

README

L'objectif :

L'Objectif est de determiner de manière pratique le nombre maximale des paramètres q'on peut l'utiliser dans une fonction avant provoquer une erreur .

Comment on peut créer une fonction qui avoir un grande nombre des argument ?

On peut fair ça en utilisant 2 methode :

Méthode Classique :

on ajoute les argument d'une manière manuelle

```
int function(int a1, int a2, int a3, int a4, int a5, int a6, int a7, int
a8, int a9, int a10, int a11, int a12, int a13, int a14, int a15, int a16,
int a17, int a18, int a19, int a20, int a21, int a22, int a23, int a24,
int a25, int a26, int a27, int a28, int a29, int a30, int a31, int a32,
int a33, int a34, int a35, int a36, int a37, int a38, int a39, int a40,
int a41, int a42, int a43, int a44
                // ETC ...){

return 0;
}
```

mais cette methode n'est pas pratique et aussi impossible d'ecrire un grand nombre des argument meme s'il est possible .

a cause de cette inconvinient on utilise les `fonction variadiques` ;

2ème Methode :

Définition :

Pour déclarer une fonction variadique, des points de suspension apparaissent après la liste des paramètres, par exemple `int printf(const char* format...);`, qui peuvent être précédés d'une virgule facultative. Voir Arguments variadiques pour plus de détails sur la syntaxe, les conversions automatiques d'arguments et les alternatives.

[réfrence](#)

Exemple :

```
#include <cstdarg>
#include <iostream>

void afficherArguments(int count, ...)

{
    va_list args;
    va_start(args, count);
    for (int i = 0; i < count; ++i){
        int arg = va_arg(args, int);
        std::cout << "Argument " << i + 1 << ": " << arg <<
std::endl;
    }
    va_end(args);
}

int main() {
    afficherArguments(3);
    return 0; }
```

Explication :

- **va_list args** : Déclare une liste pour stocker les arguments variadiques.
- **va_start(args, count)** : Initialise la liste des arguments en prenant en compte le nombre d'arguments spécifiés.
- **va_arg(args, int)** : Récupère chaque argument de type `int` de la liste d'arguments.
- **va_end(args)** : Termine l'utilisation de la liste des arguments variadiques pour libérer la mémoire allouée.

avec cette méthode on peut declarer un seule fonction and manipuler le nombre des argument qu'on peut passer .

Passons aux tests pour voir les resultat pratiquement :

NB:

les tests sont passe sur une machine linux avec les configuration suivant :

```


          ////////////////
        ////////////////
      ////////////*767//////////
    ////////////76767676*//////////
  ////////////76767//7676767//////////
//////////////767676//*76767//////////
//////////////767676//76767.//7676*//////////
//////////////767676//76767//767676//////////
//////////////76767676767//76767//////////
//////////////76767676//7676//////////
//////////////,7676,,//767//////////
//////////////*7676//////////76//////////
//////////////7676//////////767//////////
//////////////7676//767//////////
//////////////'//////////
//////////////.7676767676767676767,//////////
//////////////7676767676767676767//////////
////////////////////////
////////////////////////
////////////////////////

```

```

ezzoubair@pop-os
-----
OS: Pop!_OS 22.04 LTS x86_64
Host: 82C6 Lenovo V14-ADA
Kernel: 6.9.3-76060903-generic
Uptime: 7 days, 17 hours, 52 mins
Packages: 2749 (dpkg), 75 (flatpak)
Shell: bash 5.1.16
Resolution: 1920x1080
DE: GNOME 42.9
WM: Mutter
WM Theme: Pop
Theme: Pop-dark [GTK2/3]
Icons: Pop [GTK2/3]
Terminal: gnome-terminal
CPU: AMD 3020e with Radeon Graphics (
GPU: AMD ATI Radeon Vega Series / Rad
Memory: 3588MiB / 5802MiB

```



Test 01 : 100 argument

on règle le param count=100, et executer le programme :

-> Resultat : Programme compile correctement et executer d'une manière normal.

Test 02 : 1 000 argument

on règle le param count=1 000, et executer le programme :

-> Resultat : Programme compile correctement et executer d'une manière normal.

Test 03 : 1 100 argument

on règle le param count=1 100, et executer le programme :

-> Resultat : Programme compile correctement , mais en execution , parfois il donne un erreur de memoire :

```
Segmentation fault (core dumped)
```

cet erreur signifie que le programme tente d'accéder à un emplacement de mémoire auquel il n'a pas l'autorisation d'accéder. En général, cette erreur se produit lorsque l'accès à la mémoire est violé et constitue un type d'erreur de protection générale .

Test 03 : 1 300 argument

on regle le param count=1 300, et executer le programme :

-> Resultat : Programme compile correctement , mais en execution , le programme donne un erreur de memoire :

```
Segmentation fault (core dumped)
```

Remarque :

on peut deduire que le nombre maximale des parametre dans une fonction est entre 1100 et 1300 .

Conclusion :

En conclusion, cette étude pratique a permis de déterminer le nombre maximal d'arguments qu'une fonction peut prendre avant de provoquer une erreur, notamment dans le cadre des fonctions variadiques en C++. Les tests ont montré qu'une fonction pouvait accepter jusqu'à environ 1 100 arguments sans problèmes majeurs, tandis qu'à partir de 1 300 arguments, des erreurs de mémoire apparaissent lors de l'exécution, probablement liées à des violations d'accès mémoire. Ces résultats illustrent les limites des fonctions variadiques et l'importance de gérer efficacement le nombre d'arguments pour éviter des problèmes de mémoire et des comportements imprévus du programme.