Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Introduction (partie 1)

Concepts (extraits des sous-titres générés automatiquement) :

Score du premier joueur. Nom de la variable score. Score du deuxième joueur. Fonctions. Appel de fonction. Nombre de points. Code suivant. Corps de ma future fonction. Nom de ma future fonction. Objectif de cette nouvelle leçon. Variable score. Fonction saisie. Règle générale. Séquence linéaire d'instructions. Bon langage de programmation.

vers la recherche de séquences vidéo
(dans Initiation à la programmation (en C++).)

vers la vidéo

EPFL

Fonctions : introduction
(Partie 1)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

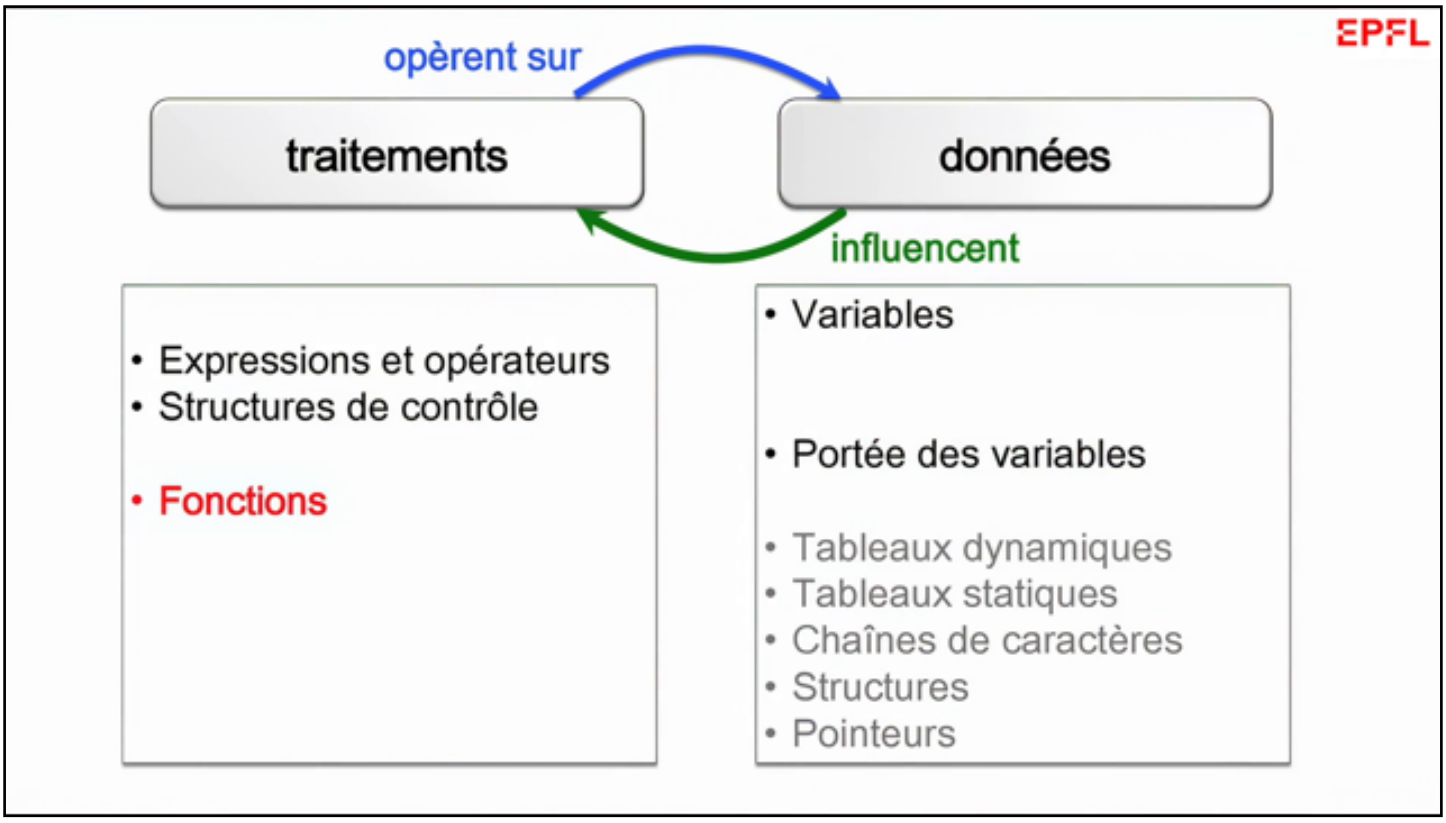
0m 0s

Bonjour ! Bienvenue à cette nouvelle leçon

notes

résumé

0m 1s



de notre cours d'introduction à la programmation en C++. L'objectif de cette nouvelle leçon est de vous apprendre à organiser votre code en le modularisant à l'aide de ce qu'on appelle des fonctions.

notes

résumé

0m 5s

Les fonctions font partie des aspects de traitement. au même titre que les expressions et opérateurs, et les structures de contrôle, que nous avons vues précédemment. En règle générale, les fonctions opèrent sur des données, et en retour les données vont influencer nos fonctions.

notes

résumé

0m 16s

Notion de réutilisabilité

EPFL

Si une tâche, par exemple :

```
do {
  cout << "Entrez le nombre de points du joueur : ";
  cin >> nb;
} while ((nb < 0) or (nb > 100));
```

doit être exécutée à *plusieurs* endroits dans un plus gros programme

☞ recopie ?

Bonne pratique : Ne *jamais dupliquer* de code en programmant :

Jamais de « copier-coller » !

☞ Ce que vous voudriez recopier doit être mis dans une **fonction**

Jusqu'à maintenant les programmes que vous avez écrits étaient constitués d'une séquence linéaire d'instructions sans organisation plus globale et sans partage des tâches répétées. Par exemple, si la tâche consistant à demander un nombre à l'utilisateur, comme le fait le code suivant, devait être exécuté plusieurs fois dans notre programme. Par exemple pour demander une fois un nombre de points, une fois un temps, une fois un âge. Que feriez-vous ? Vous pourriez être tenté de recopier le code autant de fois que nécessaire aux endroits appropriés, mais, évidemment, ceci est une très mauvaise solution. Il ne faut jamais dupliquer de code en programmant.

notes

résumé

0m 35s

Notion de réutilisabilité (2)

Pourquoi ne jamais dupliquer du code (copier/coller) :

Cela rend le programme

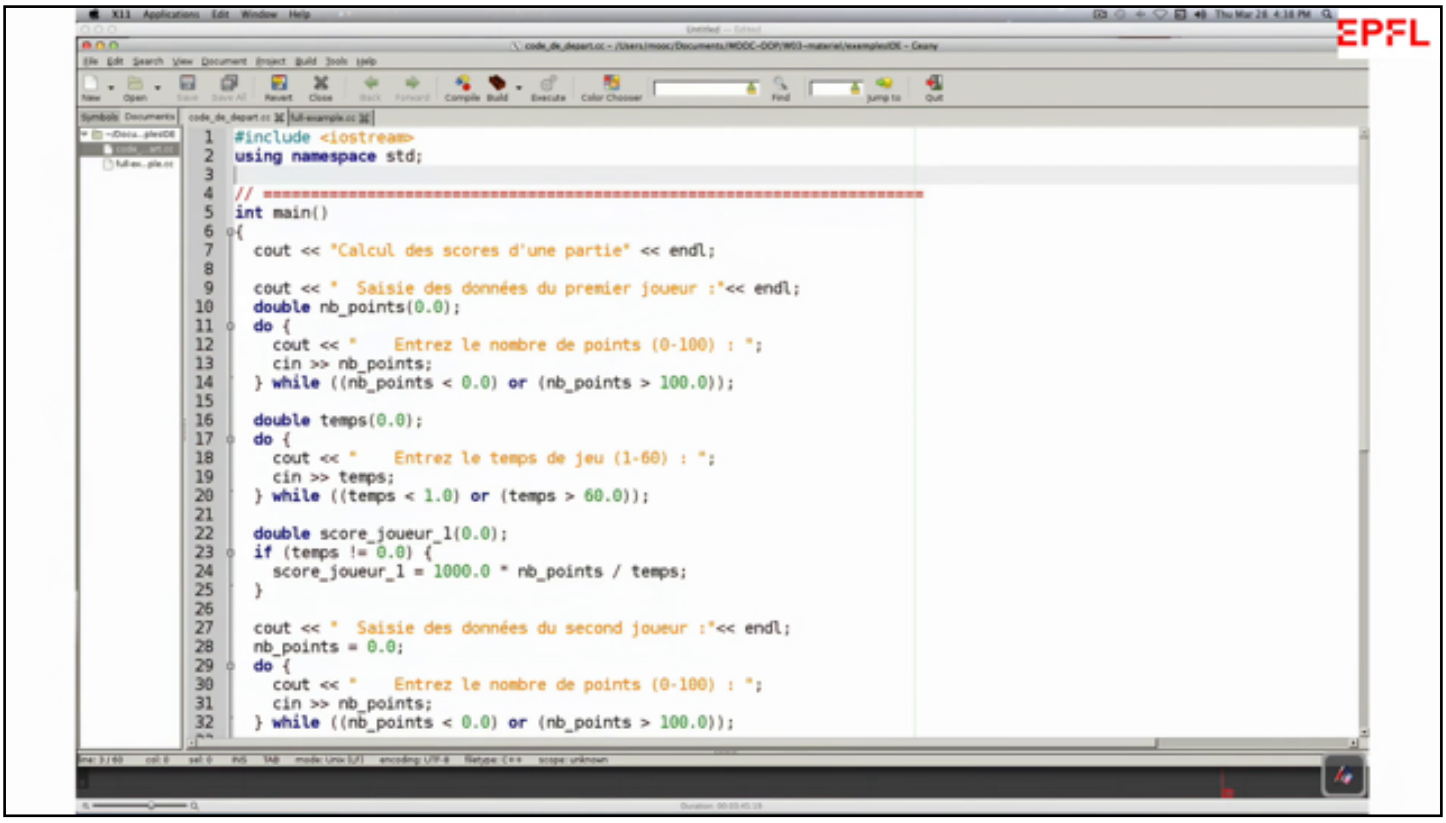
- inutilement long

Ne soyez pas tenté par le copier-coller. Ce que vous voudriez recopier doit être mis dans une fonction.

notes

résumé

1m 13s



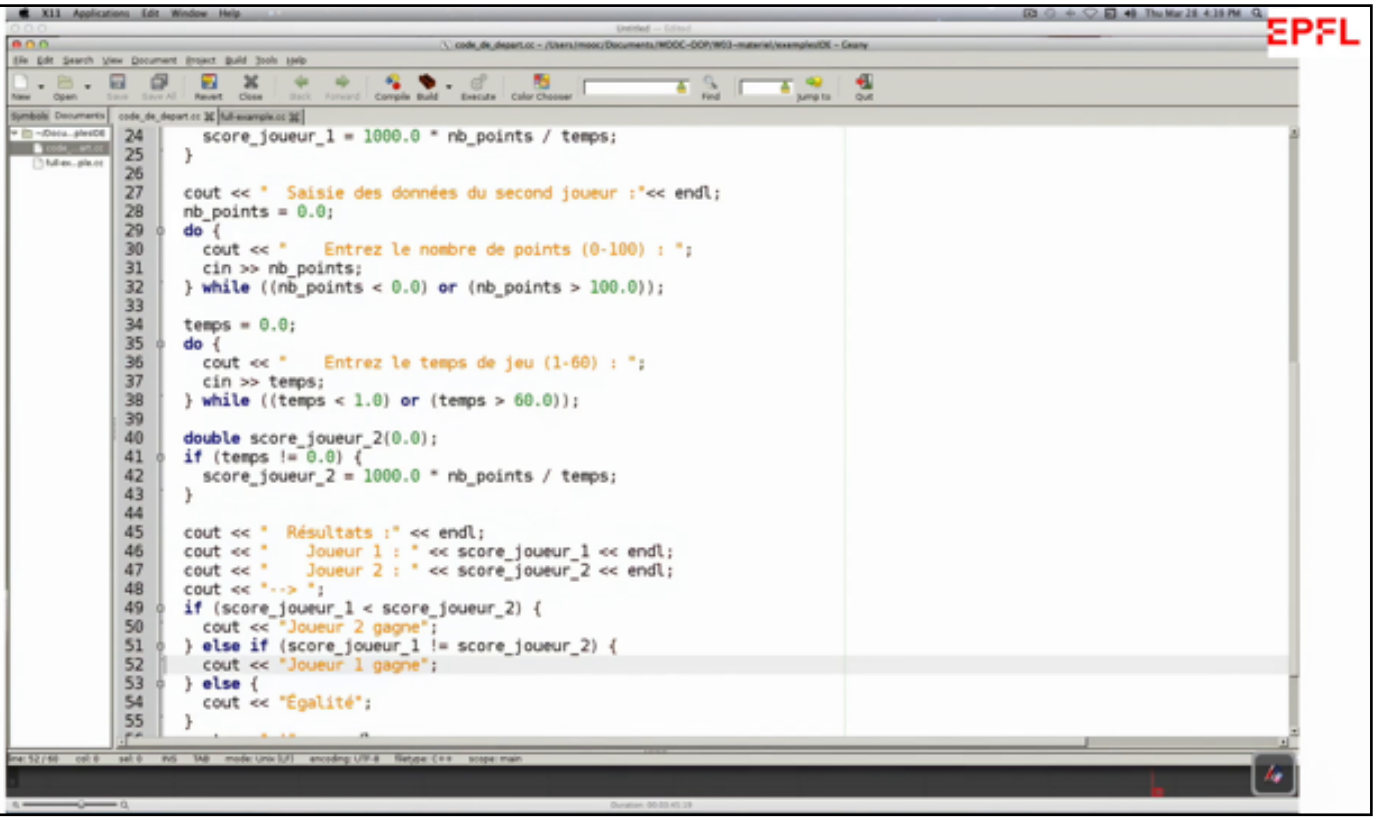
```cpp
#include <iostream>
using namespace std;

// ======================================================================
int main()
{
  cout << "Calcul des scores d'une partie" << endl;

  cout << "  Saisie des données du premier joueur :"<< endl;
  double nb_points(0.0);
  do {
    cout << "    Entrez le nombre de points (0-100) : ";
    cin >> nb_points;
  } while ((nb_points < 0.0) or (nb_points > 100.0));

  double temps(0.0);
  do {
    cout << "    Entrez le temps de jeu (1-60) : ";
    cin >> temps;
  } while ((temps < 1.0) or (temps > 60.0));

  double score_joueur_1(0.0);
  if (temps != 0.0) {
    score_joueur_1 = 1000.0 * nb_points / temps;
  }

  cout << "  Saisie des données du second joueur :"<< endl;
  nb_points = 0.0;
  do {
    cout << "    Entrez le nombre de points (0-100) : ";
    cin >> nb_points;
  } while ((nb_points < 0.0) or (nb_points > 100.0));
```

Pourquoi ne faut-il jamais dupliquer du code ? Cela rend le programme inutilement long, difficile à comprendre, et surtout difficile à maintenir. Il faudrait reporter chaque modification dans chacune des copies. C'est pour ça que tout bon langage de programmation fournit des moyens pour permettre la réutilisation de portions de programme. En C++, ça se fait en utilisant des fonctions.

notes

résumé

1m 20s



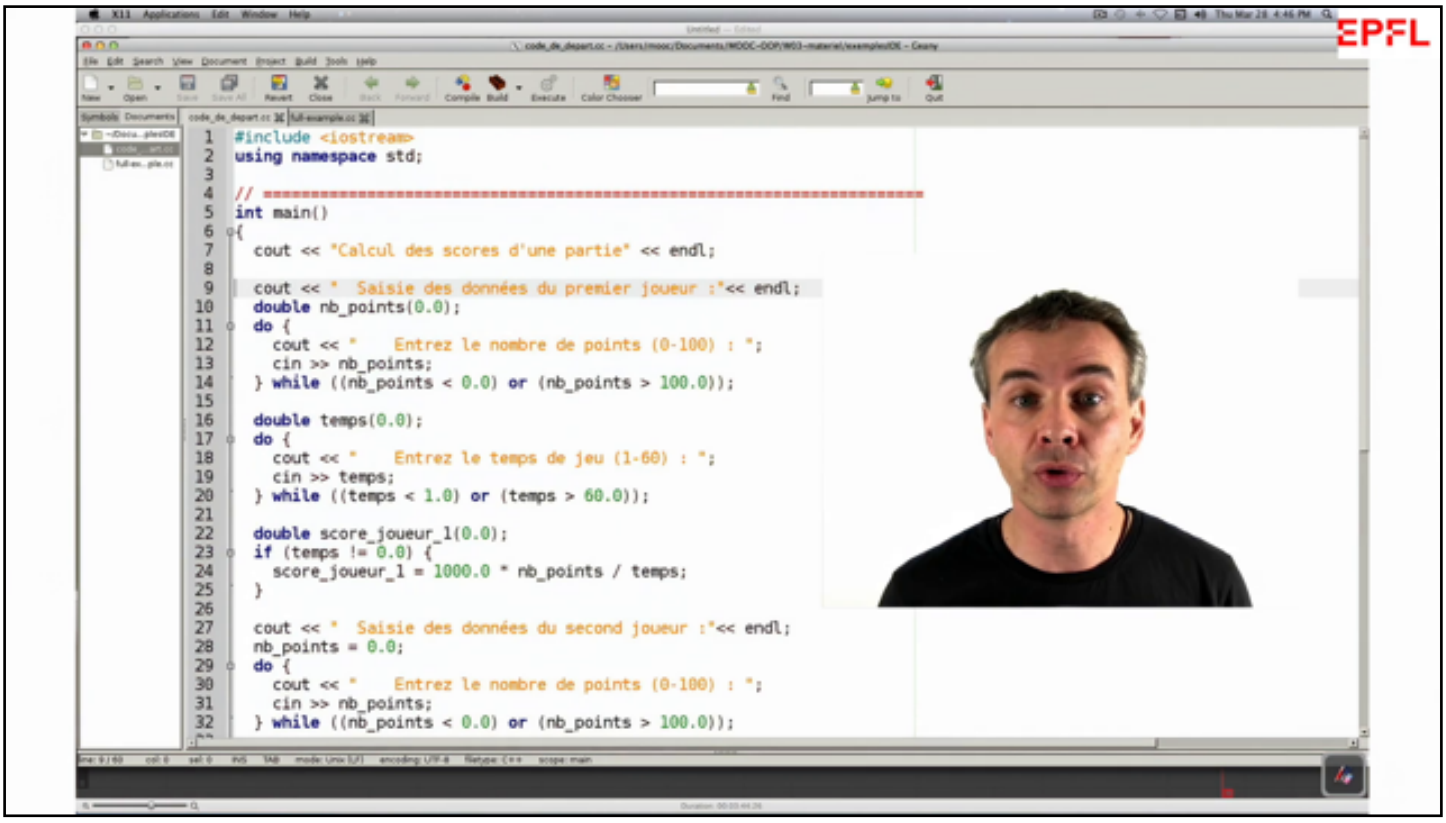
```
24    score_joueur_1 = 1000.0 * nb_points / temps;
25  }
26
27  cout << "  Saisie des données du second joueur :"<< endl;
28  nb_points = 0.0;
29  do {
30    cout << "    Entrez le nombre de points (0-100) : ";
31    cin >> nb_points;
32  } while ((nb_points < 0.0) or (nb_points > 100.0));
33
34  temps = 0.0;
35  do {
36    cout << "    Entrez le temps de jeu (1-60) : ";
37    cin >> temps;
38  } while ((temps < 1.0) or (temps > 60.0));
39
40  double score_joueur_2(0.0);
41  if (temps != 0.0) {
42    score_joueur_2 = 1000.0 * nb_points / temps;
43  }
44
45  cout << "  Résultats :" << endl;
46  cout << "    Joueur 1 : " << score_joueur_1 << endl;
47  cout << "    Joueur 2 : " << score_joueur_2 << endl;
48  cout << "--> ";
49  if (score_joueur_1 < score_joueur_2) {
50    cout << "Joueur 2 gagne";
51  } else if (score_joueur_1 != score_joueur_2) {
52    cout << "Joueur 1 gagne";
53  } else {
54    cout << "Égalité";
55  }
```

Considérons par exemple le programme suivant : Ce programme commence par demander un nombre de points obtenus par un joueur. Vous pouvez constater qu'il utilise, pour cela, une boucle do / while pour forcer l'utilisateur à entrer une valeur entre 0 et 100. Ensuite, le programme continue en demandant à l'utilisateur d'entrer le temps utilisé par le joueur. Cette fois-ci, c'est une valeur entre 1 et 60. Puis à partir du nombre de points obtenus par le joueur et du temps qu'il a utilisé, le programme calcule le score du premier joueur. Ensuite, le programme répète ces opérations pour un deuxième joueur. Il demande un nombre de points, un temps, et calcule le score du deuxième joueur. Finalement, le programme compare le score des deux joueurs,

notes

résumé

1m 47s

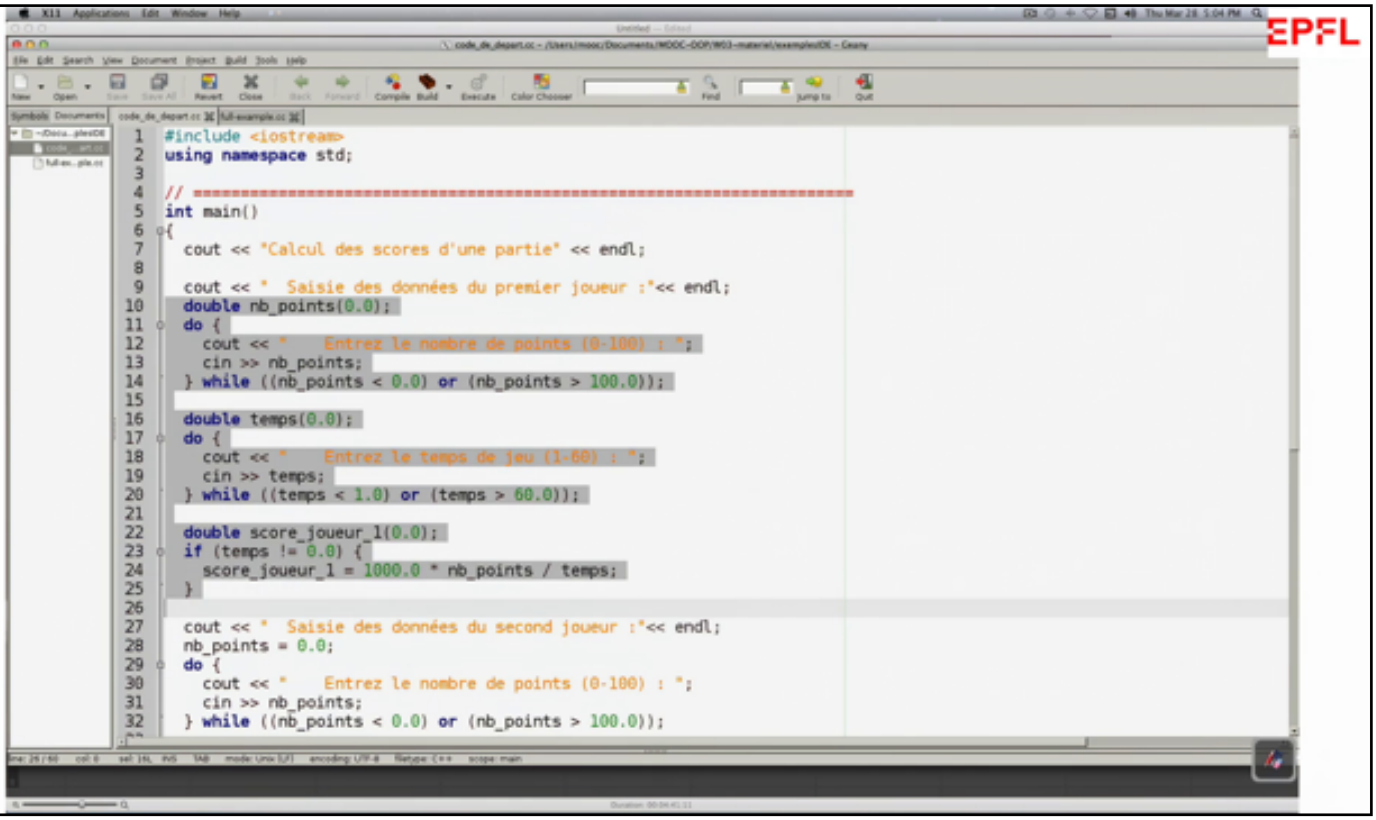


pour pouvoir annoncer, si c'est le joueur 1 ou le joueur 2 qui a gagné. Vous pouvez donc constater que dans ce programme,

notes

résumé

2m 44s



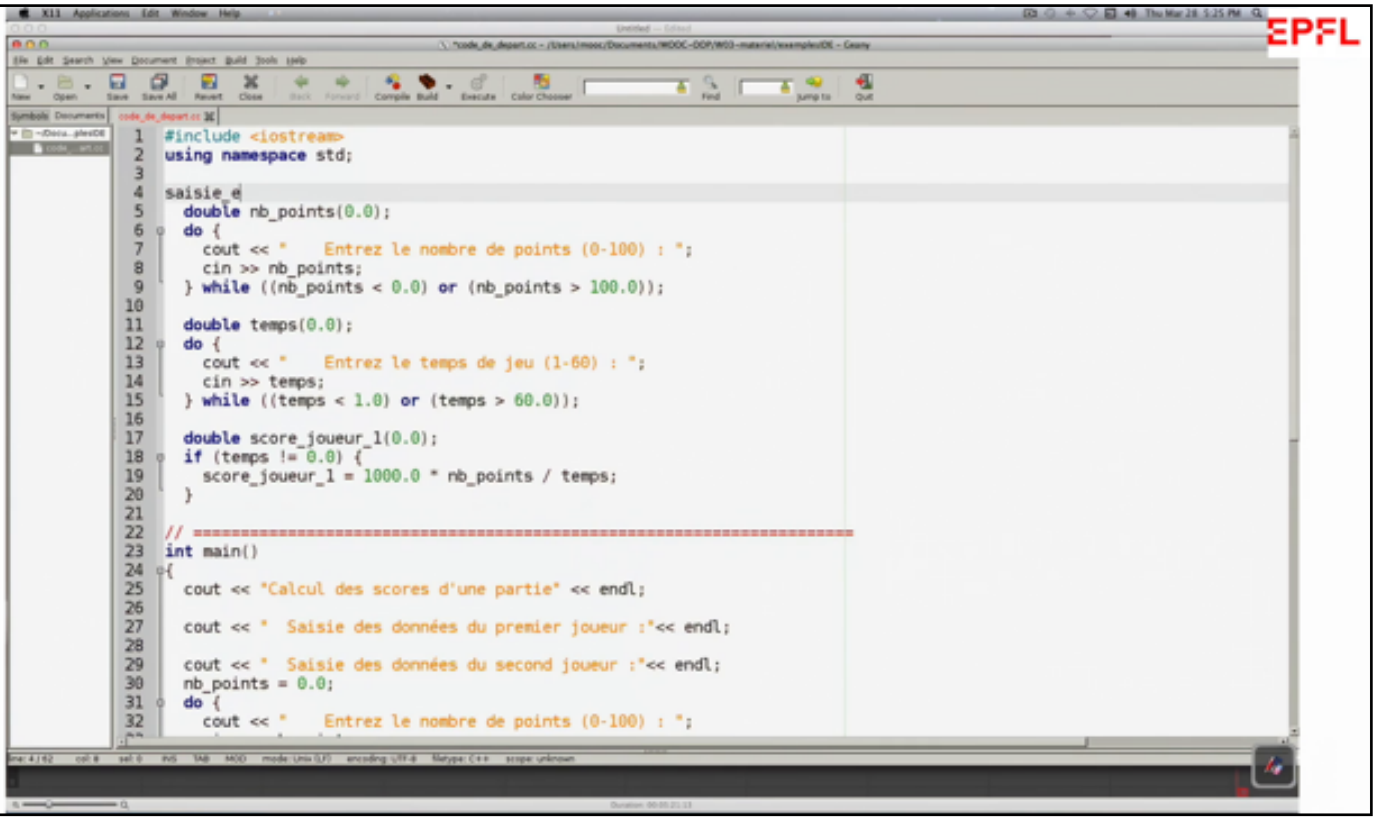
```
#include <iostream>
using namespace std;

// ======================================================================
int main()
{
  cout << "Calcul des scores d'une partie" << endl;

  cout << "  Saisie des données du premier joueur :"<< endl;
  double nb_points(0.0);
  do {
    cout << "    Entrez le nombre de points (0-100) : ";
    cin >> nb_points;
  } while ((nb_points < 0.0) or (nb_points > 100.0));

  double temps(0.0);
  do {
    cout << "    Entrez le temps de jeu (1-60) : ";
    cin >> temps;
  } while ((temps < 1.0) or (temps > 60.0));

  double score_joueur_1(0.0);
  if (temps != 0.0) {
    score_joueur_1 = 1000.0 * nb_points / temps;
  }

  cout << "  Saisie des données du second joueur :"<< endl;
  nb_points = 0.0;
  do {
    cout << "    Entrez le nombre de points (0-100) : ";
    cin >> nb_points;
  } while ((nb_points < 0.0) or (nb_points > 100.0));
```

le code qui consiste à demander un nombre de points, un temps, et de calculer un score à partir du nombre de points et du temps, est répété deux fois, une fois pour chacun des deux joueurs. Dans un cas comme celui-là, il faut utiliser une fonction pour éviter la duplication du code. Pour cela, je vais d'abord mettre de côté le code qui m'intéresse. Le code qui m'intéresse, c'est ici, le code qui demande un nombre de points un temps et qui calcule un score.

notes

résumé

2m 52s



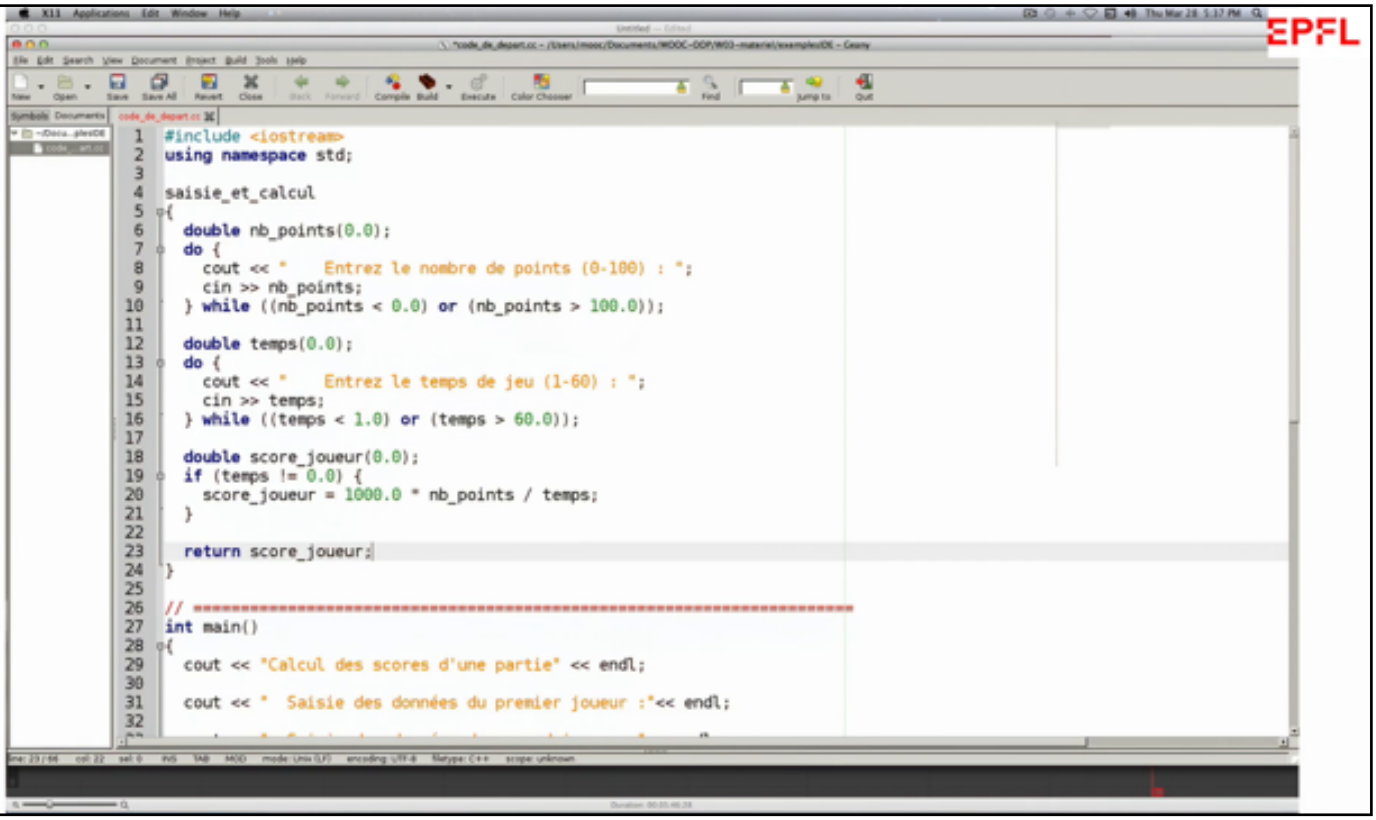
```
#include <iostream>
using namespace std;

saisie_e
  double nb_points(0.0);
  do {
    cout << "    Entrez le nombre de points (0-100) : ";
    cin >> nb_points;
  } while ((nb_points < 0.0) or (nb_points > 100.0));

  double temps(0.0);
  do {
    cout << "    Entrez le temps de jeu (1-60) : ";
    cin >> temps;
  } while ((temps < 1.0) or (temps > 60.0));

  double score_joueur_1(0.0);
  if (temps != 0.0) {
    score_joueur_1 = 1000.0 * nb_points / temps;
  }

// ========================================================================
int main()
{
  cout << "Calcul des scores d'une partie" << endl;

  cout << "  Saisie des données du premier joueur :"<< endl;

  cout << "  Saisie des données du second joueur :"<< endl;
  nb_points = 0.0;
  do {
    cout << "    Entrez le nombre de points (0-100) : ";
```

Donc je vais mettre de côté ce code. Par exemple ici. Ce code va constituer ce qu'on appelle, le corps de ma future fonction. Ce corps, il va falloir le nommer, pour pouvoir y faire référence dans le reste du programme.

notes

résumé

3m 23s



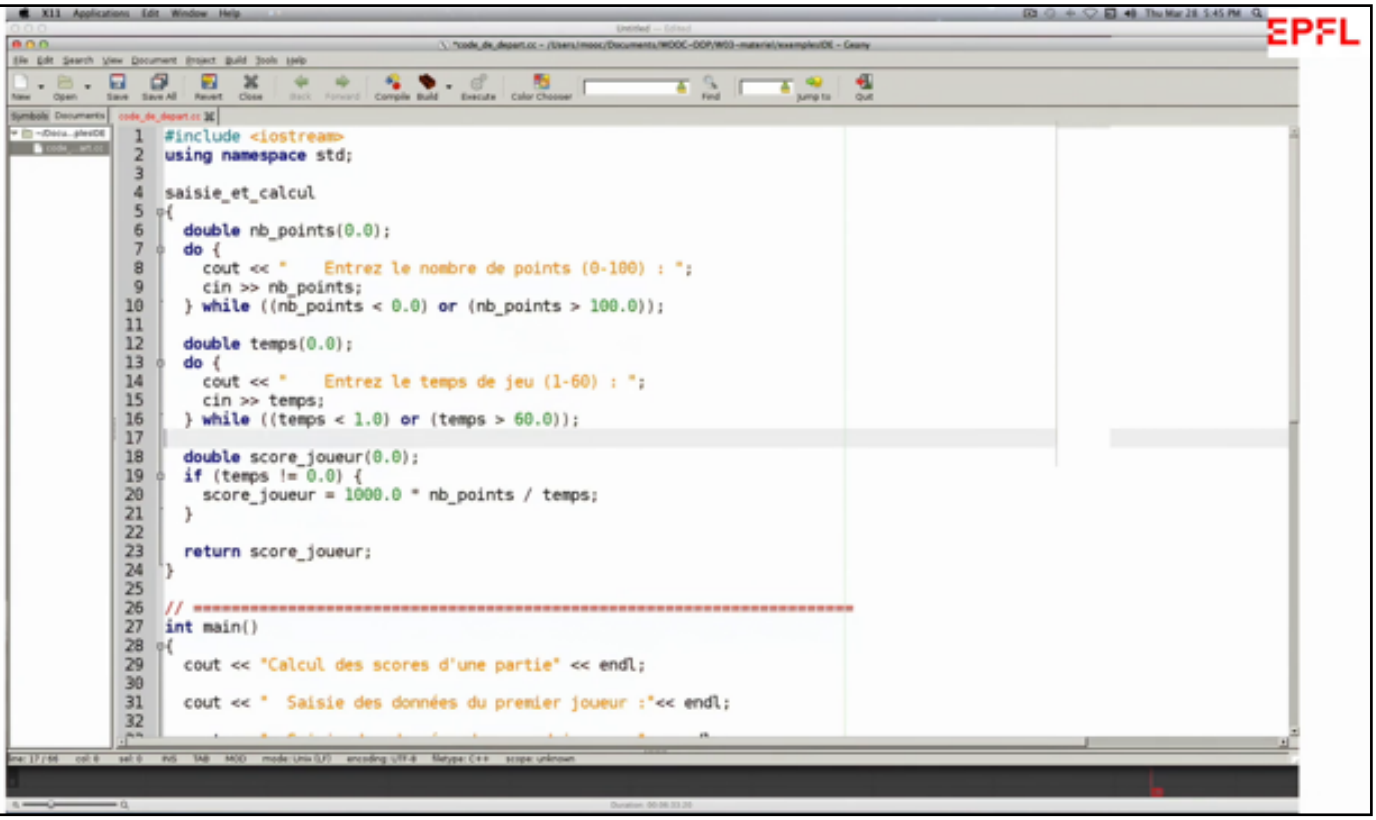
```
#include <iostream>
using namespace std;

saisie_et_calcul
{
  double nb_points(0.0);
  do {
    cout << "    Entrez le nombre de points (0-100) : ";
    cin >> nb_points;
  } while ((nb_points < 0.0) or (nb_points > 100.0));

  double temps(0.0);
  do {
    cout << "    Entrez le temps de jeu (1-60) : ";
    cin >> temps;
  } while ((temps < 1.0) or (temps > 60.0));

  double score_joueur(0.0);
  if (temps != 0.0) {
    score_joueur = 1000.0 * nb_points / temps;
  }

  return score_joueur;
}

// ========================================================================
int main()
{
  cout << "Calcul des scores d'une partie" << endl;

  cout << "  Saisie des données du premier joueur :"<< endl;
```

Je vais l'appeler ici, saisie_et_calcul et ça va devenir le nom de ma future fonction. Je veux maintenant que ma fonction puisse fournir un résultat au reste du programme. Le résultat, dans cet exemple, c'est le score du joueur qui a été calculé par la fonction. Je vais commencer par enlever les références au joueur 1, puisque ma fonction est maintenant valable pour n'importe quel joueur. Je vais utiliser le mot-clef return suivi du nom de la variable score_joueur

notes

résumé

3m 49s



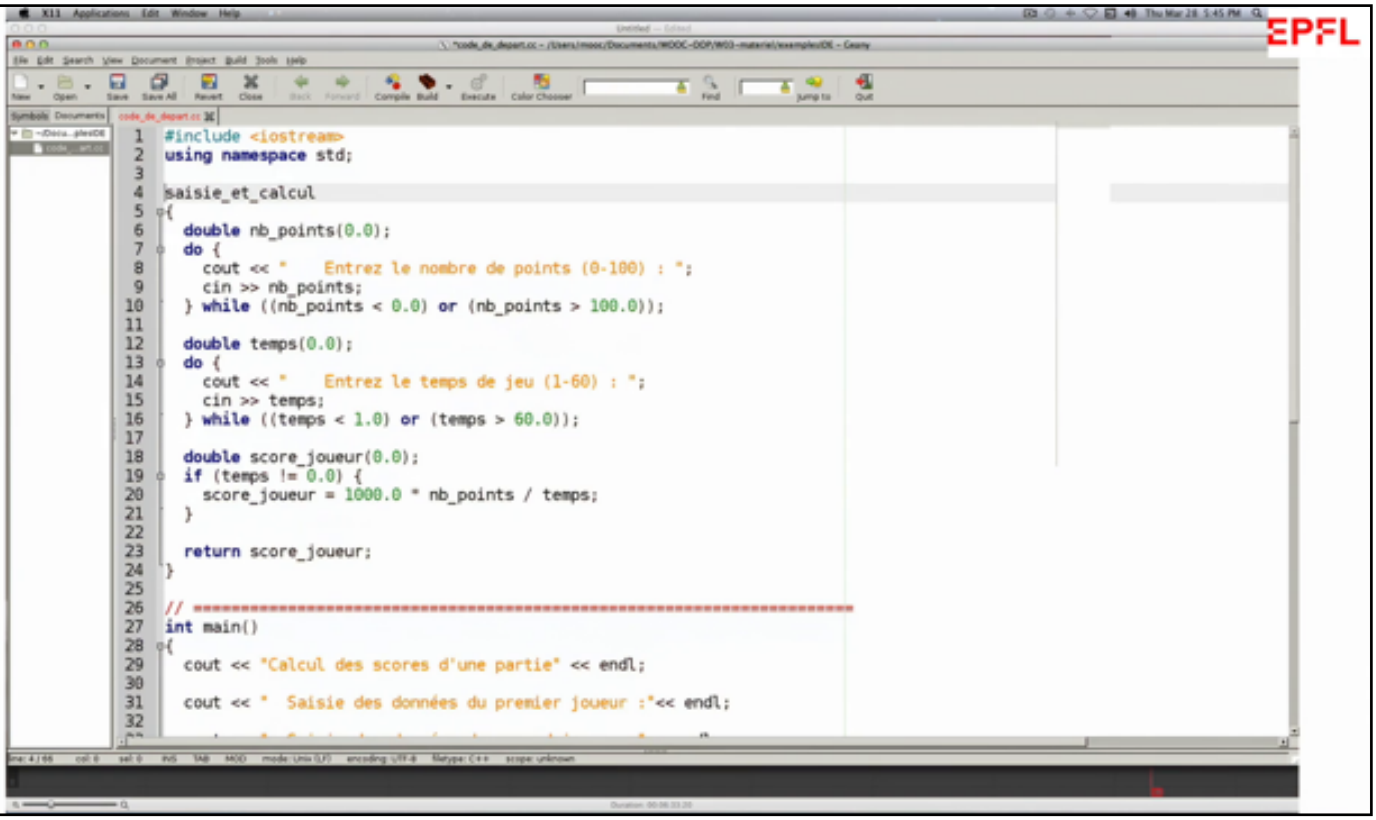
```
#include <iostream>
using namespace std;

saisie_et_calcul
{
  double nb_points(0.0);
  do {
    cout << "    Entrez le nombre de points (0-100) : ";
    cin >> nb_points;
  } while ((nb_points < 0.0) or (nb_points > 100.0));

  double temps(0.0);
  do {
    cout << "    Entrez le temps de jeu (1-60) : ";
    cin >> temps;
  } while ((temps < 1.0) or (temps > 60.0));

  double score_joueur(0.0);
  if (temps != 0.0) {
    score_joueur = 1000.0 * nb_points / temps;
  }

  return score_joueur;
}

// ========================================================================
int main()
{
  cout << "Calcul des scores d'une partie" << endl;

  cout << "  Saisie des données du premier joueur :"<< endl;
```

pour dire que ma fonction doit fournir la valeur contenue dans la variable score_joueur. Ne vous inquiétez pas, on verra ça en détail dans la suite du cours. Ma fonction peut donc fournir une valeur au reste du programme. Cette valeur a un certain type. Il faut que je signale au reste du programme quel est ce type. La valeur retournée par ma fonction c'est la valeur qui est contenue dans la variable score_joueur. score_joueur a été définie comme ayant le type double. Donc il faut que je signale que ma fonction va renvoyer une valeur de type double.

notes

résumé

4m 25s



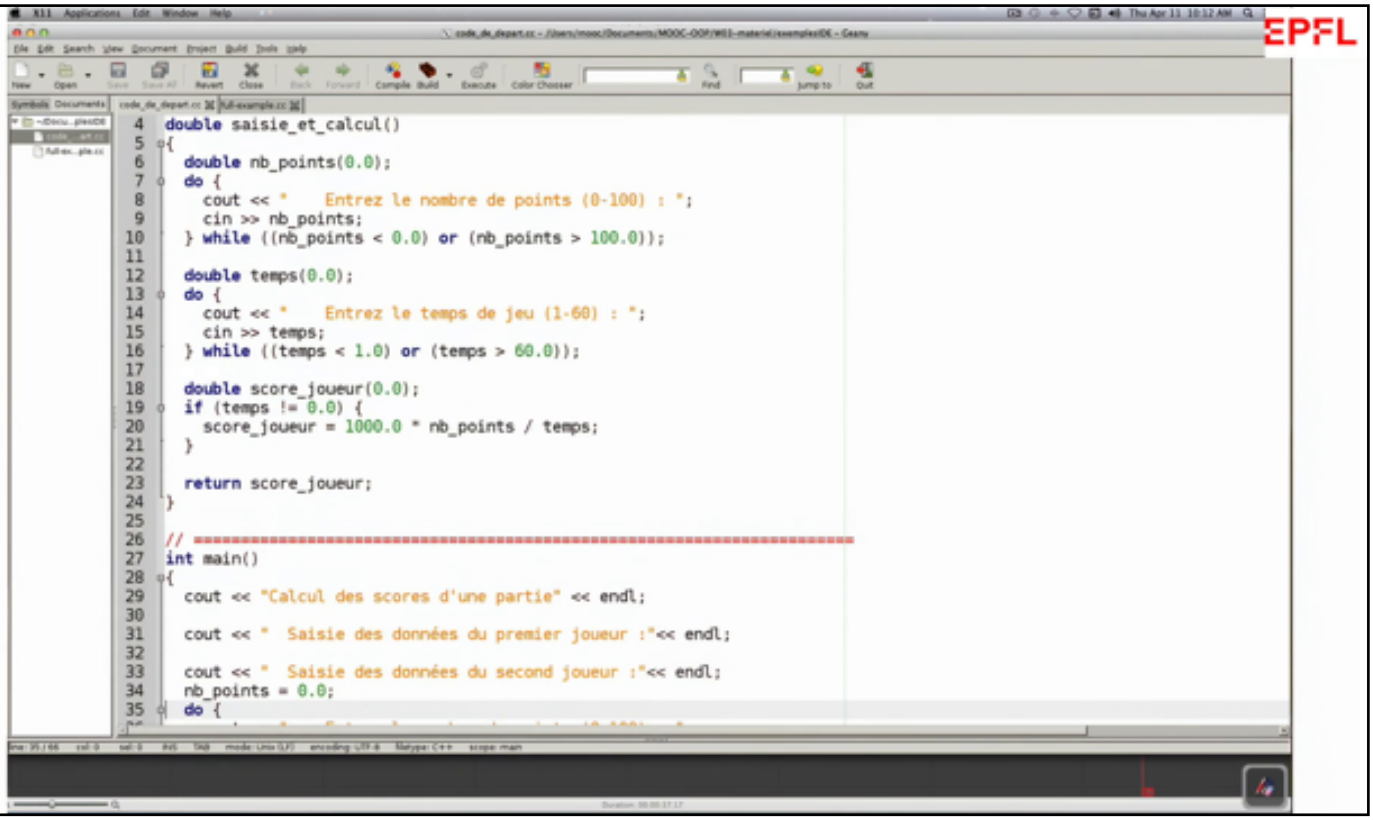
```
#include <iostream>
using namespace std;

saisie_et_calcul
{
  double nb_points(0.0);
  do {
    cout << "    Entrez le nombre de points (0-100) : ";
    cin >> nb_points;
  } while ((nb_points < 0.0) or (nb_points > 100.0));

  double temps(0.0);
  do {
    cout << "    Entrez le temps de jeu (1-60) : ";
    cin >> temps;
  } while ((temps < 1.0) or (temps > 60.0));

  double score_joueur(0.0);
  if (temps != 0.0) {
    score_joueur = 1000.0 * nb_points / temps;
  }

  return score_joueur;
}

// ========================================================================
int main()
{
  cout << "Calcul des scores d'une partie" << endl;

  cout << "  Saisie des données du premier joueur :"<< endl;
```

Et ça se fait en rajoutant, avant le nom de la fonction,

notes

résumé

5m 1s



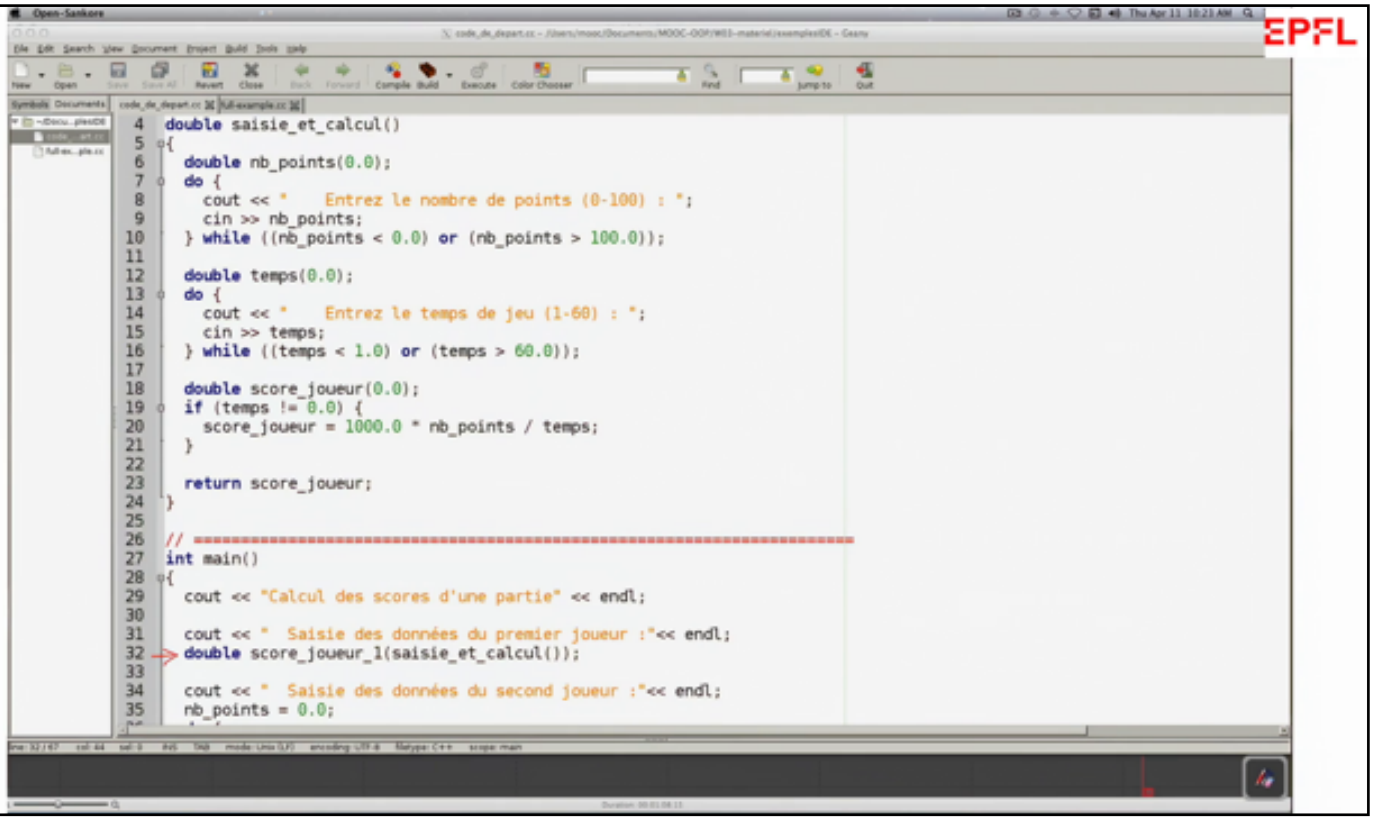
```
4  double saisie_et_calcul()
5  {
6    double nb_points(0.0);
7    do {
8      cout << "    Entrez le nombre de points (0-100) : ";
9      cin >> nb_points;
10   } while ((nb_points < 0.0) or (nb_points > 100.0));
11
12   double temps(0.0);
13   do {
14     cout << "    Entrez le temps de jeu (1-60) : ";
15     cin >> temps;
16   } while ((temps < 1.0) or (temps > 60.0));
17
18   double score_joueur(0.0);
19   if (temps != 0.0) {
20     score_joueur = 1000.0 * nb_points / temps;
21   }
22
23   return score_joueur;
24 }
25
26 // ========================================================================
27 int main()
28 {
29   cout << "Calcul des scores d'une partie" << endl;
30
31   cout << "  Saisie des données du premier joueur :"<< endl;
32
33   cout << "  Saisie des données du second joueur :"<< endl;
34   nb_points = 0.0;
35   do {
```

le type de la valeur renvoyée, qui est ici double. Un autre aspect des fonctions est qu'elles peuvent utiliser des valeurs fournies par l'extérieur pour pouvoir fonctionner. Notre fonction saisie_et_calcul est relativement simple, elle n'a pas besoin encore de telles valeurs. Pour signaler cela, il faut mettre à la fin du nom de la fonction une parenthèse ouvrante suivie d'une parenthèse fermante. Notre fonction est complète. Je vais maintenant utiliser ma fonction saisieu>et/u>calcul pour demander à l'utilisateur un nombre de points, un temps et calculer le score du joueur 1. Pour cela, je vais retourner à l'endroit où j'avais extrait le code

notes

résumé

5m 4s



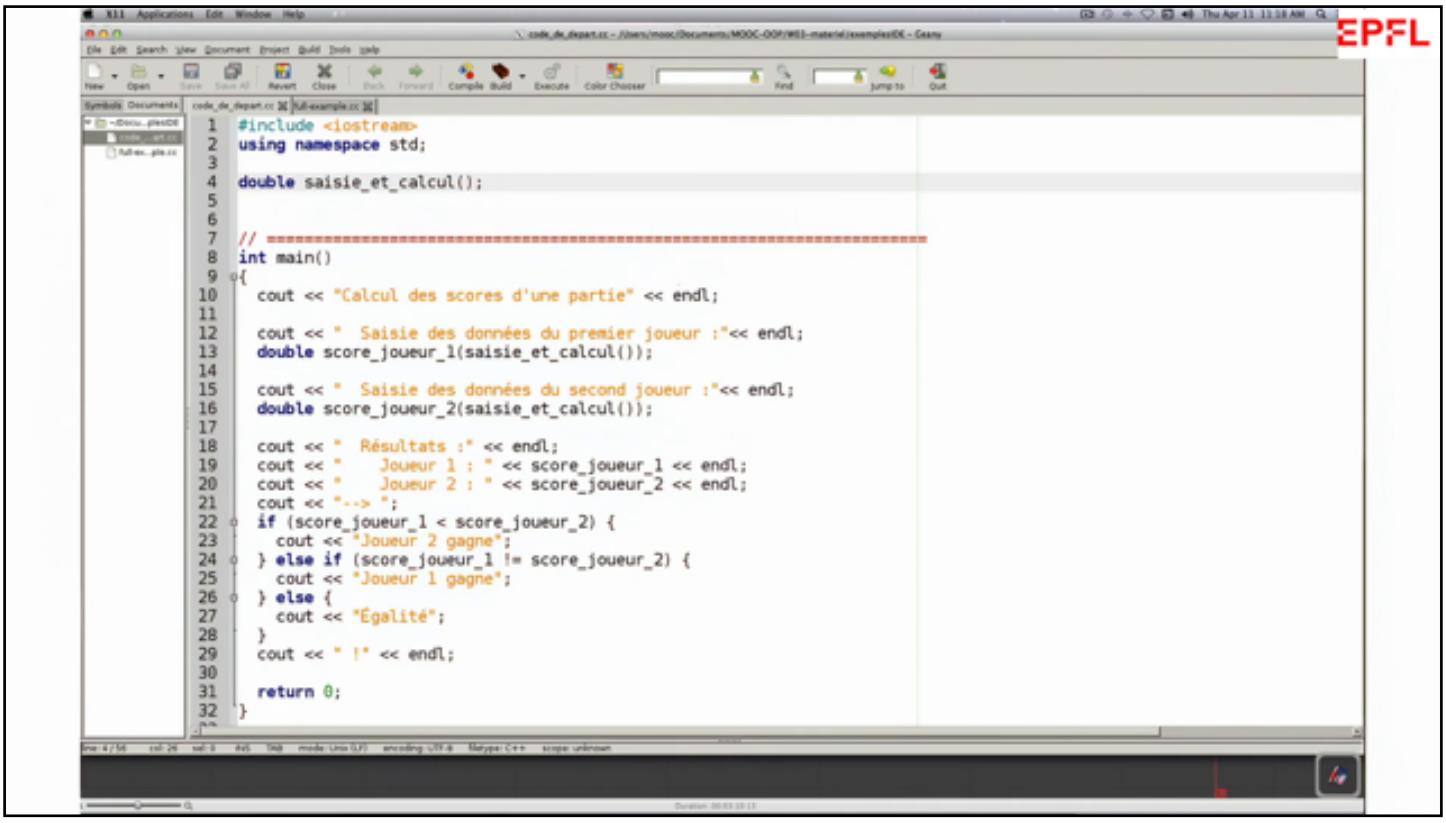
```
4  double saisie_et_calcul()
5  {
6    double nb_points(0.0);
7    do {
8      cout << "    Entrez le nombre de points (0-100) : ";
9      cin >> nb_points;
10   } while ((nb_points < 0.0) or (nb_points > 100.0));
11
12   double temps(0.0);
13   do {
14     cout << "    Entrez le temps de jeu (1-60) : ";
15     cin >> temps;
16   } while ((temps < 1.0) or (temps > 60.0));
17
18   double score_joueur(0.0);
19   if (temps != 0.0) {
20     score_joueur = 1000.0 * nb_points / temps;
21   }
22
23   return score_joueur;
24 }
25
26 // ================================================================
27 int main()
28 {
29   cout << "Calcul des scores d'une partie" << endl;
30
31   cout << "  Saisie des données du premier joueur :"<< endl;
32   double score_joueur_1(saisie_et_calcul());
33
34   cout << "  Saisie des données du second joueur :"<< endl;
35   nb_points = 0.0;
```

que j'ai utilisé pour créer ma fonction. C'est à dire, ici. Je vais déclarer une variable qui va contenir le score du joueur 1. Je vais utiliser ma fonction que j'avais appelée saisie_et_calcul et ça va s'écrire de cette façon là. L'instruction que je viens de créer, ici, est donc une déclaration de variable

notes

résumé

5m 48s

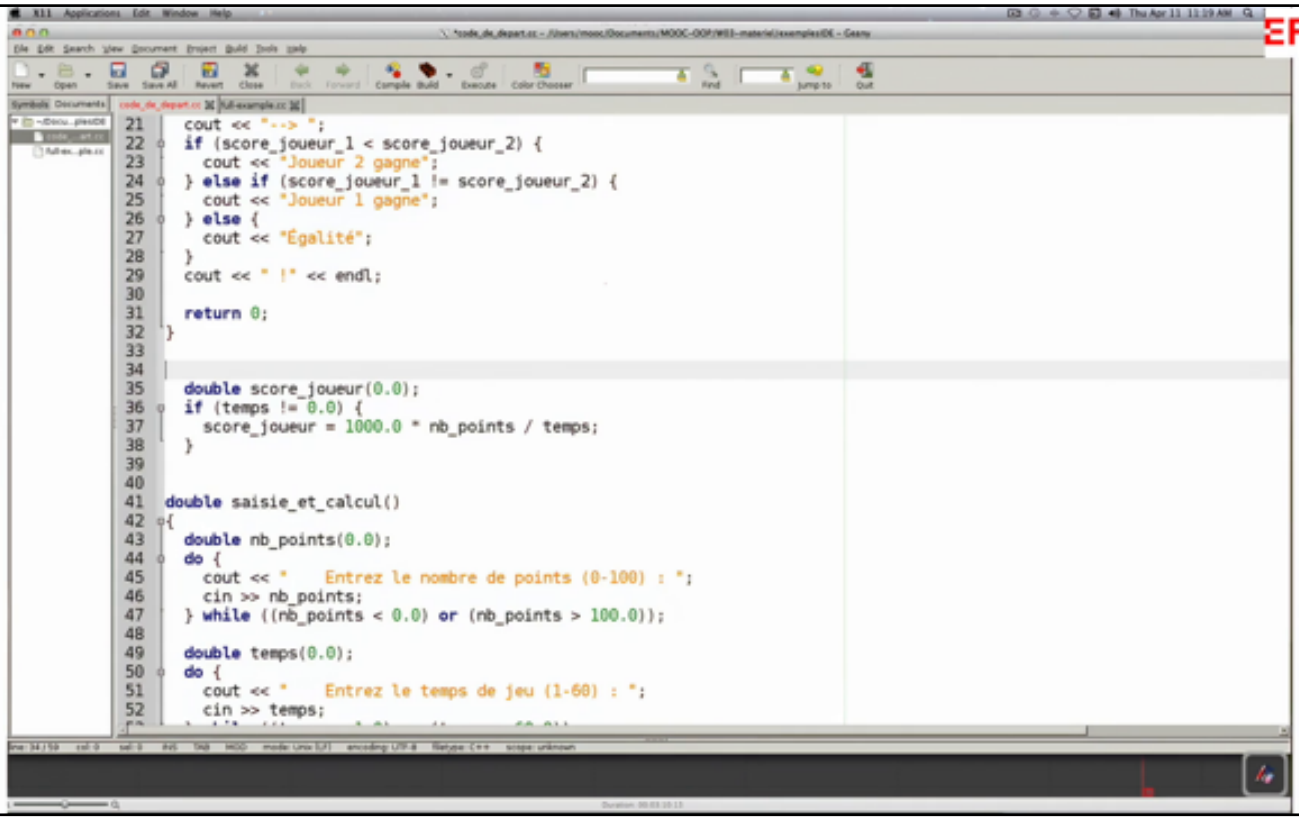


de la variable score_joueur1 et j'initialise cette variable en utilisant la fonction saisie_et_calcul. Cette utilisation s'appelle un appel de fonction. L'appel va fournir le résultat de l'exécution de la fonction. C'est à dire, ici, le score du joueur. Cette valeur peut, ensuite, par exemple, être stockée dans une variable comme c'est le cas ici. Je peux faire exactement la même chose pour le second joueur. Je peux enlever le code qui correspond à ma fonction c'est à dire celui-ci, et le remplacer par un appel à ma fonction saisie_et_calcul pour initialiser la variable score_joueur2 Vous pouvez remarquer que le code maintenant est beaucoup plus concis. Pour l'instant, ma fonction saisie_et_calcul est définie au début du programme. C'est à dire, ici. Mais je pourrais la définir à un autre endroit du programme, par exemple à la fin. Comme ceci. Mais si je veux utiliser ma fonction pour initialiser les variables score_joueur1 et score_joueur2, il faut que le compilateur connaisse cette fonction au moment de l'appel de la fonction. Pour pouvoir déclarer ma fonction avant son appel je vais juste répéter la première ligne de la définition de ma fonction. Mettre cette première ligne avant l'appel de la fonction quelque part ici, et terminer cette ligne avec un point virgule. De cette façon, maintenant, au moment de l'appel de ma fonction, le compilateur connaît déjà le nom de la fonction et le type du résultat de la fonction, c'est à dire double ici. Cette ligne qui déclare une fonction s'appelle un prototype de fonction. Créer la fonction saisie_et_calcul m'a permis d'éviter de dupliquer du code.

notes

résumé

6m 13s



```
  cout << "--> ";
  if (score_joueur_1 < score_joueur_2) {
    cout << "Joueur 2 gagne";
  } else if (score_joueur_1 != score_joueur_2) {
    cout << "Joueur 1 gagne";
  } else {
    cout << "Égalité";
  }
  cout << " !" << endl;

  return 0;
}

  double score_joueur(0.0);
  if (temps != 0.0) {
    score_joueur = 1000.0 * nb_points / temps;
  }

double saisie_et_calcul()
{
  double nb_points(0.0);
  do {
    cout << "    Entrez le nombre de points (0-100) : ";
    cin >> nb_points;
  } while ((nb_points < 0.0) or (nb_points > 100.0));

  double temps(0.0);
  do {
    cout << "    Entrez le temps de jeu (1-60) : ";
    cin >> temps;
```

C'est aussi intéressant d'utiliser une fonction pour pouvoir se concentrer sur un aspect un peu difficile du programme, comme par exemple, ici, le calcul du score du joueur. Donc je vais extraire, comme avant, le code qui calcule le score du joueur.

notes

résumé

8m 27s

Puis à partir de là, je vais créer une fonction de la même façon que pour ma fonction saisie_et_calcul. Cette nouvelle fonction va renvoyer le score du joueur, donc une valeur de type double. Je vais appeler cette nouvelle fonction, tout simplement, score et pour l'instant je vais faire suivre le nom de la fonction par une parenthèse ouvrante et une parenthèse fermante. Je vais mettre le code que je viens d'extraire entre accolades. Comme avant, je vais utiliser le mot clef return pour dire que ma fonction doit fournir le score du joueur. Ça va s'écrire de cette façon-là. La différence par rapport à la première fonction saisieu>et/u>calcul est que cette nouvelle fonction score a besoin de la variable nb_points et de la variable temps pour pouvoir calculer le score du joueur. Pour cela je vais rajouter entre parenthèses ces deux variables. Ça va se noter de cette façon-ci. Ce que je viens de rajouter entre parenthèses, ici, ça s'appelle des paramètres. Ce seront les paramètres de ma fonction. de ma fonction.

notes

résumé

8m 49s