# Initiation aux concepts objets en C++

Du C au C++ - Découverte du paradigme objet

# Sommaire

# Sommaire (détail)

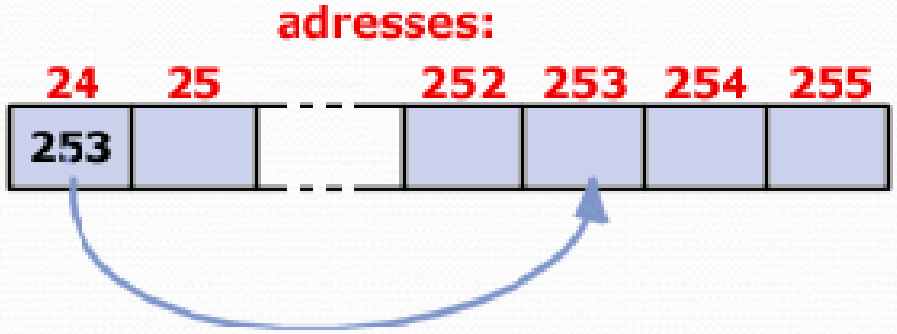
- V. Pointeurs, références et surcharge
  A. Rappel sur les pointeurs
  B. Références
  C. Surcharge d'opérateurs
  D. Surcharge de fonctions

# Objectifs

- Comprendre la différence entre pointeurs et références
- Comprendre la notion de surcharge

# Rappel sur les pointeurs (1/2)

- Pointeur = adresse mémoire




- Symbole : & (« esperluette ou et commercial ») placé **devant** une variable

- Très utile pour les chaines, les tableaux, listes chainées, passage « par adresse »…

# Rappel sur les pointeurs (2/2)

- Déférencement : Récupération de la valeur pointée

```
(*ptr)

(*Nom du pointeur)
```

- Opérateur de déférencement : * (étoile)

# Les références (1/5)

- Symbole : & « et commercial ou esperluette » placé **après** le type
- **A ne pas confondre avec l'adresse d'une variable !!**
- Alias pour un objet (un nom alternatif)

# Les références (2/5)

- Sorte de pointeur « caché »
- **Doit TOUJOURS être initialisée !!**
- Exemple :

```
int var = 10;
int& ref = var; // alias on "var"
std::cout << "ref = " << ref << std::endl;
```

# Les références (3/5)

- Utile pour le passage de paramètres :

```
void swap(int& a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

Passage par référence !

```
int x = 5, y = 10;
swap(x, y);
std::cout << "x = " << x << ", y = " << y << std::endl;
```

- Remplace le passage « par adresse » du C

# Les références (4/5)

- Utile pour le passage d'objets en paramètre :

```
void action(MyObject& obj)
{
    obj.setValue(1500);
}
```

- Souvent utilisé avec le mot-clé « const »

# Les références (5/5)

- Evite les allocations et désallocations de la mémoire
- Evite les fuites mémoires
- Evite la recopie des objets : gain de vitesse
- Garantie que l'objet sera non nul

# Surcharge d'opérateurs (1/3)

- Rédéfinir le comportement des opérateurs
  - Opérateurs arithmétiques (+, -, *, /)
  - Opérateurs logiques (!, &&, ||)
  - Opérateurs de comparaison (==, >=, <=, !=)
  - Opérateurs de flux (>> et <<)
  - …
- Permet d'utiliser des opérateurs entre des objets

# Surcharge d'opérateurs (2/3)

- Exemple : addition de deux complexes :

```
Complex &Complex::operator+=(const Complex &c)
{
    m_x += c.m_x;
    m_y += c.m_y;
    return *this;
}
```

# Surcharge d'opérateurs (3/3)

- Autre méthode : addition de deux complexes

```
Complex operator+(const Complex &c1, const Complex &c2)
{
    Complex result = c1;
    return result += c2;
}
```

- Chaque opérateur à son prototype (Google !)

# Surcharge de fonctions (1/3)

- Fonction avec même nom mais paramètres différents
- Type de constructeurs = surcharges

# Surcharge de fonctions (2/3)

- Exemple avec des fonctions classiques :

```
4   // Multiplication for integer
5   int multiply(int nb1, int nb2)
6   {
7       return (nb1 * nb2);
8   }
9
10  // Multiplication for double
11  double multiply(double nb1, double nb2)
12  {
13      return (nb1 * nb2);
14  }
```

# Surcharge de fonctions (3/3)

- Exemple avec des constructeurs :

```
Complex::Complex(double x, double y)
    : m_x(x), m_y(y)
{

}

Complex::Complex(const Complex&source)
{
    m_x = source.m_x;
    m_y = source.m_y;
}
```

# Prochaine séance

- Héritage et polymorphisme