GDB

Introduction au débogage

Qu'est-ce que le débogage ?

Définition

Le **débogage** est le processus d'identification et de correction d'erreurs présentes dans un **programme informatique**.

Importance dans le développement logiciel

Le **débogage** est essentiel pour produire un logiciel **efficace**, **fiable** et de **haute qualité**.

Erreurs fréquentes en C++

Erreurs de syntaxe

Les **erreurs de syntaxe** sont des erreurs liées à l'utilisation incorrecte des éléments du **langage C++**, comme des parenthèses mal fermées ou des points-virgules manquants.

Exemple d'erreur	Correction
`if (x > 0`	`if (x > 0)`
`cout << "Hello`	`cout << "Hello";`

Erreurs logiques

Les erreurs logiques sont des erreurs dans la **logique** du programme qui conduisent à des comportements **inattendus** ou **incorrects**.

Exemples d'erreurs logiques	Solutions possibles
Boucle infinie due à une mauvaise condition	Vérifier la condition de la boucle et la modifier si nécessaire
Utilisation de la mauvaise variable	Vérifier les noms des variables et leur initialisation

Calcul mathématique incorrect	Vérifier les opérations et les priorités

Erreurs de mémoire

Les **erreurs de mémoire** sont des erreurs liées à la gestion de la mémoire, comme les **fuites de mémoire** ou les **corruptions de données**.

Installation et lancement de GDB

Installation de GDB

Installation pour Linux

Installez **GDB** sur les distributions Linux basées sur **Debian** à l'aide de la commande suivante :

```
sudo apt update
sudo apt install gdb
```

Installation pour Windows

Pour installer **GDB** sous Windows, téléchargez et installez "**MinGW**" à partir du site officiel :
MinGW

Après l'installation, assurez-vous d'ajouter le chemin d'accès de MinGW à la variable d'environnement PATH de Windows.

Installation pour MacOS

Pour installer **GDB** sous **MacOS**, installez **Homebrew** si vous ne l'avez pas déjà fait, puis exécutez la commande suivante :

```
brew install gdb
```

Lancement de GDB

Ouverture de GDB

Pour ouvrir **GDB**, ouvrez un terminal et entrez simplement "gdb" suivi du nom de votre programme compilé en ajoutant l'argument `-g` pendant la compilation :

```
g++ -g my_program.cpp -o my_program
gdb my_program
```

L'argument `-g` permet d'inclure les informations de débogage dans le fichier binaire. Cela facilite l'utilisation de GDB pour analyser et résoudre les problèmes.

Chargement du programme

Une fois **GDB** ouvert, chargez votre programme en entrant :

```
file my_program
```

Debug avec Gdb en C++

Commandes de base de GDB

`run`

La commande `run` permet de lancer l'exécution du **programme** chargé dans **GDB**.

Avant d'utiliser la commande `run`, assurez-vous que le programme a bien été compilé avec les options de débogage appropriées (par exemple, `-g` avec `g++`).

Utilisation

```
(gdb) run [args]
```

Cette commande permet de démarrer le programme en cours de débogage avec les arguments spécifiés.

Exemple

```
(gdb) run arg1 arg2
```

Cet exemple montre comment exécuter un programme avec des arguments (arg1 et arg2) dans GDB, le débogueur GNU pour C++.

`list`

La commande `list` affiche le **code source** du programme autour de la **position actuelle** ou d'une **position spécifiée**.

Cette commande est utile pour situer rapidement le programme sans devoir ouvrir le fichier source dans un éditeur de code.

Utilisation

```
(gdb) list [location]
```

La commande `list [location]` permet d'afficher le code source autour de l'emplacement spécifié. Si aucune location n'est fournie, elle affiche la suite du code précédemment affiché.

Exemple

```
(gdb) list 42
```

Cette commande permet de visualiser le code source à partir de la ligne 42, en utilisant l'outil de débogage GDB.

Utilisation

```
(gdb) print [expression]
```

Cette commande permet d'afficher la valeur d'une **expression** à l'écran pendant le débogage en C++.

Exemple

```
(gdb) print a
```

La commande "print" permet d'afficher la valeur d'une variable dans le débogueur GDB.

`break`

La commande `break` permet de définir un **point d'arrêt** (breakpoint) à une position donnée.

Commande	Explications
`break`	Stoppe l'exécution du programme à un point spécifique dans le code.
`break line-number`	Stoppe l'exécution du programme à la ligne spécifiée.

Utilisation

```
(gdb) break [location]
```

Cette commande permet de poser un point d'arrêt à une position spécifiée du programme en débogage.

Exemple

```
(gdb) break main
```

La commande "break main" permet de mettre un point d'arrêt au début de la fonction main dans le programme à exécuter.

Break

`info breakpoints`

La commande `info breakpoints` affiche la liste de tous les **points d'arrêt** définis.

Les points d'arrêt permettent de stopper l'exécution du programme à un certain point pour pouvoir examiner son état.

Utilisation

```
(gdb) info breakpoints
```

Points d'arrêt et étapes

Points d'arrêt (breakpoints)

Les **points d'arrêt** sont des **marqueurs** placés dans le code pour indiquer à **GDB** où interrompre l'exécution du programme.

Ajout et suppression de points d'arrêt

Pour ajouter un point d'arrêt :

```
break ligne
break fonction
```

Pour supprimer un point d'arrêt :

```
delete breakpoint-numéro
```

Les points d'arrêt sont utilisés pour définir des points spécifiques où le programme doit s'arrêter lors du débogage. Ceci est particulièrement utile pour comprendre et corriger des erreurs dans le code.

Étape simple (step)

L'étape simple exécute une ligne de code et s'arrête. Si la ligne de code est un **appel de fonction**, GDB exécutera la fonction et s'arrêtera à la première ligne de celle-ci.

N'oubliez pas de mentionner que GDB est un débogueur pour les langages de programmation comme le C++.

Utilisation

```
step
```

Commande	Description
`step`	Exécute une ligne de code C++

Exemples

```
(gdb) step
```

La commande 'step' permet de déboguer pas à pas, en entrant dans les fonctions/les appels. Cela permet de trouver et corriger les erreurs de logique ou de syntaxe dans le code.

Étape sur (next)

L'étape sur exécute une ligne de code et s'arrête. Si la ligne de code est un appel de fonction, **GDB** exécutera la fonction entièrement et s'arrêtera à la ligne suivante.

Exemples

```
(gdb) next
```

La commande "next" est utilisée pour exécuter la ligne suivante du code en cours d'exécution sans rentrer dans les appels de fonction.

Commande `ptype`

La commande `ptype` **affiche** le **type** d'une **variable** en C++.

Affichage du type d'une variable

```
ptype nom_variable
```

Cette commande est utile pour comprendre le type de données d'une variable, surtout dans les situations où le code source n'est pas disponible ou difficile à suivre.

Exemples

```
ptype a
ptype *ptr
```

Gérer la fin de l'exécution

Commande `quit`

La commande `quit` permet de **quitter gdb**. Elle met fin à la session de **débogage**, mais n'affecte pas l'exécution du programme en cours de débogage.

Utilisation

```
(gdb) quit
```

Note : La commande `(gdb) quit` permet de quitter l'environnement de débogage GDB en C++.

Impact sur le programme

L'exécution du programme en cours de **débogage** n'est pas affectée par la commande `quit`. Le programme continue de s'exécuter normalement après la fermeture de **gdb**.

Commande `continue`

La commande `continue` permet de reprendre l'exécution du programme depuis le **point d'arrêt actuel** jusqu'au **prochain point d'arrêt** ou jusqu'à la fin du programme.

Utilisation

```
(gdb) continue
```

Cette commande permet de poursuivre l'exécution d'un programme en cours de débogage avec GDB jusqu'à la prochaine interruption (comme un point d'arrêt).
