Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Boucles conditionnelles (partie 4)

Concepts (extraits des sous-titres générés automatiquement) :

Nombre de notes. Boucle do while. Nombre limité d'essais. Partie du corps de la boucle. Condition de continuation de la boucle. Condition de façon. Programme de sorte. Bonne pratique. Notes inférieur. Besoin d'un do while. Résultat de l'évaluation de cette condition. Corps de la boucle. Solution de cette nature. Compteur d'essais. Formulation c.

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

vers la vidéo

EPFL

Boucles conditionnelles
(Partie 4)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

```
do {
  cout << "Entrez le nombre de notes:" << endl;
  cin >> nombre_de_notes;
} while(nombre_de_notes <= 0);
```

```
Entrez le nombre de notes:
-2
il faut entrer un nombre supérieur à 0
Entrez le nombre de notes:
5
```

Alors étoffons un petit peu notre exemple de tout à l'heure.

notes

résumé

0m 1s

bool saisie_invalide (false);

```
do {
  cout << "Entrez le nombre de notes:" << endl;
  cin >> nombre_de_notes;
  if (nombre_de_notes <= 0) {
    cout << "il faut entrer un nombre supérieur a 0" << endl;
  }
} while(nombre_de_notes <= 0);
```

saisie_invalide = (nombre_de_not

```
Entrez le nombre de notes:
-2
il faut entrer un nombre superieur a 0
Entrez le nombre de notes:
5
```

Il s'agit d'un cas typique où nous avons besoin d'un do while parce que l'utilisateur a besoin de saisir au moins une fois le nombre de notes avant que l'on soit capable de tester si elle entre dans les bornes voulues, donc nous avons recours à une boucle do while et maintenant nous nous proposons d'améliorer un petit peu ce programme de sorte à ce que l'utilisateur soit avisé, au moment où il introduit un mombre par exemple, des attentes du programme par le biais d'un message. Donc on aimerait faire en sorte que le programme si dessus soit complété de sorte à ce que l'utilisateur ait un message pertinent au moment où il introduit une note qui est négative ou nulle. Comment procéder? Evidemment ce message devrait apparaitre pour toutes les situations où le nombre ne satisfait pas la condition souhaitée donc c'est quelque chose qui vraisemblablement va faire partie du corps de la boucle, on doit répéter cette instruction à chaque fois que l'utilisateur a introduit quelque chose qui n'est pas valable. Donc on peut imaginer une solution de cette nature, on enrichit le corps de la boucle avec un test supplémentaire : après avoir fait la lecture, on va tester si le nombre introduit est inférieur ou égal à zéro et à ce moment-à, afficher le message voulu pour aviser l'utilisateur. A noter que ceci fait effectivement partie du corps de la boucle puisqu'on veut répéter ce traitement à chaque fois qu'un nombre non souhaité est introduit. Vous noterez que dans cet exemple, la condition qui consiste à évaluer si le nombre de notes est inférieur ou égal à zéro, apparait désormais deux fois, une fois dans le if et une fois comme condition de continuation de la boucle do while. il aurait été judicieux ici de stocker le résultat de l'évaluation de cette condition dans une variable afin de ne pas répéter

notes

résumé

0m 5s

bool saisie_invalide (false);

```
do {
  cout << "Entrez le nombre de notes:" << endl;
  cin >> nombre_de_notes;
  if (nombre_de_notes <= 0) {
    cout << "il faut entrer un nombre supérieur a 0" << endl;
  }
} while(nombre_de_notes <= 0);
```

saisie_invalide = (nombre_de_not

```
Entrez le nombre de notes:
-2
il faut entrer un nombre superieur a 0
Entrez le nombre de notes:
5
```

son évaluation. Par exemple je peux introduire une variable boolénne saisie_invalide que j'initialise à false, que je mets à jour ici dans le corps de ma boucle, à chaque itération en y stockant le résultat de l'évaluation de la condition, à savoir nombre_de_notes

notes

résumé

EPFL

Comment trouver la condition ?

Supposons maintenant qu'on veuille limiter le nombre de notes à 10.

On veut toujours qu'il soit supérieur à 0.

Comment trouver la nouvelle condition ?

On veut répéter la boucle **tant que** le nombre de notes est incorrect,

inférieur ou égal à zéro, donc dans saisie_invalide j'aurais true ou false et à ce moment là je peux écrire la condition de façon unifiée dans les deux cas en remplaçant ceci simplement par "if (saisie invalide)", et je continue les traitements comme je l'ai fait tout à l'heure et pareil ici. Je répète les traitements while la valeur de la variable saisie_invalide est true. Il s'agit là d'une bonne pratique. Il est toujours souhaitable dans un programme d'éviter les situations où on exprime deux fois de façon indépendante la même chose à des endroits différents. C'est comme le copié-collé. Alors toujours avec notre petit exemple de la saisie d'un nombre de notes par l'utilisateur, examinons maintenant comment on peut formuler la condition de façon différente pour répondre à d'autres besoins. Jusqu'ici nous imposions à l'utilisateur d'introduire un nombre de notes qui soient supérieures à zéro. Imaginons maintenant que l'on souhaite en plus limiter le nombre de notes à dix. La question que l'on se pose est comment trouver, formuler la nouvelle condition pour répondre à ce nouveau besoin.

notes

résumé

2m 13s

Supposons qu'on veuille écrire un programme qui demande à l'utilisateur de deviner un nombre. Pour simplifier, nous supposerons que le nombre à deviner est toujours 5.

Le programme peut s'écrire ainsi:

```
int nombre_a_deviner(5);
int nombre_entre;

do {
  cout << "Entrez un nombre entre 1 et 10" << endl;
  cin >> nombre_entre;
} while( condition ? );

cout << "Trouve" << endl;
```

Pour formuler correctement la condition d'arrêt, commençons par l'exprimer tout simplement en langue naturelle. Désormais, le nombre de notes introduit par l'utilisateur va être incorrect dans deux situations : soit que l'utilisateur a introduit un nombre inférieur ou égal à zéro, ce qui contredit ce souhait, soit l'utilisateur a introduit un nombre supérieur à 10, ce qui contredit celui ci. Donc ici on voit que l'on doit boucler tant que l'on est dans l'une ou l'autre des situations. Dans l'une ou l'autre des situations on a un nombre incorrect. Ce qui nous permet de voir assez naturellement que l'on doit en fait avoir recours au connecteur OU donc en fait on continue les traitements tant que l'une ou l'autre des situations est rencontrée. Ce qui en C++ va se traduire par une ligne de codes qui a cette allure, on répète le traitement tant que le nombre de notes introduit est inférieur ou égal à zéro ou que le nombre de note est supérieur à 10. Toujours dans le même esprit, imaginons que l'on souhaite écrire un petit programme dont le but soit de faire deviner un nombre à un utilisateur. Pour simplifier ici, on imagine que le nombre à deviner est toujours 5. L'idée ici serait d'écrire une boucle do while qui demande un nombre à l'utilisateur et qui le

notes

résumé

3m 25s

la boucle doit être répétée
tant que l'utilisateur n'a pas trouvé le nombre à deviner, c'est-à-dire

```
    cin >> nombre_entre;
} while( condition ? );

cout << "Trouve" << endl;
```

compare au nombre souhaité et on ne sort de la boucle qu'une fois le bon nombre trouvé. Donc la question que nous nous posons, c'est comment formuler cette condition ici pour sortir proprement de la boucle une fois que l'utilisateur a trouvé le bon nombre.

notes

résumé

4m 37s

Supposons qu'on veuille en plus limiter le nombre d'essais à 3.
On peut ajouter une variable qui va compter le nombre d'essais utilisés:

```
int nombre_a_deviner(5);
int nombre_entre;
int nombre_essais(0);

do {
  cout << "Entrez un nombre entre 1 et 10" << endl;
  cin >> nombre_entre;
  ++nombre_essais;
} while( condition ? );
```

Comment modifier la condition pour que la boucle s'arrête quand le nombre d'essais dépasse 3 ?

Comme pour tout à l'heure, on va formuler la condition en langue naturelle. On se dit que la boucle doit être répétée tant que l'utilisateur n'a pas trouvé le nombre à deviner. Ça veut dire quoi concrètement? Il n'a pas trouvé le nombre à deviner tant que le nombre qu'il a entré est différent du nombre à deviner qui était 5. Comment formuler cette condition en C++? Tout simplement par la ligne suivante, donc on répète le traitement tant que le nombre entré est différent du nombre à deviner. Donc on voit que la formulation C++ est très voisine, très proche de la façon de le formuler en langue naturelle. Sophistiquons maintenant un petit peu cet exemple où l'utilisateur doit deviner un nombre, donc on imagine qu'on veut lui donner un nombre limité d'essais. Il ne va pas pouvoir indéfiniment proposer une valeur, donc ici on imagine qu'on va devoir avoir recours à un compteur d'essais donc une variable que l'on introduit ici et que ce compteur d'essais va être incrémenté à chaque itération de la boucle et en fait on veut que

notes

résumé

4m 52s

la boucle doit être répétée
tant que l'utilisateur n'a pas trouvé le nombre à deviner et qu'il reste des essais,

```
} while( condition ? );
```

au bout de trois essais l'utilisateur ait perdu s'il n'a pas trouvé le nombre à deviner.
Donc la question que l'on se pose maintenant est comment formuler cette condition.
Encore une fois, commençons par formuler la condition en langue naturelle.

notes

résumé

5m 49s

la boucle doit être répétée
tant que l'utilisateur n'a pas trouvé le nombre à deviner et qu'il reste des essais,
c'est-à-dire
tant que `nombre_entre` est différent de `nombre_a_deviner` et que
`nombre_essais` est inférieur à 3,
la condition est donc:

```
} while(nombre_entre != nombre_a_deviner and nombre_essais < 3);
```

Donc on se dit qu'on doit répéter la boucle tant que l'utilisateur n'a pas trouvé le bon nombre et qu'il lui reste des essais, donc ici c'est clair que les deux conditions doivent simultanément être vérifiées, on doit avoir les deux conditions pour pouvoir continuer à itérer dans la boucle. Donc ici on va essayer de le formuler d'une façon un petit peu plus proche du C++. Donc première condition, tant que le nombre entré est différent du nombre à deviner et deuxième condition, tant que le nombre d'essais est inférieur à trois, les deux conditions devant être simultanément vérifiées. En C++ cela va correspondre à la ligne de codes suivante. Donc première condition ici, tant que le nombre entré est différent du nombre à deviner, ce qui correspond exactement à ce qu'on a formulé ici et deuxième condition, tant que le nombre d'essais est inférieur à trois, ce qui correspond à cette seconde condition et nous avons ici notre opérateur logique ET, and, qui permet d'assurer que

notes

résumé

6m 3s

```
int nombre_a_deviner(5);
int nombre_entre;
int nombre_essais(0);

do {
  cout << "Entrez un nombre entre 1 et 10" << endl;
  cin >> nombre_entre;
  ++nombre_essais;
} while(nombre_entre != nombre_a_deviner and nombre_essais < 3);
```

Si on veut afficher un message pour indiquer à l'utilisateur s'il a trouvé le nombre ou si il a épuisé ses essais, on peut ajouter après la boucle:

```
if (nombre_entre == nombre_a_deviner) {
  cout << "Trouve" << endl;
} else {
  cout << "Perdu. Le nombre etait " << nombre_a_deviner << endl;
}
```

les deux conditions soient simultanément vérifiées. Donc ici encore une fois on voit que le formuler en C++ n'est pas très loin de le formuler en langue naturelle. L'utilisateur a désormais deux façons possibles de sortir de la boucle. Il peut en sortir soit parce qu'il a trouvé le nombre, ce qui veut dire que cette partie de la condition devient fausse, ou alors parce qu'il a dépassé le nombre d'essais. Et maintenant supposons que nous souhaitions indiquer à l'utilisateur en clair, pourquoi il est sorti concrètement de la boucle. Alors si l'on veut informer l'utilisateur de pour quelle raison on est sorti exactement de la boucle, et bien il suffit de complémenter le programme tel qu'il était à l'origine en faisant en sorte qu'une fois qu'on est sorti de la boucle, on teste pour quelle condition on est sorti. Est-ce que l'on est sorti de la boucle en raison de l'échec de cette condition, c'est-à-dire que l'utilisateur a deviné le nombre ou est ce qu'on est sorti de la boucle en raison de l'échec de cette seconde condition, est-ce qu'on est sorti de la boucle parce qu'on a dépassé le nombre d'essais. Donc ici on va le tester tout simplement. Donc si le nombre entré est égal au nombre à deviner ce qui veut dire qu'on serait sorti en raison de l'échec de cette condition et bien on

notes

résumé

7m 1s

```
int nombre_a_deviner(5);
int nombre_entre;
int nombre_essais(0);

do {
  cout << "Entrez un nombre entre 1 et 10" << endl;
  cin >> nombre_entre;
  ++nombre_essais;
} while(nombre_entre != nombre_a_deviner and nombre_essais < 3);
```

Attention, si on avait utilisé `nombre_essais < 3` comme condition:

```
if (nombre_essais < 3) {
  cout << "Trouve" << endl;
} else {
  cout << "Perdu. Le nombre etait " << nombre_a_deviner << endl;
}
```

le programme afficherait `"Perdu. ..."` quand l'utilisateur trouve au troisième essai.

peut signifier à l'utilisateur qu'il a trouvé le bon nombre. Sinon et bien cela signifie qu'il a dépassé le nombre d'essais qui lui était alloué, on peut donc l'informer qu'il a perdu et on peut aussi faire afficher de surcroit le nombre à deviner de sorte à l'informer de ce que l'on s'attendait à ce qu'il trouve.

notes

résumé

8m 13s

Alors attention ici aux erreurs de logique dans la formulation des tests. Imaginons que pour tester cette condition que j'ai effectivement trouvé le bon nombre, je teste que je n'ai pas dépassé le nombre d'essais et bien je vais passer à coté de la situation où l'utilisateur aurait deviné au troisième essai exactement, le bon nombre et à ce moment là on va comptabiliser ce cas comme un cas d'échec au lieu de le comptabiliser proprement comme un cas de succès. Donc il faut être bien vigilant à la logique de la formulation des tests si l'on veut réaliser ce genre de traitement. La bonne solution ici était celle qui était montrée dans la première solution à savoir formuler la condition non pas par rapport au nombre d'essais mais par rapport au fait d'avoir trouvé ou non le nombre et cette façon de formuler la condition va nous permettre d'éviter l'écueil que nous venons juste d'écrire nous venons juste d'écrire

notes

résumé

8m 32s