CM 17 - PROGRAMMATION AVANCÉE

G. Bianchi, G. Blin, A. Bugeau, S. Gueorguieva, R. Uricaru

2015-2016

Programmation 1 - uf-info.ue.prog1@diff.u-bordeaux.fr

université de BORDEAUX

FONCTION VARIADIQUE

DÉFINITION

▷ Une fonction est variadique si elle accepte un nombre d'arguments variable (indiqué par '...')

▷ Exemple: `int printf(const char*, ...);`

```c
#include <stdio.h>
#include <stdlib.h>
int main(void){
  printf("%d\n",1);
  printf("%d %c %f\n",1,'a',1.0);
  return EXIT_SUCCESS;
}
```

▷ Ces fonctions sont gérées par la librairie **`stdarg.h`** qui définit des types et des macros

▷ ⚠Il faut au moins un paramètre défini dans une fonction variadique

UTILISATION

▷ Dans une fonction variadique

▷ On crée une variable de type `va_list` qui contiendra successivement les différents paramètres

▷ On l'initialise à partir du dernier paramètre défini et connu (disons `last`) en utilisant la macro

```
void va_start(va_list ap, last);
```

▷ On accède au prochain paramètre avec la macro

```
type va_arg(va_list ap, type);
```

⚠ C'est au programmeur de savoir quel est le type du prochain paramètre

▷ Quand on a fini, on doit appeler une et une seule fois la macro `void va_end(va_list ap);`

EXEMPLE - FORMATAGE D'UNE DATE

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
void printD(const char* fmt, ...){
  if(fmt==NULL){return;}
  va_list args;
  va_start(args, fmt);
  int i=0, d;
  while(fmt[i]!='\0'){
    switch(fmt[i]){
      case 'D': case 'M': case 'Y':
          d=va_arg(args,int);
          if(fmt[i]=='Y'&&fmt[i+1]=='Y'){
            printf("%04d",d); i++;
          }else{printf("%02d",d%100);}
          break;
      default: printf("%c",fmt[i]);
    }
    i++;
  }
  va_end(args);
}
```

```c
int main(void){
  printD("D/M/Y\n",30,11,2015);
  printD("date:D-M-YY\n",30,11,2015);
  return EXIT_SUCCESS;
}
```

```
$>./a.out
30/11/15
date:30-11-2015
```

UTILISATION DES ARGUMENTS VARIABLES

- Les '...' n'ont pas d'identité (i.e., de symbole) et ne peuvent donc pas être transmis à une autre fonction
- En revanche, la `va_list` peut l'être
- Il existe des fonctions dans `stdio.h` l'acceptant
  - `vprintf`
  - `vfprintf`
  - `vsprintf`

FONCTION VARIADIQUE SANS CHAÎNE DE FORMATAGE

▷ On peut se passer de la chaîne de formatage si on suppose le type commun et connu en utilisant une valeur spéciale pour tester la fin des arguments

```
/**
* Like strcat, but optimized for multiple strings. The last
* parameter must be NULL. 'dest' is supposed to be large enough.
*/
void myStrcat(char* dest, ...){
  va_list args;
  va_start(args, dest);
  dest=dest+strlen(dest);
  char* s;
  while((s=va_arg(args,char*))!=NULL){
    *dest=*s;
    while(*dest){dest++; s++;*dest=*s; }
    dest--; //to posibly erase '\0' if another string is coming
  }
  va_end(args);
}
```

LE VRAI VISAGE DE MAIN

INT MAIN(???)

▷ Le prototype de la fonction `main` est

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

où `env` est un tableau contenant toutes les variables d'environnement dont le dernier élément vaut `NULL`

```
int main(int argc, char* argv[], char* env[]){
  int i;
  for(i=0; env[i]!=NULL; i++){
    printf("%s\n",env[i]);
  }
  return EXIT_SUCCESS;
}
```

```
$>./a.out
SHELL=/bin/bash
TERM=xterm-256color
USER=gblin
PWD=/Users/gblin
LANG=fr_FR.UTF-8
HOME=/Users/gblin
...
```

Accès aux variables d'environnement

▷ La librairie `stdlib.h` fournit la fonction

```
char* getenv(const char* name);
```

qui renvoit une chaîne correspondant à la variable d'environnement de nom **name** ou NULL si cette dernière n'est pas trouvée

```
int main(void){
  char* login=getenv("USER");
  if(login!=NULL){
    printf("Hello %s !\n",login);
  }
  return EXIT_SUCCESS;
}
```

```
$>./a.out
Hello gblin !
```

QUELQUES FONCTIONS UTILES

INVOQUER UNE COMMANDE DU SYSTÈME

▷ La librairie **stdlib.h** fournit la fonction

```
int system(const char* cmd);
```

qui démarre un shell, lance la commande **cmd** et attend la fin de son exécution

```
int main(void){
  printf("exit status:%d\n",system("date"));
  printf("exit status:%d\n",system("sleep 10"));
  printf("exit status:%d\n",system("date"));
  return EXIT_SUCCESS;
}
```

```
$>./a.out
Dim 29 nov 2015 14:25:16 CET
exit status:0
exit status:0
Dim 29 nov 2015 14:25:26 CET
exit status:0
```

Calculer un temps d'exécution

▷ La librairie `time.h` fournit la fonction

```
clock_t clock(void);
```

qui retourne le nombre de cycles d'horloge passés depuis le démarrage du programme

```
int main(void){
  clock_t start, end;
  start = clock ();
  /* code to time */
  end = clock ();
  printf ("Temps en secondes : %f\n", (end - start) /
                                      (double)CLOCKS_PER_SEC);
  return EXIT_SUCCESS;
}
```

Les crochets d'arrêt

▷ La librairie `stdlib.h` fournit la fonction

```
int atexit(void (*f)(void));
```

qui permet d'automatiser l'exécution de la fonction pointée par `f` à la fin du `main` ou quand `exit` est invoquée

▷ Les crochets peuvent être empilés - ce qui implique une exécution en ordre inversé d'exécution

LES CROCHETS D'ARRÊT - EXEMPLE 1

```
static int cpt=0;
void yousay(){
  printf((cpt==0?"You say yes, ","You say stop "));
}

void isay(){
  printf((cpt==0?"I say no\n","and I say go, go, go\n"));
  cpt++;
}

int main(void){
  atexit(isay);
  atexit(yousay);
  atexit(isay);
  atexit(yousay);
  return EXIT_SUCCESS;
}
```

```
$>./a.out
You say yes, I say no
You say stop and I say go, go, go
```

LES CROCHETS D'ARRÊT - EXEMPLE INTÉRESSANT

```c
static FILE *f;

static void clean(void){
    remove("tmpfile");
    fclose(f);
}

static void randexecute(void){
    if(rand()%2==0){
      exit(EXIT_FAILURE);
    }
}
```

```c
int main(void){
    srand(time(NULL));
    f = fopen("tmpfile", "w");
    if(f==NULL){
        exit(EXIT_FAILURE);
    }
    atexit(clean);
    randexecute();
    randexecute();
    return EXIT_SUCCESS;
}
```

```
$>./a.out
$>ls tmp*
ls: tmp*: No such file or directory
```

▷ Dans tous les cas d'arrêt du programme le fichier temporaire est supprimé (sauf en cas d'interruption)

LES CROCHETS D'ARRÊT - EXEMPLE INTÉRESSANT

```c
static FILE *f;

static void clean(void){
    remove("tmpfile");
    fclose(f);
}

static void randexecute(void){
    if(rand()%2==0){
      exit(EXIT_FAILURE);
    }
}
```

```c
int main(void){
    srand(time(NULL));
    f = fopen("tmpfile", "w");
    if(f==NULL){
        exit(EXIT_FAILURE);
    }
    atexit(clean);
    randexecute();
    randexecute();
    return EXIT_SUCCESS;
}
```

```
$>./a.out
^Z
[1]+  Stopped  ./a.out
$>ls tmp*
tmpfile
```

▷ Dans tous les cas d'arrêt du programme le fichier temporaire est supprimé (sauf en cas d'interruption)

TRAITEMENT DES OPTIONS DU PROGRAMME

▷ La librairie `unistd.h` fournit la fonction

```
int getopt(int argc, char* const argv[],
const char* optstring);
```

qui permet d'analyser les arguments d'un programme en ignorant leur ordre

▷ La fonction renvoie le caractère de l'option courante ou **EOF** si il n'y en a plus

▷ Fonctionnalité standard sous un système UNIX mais mal portée sous Windows

Traitement des options du programme - règles

▷ Si un argument commence par **-**, c'est une option

▷ Une option courte est symbolisée par un unique caractère précédé de **-**

▷ Une option longue est symbolisée par une chaîne de caractères précédée de **--**

▷ L'argument **-xyz** correspond donc à trois options courtes (i.e., **x**, **y** et **z**)

TRAITEMENT DES OPTIONS DU PROGRAMME - RÈGLES

▷ L'argument **optstring** de la fonction **getopt** permet de définir les options courtes autorisées

▷ Un caractère alphanumérique dans **optstring** correspond à une option

▷ Si un caractère est suivi de **':'** (resp. **'::'**), cela indique que l'option correspondante a un paramètre (resp. optionnel)

```
int main(int argc, char* argv[]){
  /* -h -> help
   * -i input
   * -o [output]
   */
   char* optstring="hi:o::";
   ...
   return EXIT_SUCCESS;
}
```

TRAITEMENT DES OPTIONS DU PROGRAMME - RÈGLES

▷ La paramètre d'une option doit être introduit soit par un espace, soit par rien

  ▷ **-i xxx** ou **-ixxx** correspond au paramètre **xxx**

  ▷ **-i=xxx** correspond au paramètre **=xxx**

▷ Dans le cas d'un paramètre optionnel, seule la forme **-ixxx** fonctionne

▷ Si **optstring** commence par un **':'**, la fonction **getopt** renverra **'?'** si une option est inconnue et **':'** si un argument obligatoire est manquant

▷ Le caractère de l'option courante est disponible dans la variable globale externe **int optopt**

UTILISATION DE GETOPT

```
int main(int argc, char* argv[]){
  /* -h -> help
   * -i input
   * -o [output]
   */
   char* optstring=":hi:o::";
   int val=getopt(argc,argv,optstring);
   while(val!=EOF){
     switch(val){
       case 'h': printf("help\n"); break;
       case 'o': printf("output %s\n",optarg); break;
       case 'i': printf("input %s\n",optarg); break;
       case ':': printf("arg missing for option %c\n",optopt); break;
       case '?': printf("unknown option %c\n",optopt); break;
     }
     val=getopt(argc,argv,optstring);
   }
   return EXIT_SUCCESS;
}
```

```
$>./a.out -hi
help
arg missing for option i
```

UTILISATION DE GETOPT

```
int main(int argc, char* argv[]){
  /* -h -> help
   * -i input
   * -o [output]
   */
   char* optstring=":hi:o::";
   int val=getopt(argc,argv,optstring);
   while(val!=EOF){
     switch(val){
       case 'h': printf("help\n"); break;
       case 'o': printf("output %s\n",optarg); break;
       case 'i': printf("input %s\n",optarg); break;
       case ':': printf("arg missing for option %c\n",optopt); break;
       case '?': printf("unknown option %c\n",optopt); break;
     }
     val=getopt(argc,argv,optstring);
   }
   return EXIT_SUCCESS;
}
```

```
$>./a.out -o -h -i tutu
output (null)
help
input tutu
```

UTILISATION DE GETOPT

```
int main(int argc, char* argv[]){
  /* -h -> help
   * -i input
   * -o [output]
   */
   char* optstring=":hi:o::";
   int val=getopt(argc,argv,optstring);
   while(val!=EOF){
     switch(val){
       case 'h': printf("help\n"); break;
       case 'o': printf("output %s\n",optarg); break;
       case 'i': printf("input %s\n",optarg); break;
       case ':': printf("arg missing for option %c\n",optopt); break;
       case '?': printf("unknown option %c\n",optopt); break;
     }
     val=getopt(argc,argv,optstring);
   }
   return EXIT_SUCCESS;
}
```

```
$>./a.out -i -y -z -oh
input -y
unknown option z
ouput h
```

TRAITEMENT DES OPTIONS DU PROGRAMME - ATTENTION

▷ ⚠La fonction **getopt** réorganise les paramètres du programme

▷ Quand la fonction renvoie **EOF**, les paramètres autres que les options et leur argument sont rangés dans **argv** entre les positions **optind** et **argc** (l'ordre est conservé)

```
int main(int argc, char* argv[]){
  char* optstring=":hi:o::";
  int val=getopt(argc,argv,optstring);
  while(val!=EOF){
    val=getopt(argc,argv,optstring);
  }
  while(optind!=argc){
    printf("%s\n",argv[optind]);
    optind++;
  }
  return EXIT_SUCCESS;
}
```

```
$>./a.out aa -h bb -i cc dd
aa
bb
cc
```

TRAITEMENT DES OPTIONS LONGUES DU PROGRAMME

▷ La librairie `unistd.h` fournit également la fonction

```
int getopt_long(int argc, char* const argv[],
const char* optstring, const struct option*
longopts, int* longindex);
```

qui permet d'analyser également les options longues

▷ Pour ces dernières, les paramètres peuvent être introduit soit avec un '=', soit avec un espace

▷ **--input=xxx** ou **--input xxx**

TRAITEMENT DES OPTIONS LONGUES DU PROGRAMME

▷ Le type **struct option** est défini comme suit

```
struct option{
  const char* name;
  int has_arg;
  int* flag;
  int val;
};
```

▷ **has_arg** vaut 0, 1 ou 2 pour respectivement indiquer l'absence, la nécessité ou la possibilité d'un paramètre

▷ Si **flag** vaut **NULL**, la fonction **getopt_long** renvoie **val**; sinon elle renvoie 0 et **flag** pointe vers **val**

▷ **val** correspond à la valeur à renvoyer

TRAITEMENT DES OPTIONS LONGUES DU PROGRAMME

▷ La fonction **getopt_long** permet de gérer les options courtes et longues

```
const char* optstring=":hi:o::";
const struct option lopts[] = {
  {"help", 0, NULL, 'h'},
  {"input", 1, NULL, 'i'},
  {"output", 2, NULL, 'o'},
  {NULL, 0, NULL, 0}
};
```

▷ La fonction **getopt_long** renvoie le caractère d'option

▷ Si **longindex** est non **NULL** et si une option longue est trouvée, alors ***longindex** correspond à l'indice de l'option dans le tableau

TRAITEMENT DES OPTIONS LONGUES DU PROGRAMME

```
const char* optstring=":hi:o::";
const struct option lopts[]=/*...*/;
int index=-1;
int val=getopt_long(argc,argv,optstring,lopts,&index);
while(val!=EOF){
  char msg[32];
  if(index==-1){
    sprintf(msg,"short option -%c",val);
  }else{
    sprintf(msg,"long option --%s",lopts[index].name);
  }
  switch(val){
    case 'h': printf("%s\n",msg); break;
    case 'o': printf("%s arg=%s\n",msg,optarg); break;
    case 'i': printf("%s arg=%s\n",msg,optarg); break;
    case ':': printf("argument missing for option %c\n",optopt);break;
    case '?': printf("unknown option %c\n",optopt); break;
  }
  index=-1;
  val=getopt_long(argc,argv,optstring,lopts,&index);
}
```

TRAITEMENT DES OPTIONS LONGUES DU PROGRAMME

▷ S'il n'y a pas d'ambiguïté, les options longues peuvent être abrégées

```
$>./a.out --help --he --h -h --i=toto --i tutu --output
long option --help
long option --help
long option --help
short option -h
long option --input arg=toto
long option --input arg=tutu
long option --output arg=(null)
```

DOGGY BAG

TO TAKE AWAY ...

- Les librairies standard fournissent un grand nombre d'outils bien utiles comme
  - La gestion de fonctions variadiques
  - L'accès aux variables d'environnement
  - Les crochets d'arrêt
  - Le traitement des options du programme

Questions?