-il ?

- Solution :
  Au programmeur et à l'utilisateur de la fonction C d'introduire un niveau d'indirection …

  ```
  void swap(int *pa, int *pb)
  { int c;
    c = *pa;
   *pa = *pb;
   *pb = c;
  }
  ```

  Paramètres formels de type adresse sur les variables à modifier

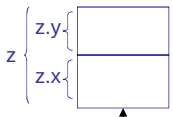
- A l'appel :

  ```
  int aa=5, bb=3;
  swap(&aa,&bb);
  ```

  Ce sont des valeurs d'adresses qui sont communiquées

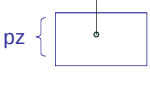
Pointeurs sur structures

Complexe z;
//Création sur la pile

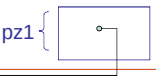


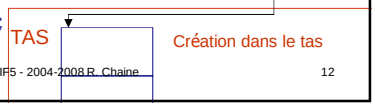
Complexe *pz=&z;

Notations :
(*pz).x peut s'écrire pz->x



Complexe *pz1=new Complexe;
initialiser(*pz1);
…
testament(*pz1);
delete pz1;





Attention:

- Une structure A ne peut avoir de donnée membre de type A
- Une structure A peut avoir une donnée membre de type A*