École de technologie supérieure

Document spécial: les arguments de la fonction "main"

INF125 Introduction à la programmation
Sylvie Ratté et Hugues Saulnier

Comme vous le savez sans doute, le "main" est une fonction qui peut accepter des arguments. Grâce à ceux-ci, le programme peut recevoir des informations provenant de l'extérieur sans pour autant utiliser les librairies d'entrée/sortie.

1. Syntaxe et récupération

Puisqu'un programme peut être appelé grâce à une ligne de commande de longueur variable, on a prévu un mécanisme très simple pour récupérer tous les arguments utilisés. Deux paramètres doivent être définis si l'on désire fournir des informations à un programme: le premier contiendra le nombre d'arguments (ce paramètre est donc entier) et le second contiendra (dans un tableau de chaînes de caractères) la valeur de chaque argument. Traditionnellement, les programmeurs utilisent les noms ARGC et ARGV (argument count et argument values) pour identifier ces deux variables. On obtient ainsi:

```
int main(int argc, char * argv[])
```

Il est important de noter qu'il y a toujours au moins 1 argument (donc "argc" vaut minimalement 1). Cet argument est en fait "le nom du programme lui-même. Ainsi, si votre programme compilé se trouve dans le fichier "toto.exe" dans le répertoire "C:\TC\BIN" et que votre "main" ressemble à la ligne précédente, alors lors de l'exécution de votre programme, "argc" vaut 1 et de ce fait, argv[0] vaut "C:\TC\BIN\toto.exe".

On comprendra dès lors une chose. Pour récupérer la valeur de chaque argument à partir du tableau "argv", il faudra procéder à une conversion de la chaîne de caractères vers le type approprié.

2. Communication

Voici un petit programme simple qui vous permettra de comprendre le mécanisme utilisé. Le fichier se nomme "ARGU.C" et le code compilé se trouve à la racine du disque C.

`int main(int argc, char * argv[]) {`	Le programme admet des arguments...
`    printf("\n\n\n%d", argc);`	Affichage de la valeur de "argc".

`    for (int k = 0; k < argc; k++)`
`       printf("\n%s", argv[k]);`
`}`
	Affichage de la valeur de chacun des arguments.

Si le programme est appelé avec la ligne de commande suivante:

```
ARGU 567 903 Sylvie
```

Le programme affichera:

```
4
C:\ARGU.EXE
567
903
Sylvie
```

Nous pouvons maintenant créer de véritables commandes pour le système d'exploitation!

3. Imitations véritables (!) de TYPE et COPY

A. Imitation de la commande TYPE

`#define NB 1024`	la taille du tampon de lecture
`int main(int argc, char * argv[]){`	les paramètres
`  FILE * source;`	
`  char tampon[NB];`	notre tampon de lecture
`  int nb_lu;`	contiendra le nombre d'ÉLÉMENTS lus
TRAITEMENT DU CAS D'ERREUR SUR LE NOMBRE D'ARGUMENTS	

`  if (argc < 2){`
`   printf("\nje n'ai pas de fichiers...?");`
`   return 0;`
`  }`
	Lorsque la ligne de commande ne comporte pas de paramètres, comme dans:
	`C:\>MTYPE`
	le programme affiche un petit message. RAPPEL: "argc" vaut alors 1. argv[0] vaut "C:\MTYPE"

OUVERTURE DU FICHIER EN LECTURE ET TRAITEMENT D'ERREUR SUR CELLE-CI	
`  source = fopen(argv[1], "rb");`	La commande effectuée comporte un nombre d'arguments adéquats: `C:\>MTYPE toto.txt` Dans ce cas, argv[0] vaut "C\MTYPE", argv[1] vaut "toto.txt".

`  if (!source) {`
`    printf("\nLe fichier source est introuvable");`
`    return 0;`
`  }`
	Petite validation.

FONCTIONNEMENT DE STYLE "commande TYPE"	

`    while(1) {`
`       nb_lu = fread(tampon, 1, NB, source);`
`       if (!nb_lu)   break;`
`       fwrite(tampon, 1, nb_lu, stdout);`
`       if (nb_lu < NB)   break;`
`    }`
`    fclose(source);`
`    return 0;`
`}`
	le "type" comme tel...

B. Imitation de la commande COPY

`#define NB 1024`	la taille du tampon de lecture
`int main(int argc, char * argv[]){`	les paramètres habituels
`  FILE * source;`	
`  FILE * cible;`	
`  char tampon[NB];`	notre tampon de lecture
`  int nb_lu;`	contiendra le nombre d'ÉLÉMENTS lus
TRAITEMENT DU CAS D'ERREUR SUR LE NOMBRE D'ARGUMENTS	

`  if (argc < 3){`
`   printf("\nje n'ai pas assez de fichiers...?");`
`   return 0;`
`  }`
	Lorsque la ligne de commande ne comporte pas de paramètres, comme dans:
	`C:\>MCOPY`
	ou
	`C:\>MCOPY toto.txt`
	le programme affiche un petit message. RAPPEL: "argc" vaut alors 1. argv[0] vaut "C:\MCOPY"

OUVERTURE DU PREMIER FICHIER EN LECTURE ET TRAITEMENT D'ERREUR SUR CELLE-CI	
`  source = fopen(argv[1], "rb");`	Le programme ouvre le fichier spécifié en lecture (r) et en binaire (b).

`  if (!source) {`
`    printf("\nLe fichier source est introuvable");`
`    return 0;`
`  }`
	Petite validation. Si tout est ok, la commande effectuée comporte un nombre d'arguments adéquats:
	`C:\>MCOPY toto.txt lulu.txt`
	Dans ce cas, argv[0] vaut "C\MCOPY", argv[1] vaut "toto.txt" et argv[2] vaut "lulu.txt".

OUVERTURE DU SECOND FICHIER EN ÉCRITURE ET TRAITEMENT D'ERREUR SUR CELLE-CI	
`    cible = fopen(argv[2], "wb");`	Le programme ouvre le fichier spécifié en écriture (w) et en binaire (b).

`  if (!cible){`
`    printf("\nLe fichier cible est inutilisable");`
`    fclose(source);`
`    return 0;`
`  }`
	Petite validation sur le résultat. N'oublions surtout pas de fermer la source si la cible n'est pas disponible.

FONCTIONNEMENT DE STYLE "commande COPY "	

`  while(1){`
`    nb_lu = fread(tampon, 1, NB, source);`
`    if (!nb_lu)  break;`
`    fwrite(tampon, 1, nb_lu, cible);`
`    if (nb_lu < NB)  break;`
`  }`
`  fclose(source);`
`  fclose(cible);`
`  return 0;`
`}`
	la copie comme telle.