Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Définitions (partie 4)

Concepts (extraits des sous-titres générés automatiquement) :

Cas particuliers de fonctions. Variable n. Version de départ du programme. Boucle do while. Fonction mathématique. Prototype particulier. Ensemble d'instructions. Corps de la fonction. Besoin d'une déclaration particulière. Point virgule. Reste du programme. Telle fonction. Déclaration particulière de son type. Bloc do while. Bas du transparent.

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

vers la vidéo

EPFL

Fonctions : définitions
(Partie 4)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

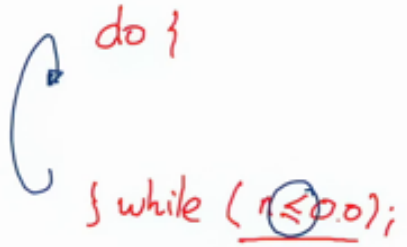
0m 0s

Remarques sur l'instruction `return` (4/4)

EPFL

Le compilateur doit être sûr de toujours pouvoir exécuter un `return`:

```
double lire() {
  cout << "Entrez un nombre : ";
  double n(0.0);
  cin >> n;
  if (n > 0.0) {
    return n;
  }
  // Erreur : pas de return si n <= 0 !
}
```



Plus précisément, puisque l'on veut d'abord demander à l'utilisateur d'entrer une valeur avant de tester si elle est positive ou négative, il s'agit ici bien sûr d'une boucle do while. On va donc écrire la boucle do while ici avec comme condition, on va demander la question tant qu'on n'a pas la bonne réponse. On voulait dans la version de départ du programme que n soit positif pour avoir une bonne réponse. Donc, on va répéter while tant que n est cette fois-ci négatif, tant que n est négatif,

notes

résumé

0m 1s

il faut répéter la boucle et dès que n sera positif, on pourra retourner la valeur, donc on peut déjà écrire ici le return. Ensuite, dans la boucle, on va saisir la réponse de l'utilisateur commencer par lui écrire la question, et ensuite saisir sa réponse c'est ça qui va faire l'objet d'une boucle. On l'écrit ici, le message et puis la lecture dans la variable n. Reste maintenant à déclarer cette variable n comme on l'utilise ici dans la condition du while qui est hors du bloc do while et qu'on l'utilise ici aussi en retour, on va déclarer la variable hors du bloc pour pouvoir l'utiliser hors du bloc, donc, on va la déclarer ici, au départ l'initialiser avec une valeur qui fait sens, donc, par exemple 0. Et voici donc, la formulation précise qui corrige l'erreur précédente. Voyons maintenant quelques cas particuliers de fonctions.

notes

résumé

0m 49s

Fonctions sans valeur de retour

EPFL

Quand une fonction ne doit fournir aucun résultat : procédure

☞ définir une fonction sans valeur de retour

On utilise alors le type particulier `void` comme type de retour.

Dans ce cas la commande de retour `return` est optionnelle :

- soit on ne place aucun `return` dans le corps de la fonction
- soit on utilise l'instruction `return` sans la faire suivre d'une expression : `return;`

Exemple :

```
void affiche_racine(double a)
{
  if (a < 0.0) {
    return; /* Grâce à ce return, on quitte la fonction avant *
             * de calculer sqrt(a) si a est négatif.          */
  }
  cout << sqrt(a);
  // il n'est pas nécessaire de mettre un return ici
}
```

On va commencer par exemple par une fonction qui n'a pas de valeur de retour. Je dois vous rappeler qu'une fonction en programmation n'a rien à voir avec une fonction mathématique. Une fonction en programmation, c'est simplement un ensemble d'instructions que l'on a réservé de côté, que l'on a nommé. Cet ensemble d'instructions peut tout à fait n'avoir aucune valeur de retour n'avoir aucun résultat à fournir au reste du programme. Une telle fonction, c'est ce que l'on appelle parfois une procédure. Une fonction comme ceci va avoir besoin d'une déclaration particulière. La déclaration particulière de son type, puisqu'elle ne renvoie rien, sera d'utiliser le mot réservé void pour indiquer qu'elle n'a pas de type de retour. Si je prends l'exemple ici en bas du transparent, la fonction affichée racine va simplement afficher quelque chose sur le terminal et elle n'a pas besoin de retour et elle n'a pas besoin de retourner quelque chose au reste du programme. On va définir son type de retour entre guillemets, indiquer qu'il n'y a pas de retour par le mot réservé void. Comme une telle fonction ne retourne rien au reste du programme, elle n'a pas besoin d'instructions return et l'instruction return dans de telles fonctions est optionnelle. Cependant, il pourrait être utile de devoir arrêter préalablement le corps de la fonction, en fonction par exemple d'un if. A ce moment-là, on peut tout à fait mettre un return dans le corps de la fonction et le return est écrit avec un point virgule derrière. Si je reprends l'exemple de la fonction affiche racine ici en bas du transparent, qui retourne donc rien du tout puisqu'elle fait simplement afficher sur l'écran la racine carré de l'argument reçu. Si a est ici négatif, alors il faut simplement ne rien afficher puisque la racine carré d'un nombre négatif n'est pas définie

notes

résumé

1m 49s

Fonctions sans paramètre

EPFL

Il est aussi possible de définir des fonctions sans paramètre.
Il suffit, dans le prototype et la définition, d'utiliser une liste de paramètres vide : ()

Exemple :

```
double saisie()
{
  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));

  return nb_points;
}
```

et donc on va provoquer ici l'arrêt de l'exécution du corps de la fonction par une expression return qui ne contient absolument aucune expression à renvoyer, qui ne contient rien derrière lui. Si a, par exemple, était positif, le if ne s'exécute pas et le corps de la fonction continue normalement ici en exécutant l'affichage de la racine carré. Autre cas particulier de fonction, les fonctions sans paramètre. On peut tout à fait définir des fonctions qui n'ont besoin d'aucun paramètre de l'extérieur pour fonctionner pour faire ce qu'elle doit faire. On a déjà vu cela dans le cours sur les prototypes, il suffit alors simplement dans l'entête de mettre une liste de paramètres vide, c'est simplement deux parenthèses, comme ceci () sans rien à l'intérieur. Par exemple, si je veux une fonction saisie qui dont le but est de demander un nombre double à l'utilisateur sur le terminal. Cette fonction doit retourner un double mais elle n'a besoin d'aucun paramètre.

notes

résumé

3m 37s

Fonctions sans paramètre

EPFL

Il est aussi possible de définir des fonctions sans paramètre.
Il suffit, dans le prototype et la définition, d'utiliser une liste de paramètres vide : ()

Exemple :

```
double saisie()
{
  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));

  return nb_points;
}
```

On va alors déclarer ici une variable, par exemple, un nombre de points, à demander à l'utilisateur, qu'on va déclarer de type double. Ici le type qu'on va utiliser va être le même type de la valeur de retour puisque c'est cette variable qu'on va utiliser pour retourner la valeur au reste du programme. On déclare cette variable ici "nb_points", on l'initialise à zéro. Ensuite, on pose la question à l'utilisateur d'entrer un nombre de points. On va lire sur le terminal le nombre de points que l'utilisateur doit nous fournir et puis tant le nombre de points est négatif ou plus grand qu'un certain seuil, à ce moment-là, on va reboucler demander à l'utilisateur. Quand on aura enfin une valeur correcte,

notes

résumé

4m 37s

La fonction `main()`

`main` est aussi une fonction avec un nom et un prototype imposés.

Par convention, tout programme C++ doit avoir une fonction `main`, qui est appelée automatiquement quand on exécute le programme.

Cette fonction doit retourner une valeur de type `int`. La valeur 0 indique par convention que le programme s'est bien déroulé.

Les deux seuls prototypes autorisés pour `main` sont :

```
int main();
int main(int argc, char** argv);
```

int main();

Seul le premier sera utilisé dans ce cours.

on retourne au reste du programme le nombre de points que nous aura donné l'utilisateur. Vous voyez que le programme qui va utiliser, qui va faire appel à saisie n'a absolument besoin de ne rien passer comme arguments. La fonction saisie travaille tout à fait de façon autonome. Dernier cas un peu particulier de fonction. La fonction « main », la fonction principale. « Main » est une fonction comme pratiquement toutes les autres, simplement, c'est la fonction qui est appelée au tout début du programme. Cette fonction a un prototype particulier. Elle en a en fait deux, mais le prototype qu'on utilisera dans ce cours, c'est le protoype suivant : elle doit retourner un entier, int, et ne recevoir aucun argument. La façon standard de définir « main » est la suivante. Cet entier, int, ici qu'on retourne au reste du programme c'est ce qu'on retourne au programme qui a appelé notre programme c'est-à-dire l'environnement dans lequel vous allez lancer votre programme

notes

résumé

5m 25s

et ça va être typiquement un code d'erreur avec la convention que la valeur 0 indique par convention que le programme s'est bien déroulé ici. Cela veut dire qu'il n'y a pas d'erreur. C'est peut-être un peu surprenant mais c'est une convention habituelle, zéro veut dire qu'il n'y a pas d'erreur. En général, on utilise ici un code d'erreur plus grand que zéro, un, deux etc. pour indiquer qu'il y a une erreur. Mais ça, ça sort du cadre de ce cours. Juste pour signaler un autre prototype, le seul autre prototype autorisé ici pour « main », que l'on peut parfois rencontrer dans certains environnements de développement, c'est celui que vous aurez de façon standard. C'est un prototype qui prend deux arguments mais ça nous amènerait un peu trop loin d'expliquer ici ce que sont ces deux arguments. Ce sont tout simplement des paramètres que l'on pourrait recevoir de l'environnement dans lequel vous allez lancer votre programme. vous allez lancer votre programme.

notes

résumé

6m 25s