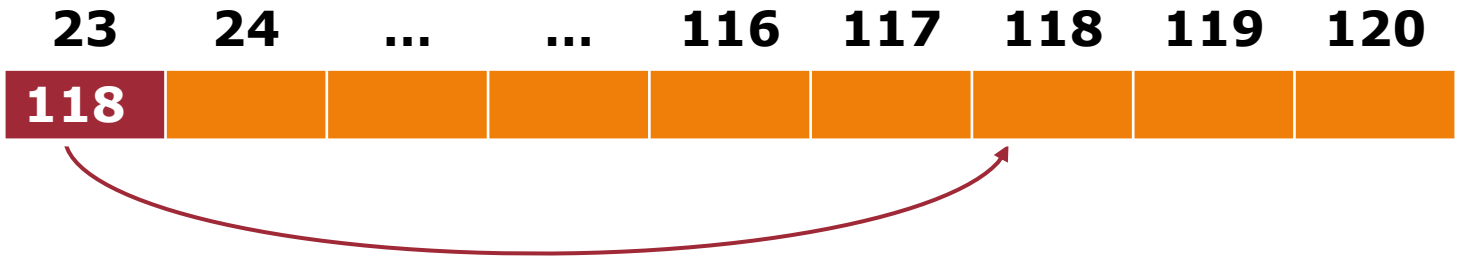
Les pointeurs

Définition

- Variable contenant l'adresse d'une autre variable d'un type donné



Intérêt des pointeurs

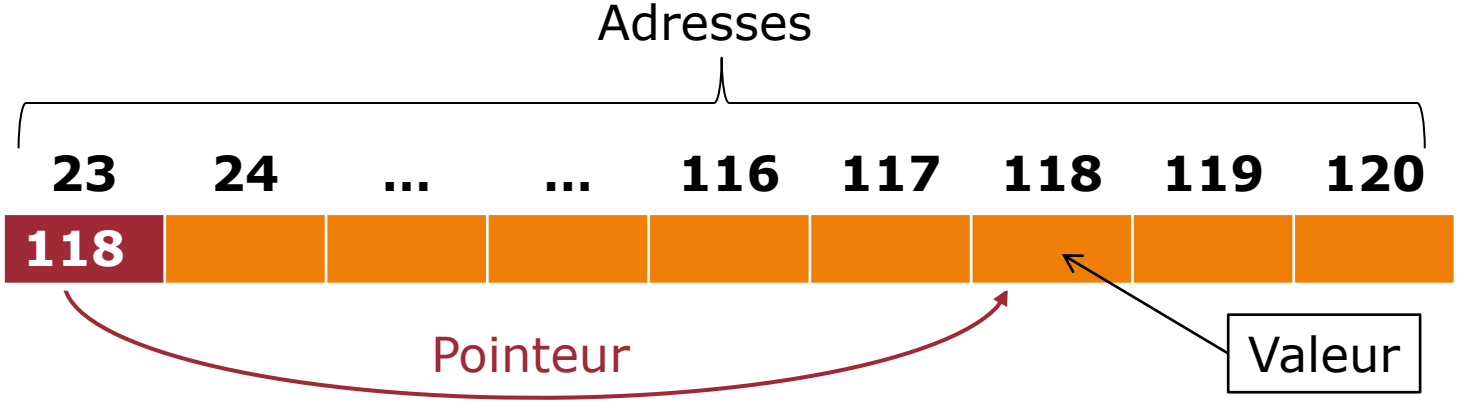
- Possibilité de manipuler des données de grande taille très simplement

- Créer des structures dynamiques
  - Taille évoluant au cours du temps
  - ≠ des tableaux

- Créer des structures complexes chainées (listes, arbres…)

Notion d'adresse

- Chaque bloc de la mémoire est identifié par un numéro (son adresse)
- Toute variable prend un nombre précis de cases mémoires (selon le type)
- Stockage de l'adresse de la variable dans un pointeur



Adresse d'une variable

- Pas besoin de manipuler directement l'adresse d'une variable

- Syntaxe pour connaitre l'adresse d'une variable:

```
&Nom_de_variable
```

Déclaration des pointeurs

- Syntaxe de la déclaration:

```
type* pointeur;
```

- Le type peut être un type de base (`int`, `char`), ou alors un type complexe (`struct`…)

Initialisation d'un pointeur

- Etape nécessaire car par défaut pointe n'importe où…

- Syntaxe:

```
Nom_du_pointeur = &variable_pointee;
```

```
int a = 4;
int* p1 = &a;
```

Accès à une variable pointée

- Pour accéder au contenu:

```
*pointeur
```

```
*p1=10;
```

- Dans une expression plus complexe:

```
a= (*p1)+3;
```

Utilisation des pointeurs

Passage d'argument à une fonction **par valeur**

```
int Ajout2(int a)
{ a +=2;
  return a; }

int b = 3;
b = Ajout2(b)
```

Passage d'argument **par adresse**

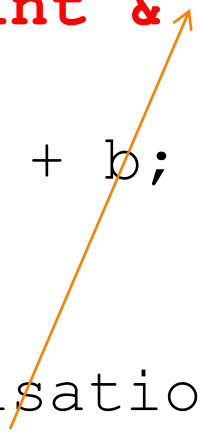
```
void Ajout2(int * a)
{ *a +=2; }

int b = 3;
Ajout2(&b);
```

Passage d'argument **par référence**

```
int Ajout2(int &);

int Ajout2(int & a)
{ a +=2; }

int b = 3;
Ajout2(b);
```

Passage d'argument par référence

```
/*Prototype*/
Ajout(int &, int);

/*Fonction*/
Ajout(int & a, int b)
{
  a = a + b;
}

/*Utilisation*/
int a = 3;
Ajout(a,4); /*Ajoute 4 à la variable a*/
```



Les références

- Utilisation en dehors du passage de paramètres à une fonction:

```
int e1;
int e2 = 10;

//Initialisation de la référence obligatoire
int& ref_e1=e1;

//Change la valeur de ref_e1 et e1 à 25.
ref_e1=25;

//Ne change pas la référence vers e1 mais seulement la
  valeur.
ref_e1= e2;

//Toutes les valeurs sont à 10
cout << e1 << e2 << ref_e1 << ref_e2;
```

Tableaux et pointeurs

- L'identificateur d'un tableau sans les [] est un pointeur

```
int t[10];
```

- `t` est un pointeur vers la première case du tableau
  - même chose que `&t[0]`
  - de type `int*`
- `t+i` est un pointeur vers la (`i+1`)ème case du tableau (`&t[i]`)
- Initialisation d'un tableau

```
for(int i=0; i<10;i++) {*(t+i)=0;}
```

Passage d'un tableau en argument d'une fonction

- Un tableau est automatiquement passé comme un argument par adresse d'une fonction
  - ses valeurs sont modifiables par la fonction

```
void init(int t[], int n, int val) //autre écriture: int *t
{
        int i;
        for(i=0; i<n; i++)
        {
                t[i]=val;
        }
}
```

Cas des tableaux à deux dimensions

- L'identificateur d'un tableau à deux dimension est aussi un pointeur

```
int t[3][4];
```

- C'est un tableau à trois éléments qui sont des tableaux de 4 entiers.

- `t[0]` est un pointeur vers le premier entier du premier tableau (`&t[0][0]`)