

Support de cours

Cours:

## Introduction à la programmation orientée objet (en C++)

Vidéo:

### W11-01-intropoo-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

**Savoir de façon globale. Moyen d'une largeur. Apprentissage de la programmation. Surface d'un rectangle. Outillage de base. Nombre de propriétés intéressantes. Lien direct. Façon particulière. Variable largeur. Séquence présente. Stade du cours. Absence de lien sémantique. Exemple des tableaux. Langage c. Notion centrale de fonction.**



[vers la recherche de séquences vidéo](#)

(dans Introduction à la programmation orientée objet (en C++).)



[vers la vidéo](#)

Center for Digital Education. Plus de matériel de soutien pédagogique ici :

<https://www.epfl.ch/education/educational-initiatives/cede/educational-technologies-gallery/boocs-en/>



# Introduction

## (Partie 1)

### Introduction à la programmation orientée objet (en C++)

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





Nous abordons aujourd'hui ensemble un volet très important de votre apprentissage de la programmation. La programmation dite orientée objet. Il s'agit d'une façon particulière de programmer qui n'est d'ailleurs pas spécifique au langage C++ et qui va donner à vos programmes, un certain nombre de propriétés intéressantes en terme de maintenabilité et de modularité. A ce stade du cours, vous êtes censés connaître un certain nombre d'éléments fondamentaux relatif à la programmation tout court. Vous savez par exemple, exprimer des traitements en utilisant des structures de contrôles, comme par exemple, des boucles conditionnelles. Vous savez aussi structurer un minimum vos données

notes

résumé

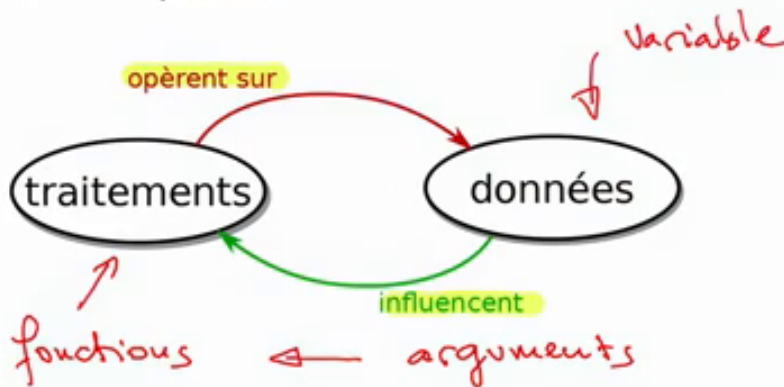
0m 1s



Dans les programmes que vous avez écrits jusqu'à maintenant, les notions

- ▶ de variables/types de **données**
- ▶ et de **traitement** de ces données

étaient séparées :



en utilisant par exemple des tableaux, et vous savez surtout modulariser vos programmes en utilisant la notion centrale de fonction. Cet outillage de base va vous permettre d'exercer un style de programmation particulier dit programmation procédurale ou impérative. Il a la particularité de faire en sorte que les données et les traitements apparaissent de façon séparée dans un programme. Certes, il y a une interaction entre les deux, les traitements opèrent typiquement sur les données, lesquelles influencent à leur tour les traitements, Mais les deux entités apparaissent de façon séparée. Par exemple, comme nous allons le voir dans quelques minutes sur un exemple concret, les traitements peuvent s'exprimer par le biais des fonctions. Et le lien entre données et traitement peut se faire alors par le biais du passage des arguments. Les données manipulées apparaissent au travers d'entités distinctes, comme par exemple, des variables.

notes

résumé

0m 37s



```
double surface(double largeur, double hauteur);

int main()
{
    double largeur(3.0);
    double hauteur(4.0);

    cout << "La surface du rectangle est : "
          << surface(largeur, hauteur) << endl;

    return 0;
}

double surface(double largeur, double hauteur)
{
    return largeur * hauteur;
}
```

Une des particularités fondamentales de l'orienté objet est de permettre le regroupement des traitements et des données en une seule et même entité.

notes

résumé

1m 25s



```
double surface(double largeur, double hauteur);

int main()
{
    double largeur(3.0);
    double hauteur(4.0);

    cout << "La surface du rectangle est : "
         << surface(largeur, hauteur) << endl;

    return 0;
}

double surface(double largeur, double hauteur)
{
    return largeur * hauteur;
}
```

Partons d'un exemple concret. Imaginons que je souhaite écrire un petit programme C++ qui calcule la surface d'un rectangle. Le rectangle est représenté au moyen d'une largeur et d'une hauteur. Et je souhaite, donc, en calculer la surface. En programmation procédurale, j'aurais tendance à procéder de cette façon. Je vais commencer par déclarer une variable largeur pour représenter la largeur du rectangle, que j'initialiserai avec une valeur appropriée. Et je vais faire de même pour la hauteur. Ensuite, le calcul de surface peut se faire en passant la largeur et la hauteur ainsi définies à une fonction surface qui se charge de réaliser le calcul approprié.

notes

résumé

1m 37s



```
double surface(double largeur, double hauteur);

int main()
{
    double largeur(3.0);
    double hauteur(4.0);

    cout << "La surface du rectangle est : "
          << surface(largeur, hauteur) << endl;

    return 0;
}

double surface(double largeur, double hauteur)
{
    return largeur * hauteur;
}
```

Alors, on voit bien au travers de cet exemple, que dans un tel programme, les données et les traitements apparaissent de façon séparée. Les variables me permettent de modéliser et de stocker les données dont j'ai besoin. Et les traitements sont réalisés par le biais de fonctions. Il n'existe pas de lien direct entre les données et le traitement, le lien s'établit ici, uniquement, par le biais, du passage des arguments. Alors une des critiques fondamentales, que l'on peut émettre sur ce genre de programme, est l'absence de lien sémantique entre les différentes entités. Par exemple, le lien sémantique qui unit la largeur et la hauteur, il s'agit de la largeur et de la hauteur d'un rectangle, est difficile à établir. Imaginons, par exemple, que je ne sois pas francophone. Donc pour moi, largeur et hauteur sont des noms peu parlant.

notes

résumé

2m 14s



```
double surface(double largeur, double hauteur);

int main()
{
    double largeur(3.0);
    double hauteur(4.0);

    cout << "La surface du rectangle est : "
          << surface(largeur, hauteur) << endl;

    return 0;
}

double surface(double largeur, double hauteur)
{
    return largeur * hauteur;
}
```

*Handwritten annotations:*

- A blue arrow points from `largeur(3.0)` to `largeur` in the function signature, with the text "lien ?" next to it.
- A red arrow points from `surface(largeur, hauteur)` to the `surface` function definition, with the text "surface" next to it.
- A red bracket groups `largeur` and `hauteur` in the function signature, with the text "produit" next to it.
- A red bracket groups `largeur` and `hauteur` in the function body, with the text "h" next to it.

Il est difficile pour moi de voir qu'il s'agit de deux entités liées à un même concept : celui du rectangle. Le lien sémantique qui unit ici, les données entre elles, n'est pas explicite. Mais il en va de même pour le lien qui unit les données et le traitement qui agissent sur ces données. Imaginons que je sois un petit peu moins précautionneuse dans le choix de mes identificateurs. Et que j'appelle cette fonction produit par exemple. On peut même imaginer que j'appelle mes arguments, que je leur donne des noms moins parlant. Du coup, il devient très difficile pour quelqu'un qui lit le code de voir que l'on est effectivement, ici en train de faire un calcul de surface sur un rectangle.

notes

résumé

3m 1s

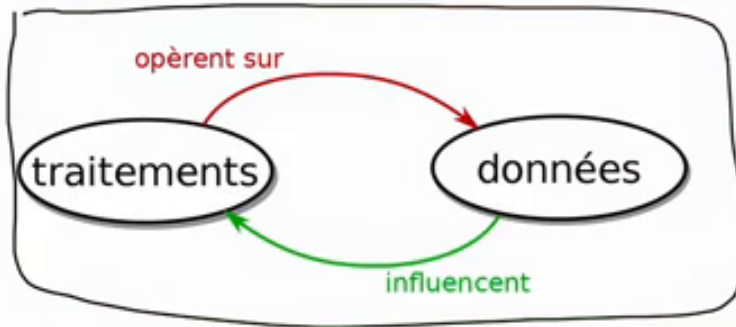




Dans les programmes que vous avez écrits jusqu'à maintenant, les notions

- ▶ de variables/types de **données**
- ▶ et de **traitement** de ces données

étaient séparées :



Or, ce lien existe conceptuellement. Ici, c'est bel et bien, la largeur et la hauteur d'un rectangle, que je suis en train de manipuler Et ici, c'est bel et bien la surface d'un rectangle, que je souhaite calculer. Donc le fait, de pouvoir regrouper en une seule et même entité, la notion de rectangle. Les données caractéristiques du rectangle, comme sa largeur et sa hauteur, ainsi que les traitements qui lui sont spécifiquement associés, vont me permettre d'établir un lien explicite entre ces différentes entités et de donner beaucoup plus de clarté à mon programme. Il s'agit de l'un des fondamentaux de la programmation orientée objet. Ce qu'il faut savoir de façon globale, c'est que la programmation orientée objet va vous donner

notes

résumé

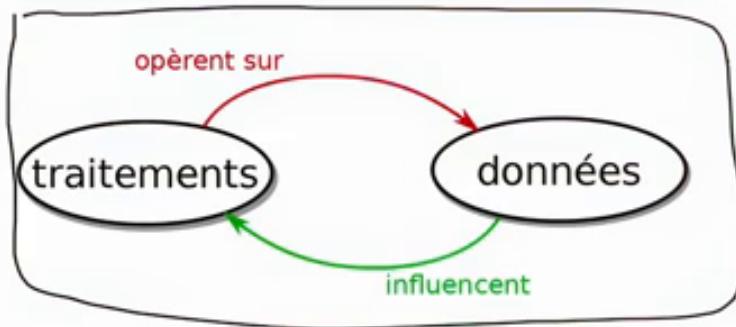
3m 37s



Dans les programmes que vous avez écrits jusqu'à maintenant, les notions

- ▶ de variables/types de **données**
- ▶ et de **traitement** de ces données

étaient séparées :



Robustesse  
- changement  
- erreur de manipulation

modularité

lisibilité

→ maintenabilité

un certain nombre d'outillages permettant davantage de robustesse, de modularité, de lisibilité à vos programmes. Ce qui va dans le sens d'une meilleure maintenabilité. Alors, robustesse par rapport au changement: si votre programme est amené à changer, à être étendu, on ne veut pas être dans l'obligation de tout réécrire. Et robustesse face aux erreurs de manipulation, par exemple, des données. En effet, la plupart des applications développées de nos jours, ne sont pas redéveloppées depuis zéro, mais consistent à étendre, à maintenir du code existant. Et il est important de pouvoir le faire à moindre coût. Nous allons voir que les propriétés fondamentales de l'orientée objet

notes

résumé

4m 25s



Un des objectifs principaux de la notion d'**objet** :

**organiser** des programmes complexes

grâce aux notions :

- ▶ d'encapsulation
- ▶ d'abstraction
- ▶ d'héritage
- ▶ et de polymorphisme

à savoir davantage de robustesse, de modularité et de lisibilité vont exactement dans ce sens là. La programmation orientée objet offre, en réalité, quatre concepts centraux : encapsulation, abstraction, héritage et polymorphisme. qui permettent de mieux organiser les programmes dans le sens de la robustesse, de lisibilité, de modularité et maintenabilité, comme je l'ai exprimé tout à l'heure. Ces concepts centraux ne sont pas spécifiques à un langage, il s'agit des concepts centraux de l'orientée objet. Dans la séquence présente, nous allons essentiellement nous occuper de définir encapsulation et abstraction. Nous aborderons dans les séquences suivantes, les notions fondamentales d'héritage et de polymorphisme. d'héritage et de polymorphisme.

notes

résumé

5m 13s

