Programmation en Langage C

Ph. Langevin

Année Universitaire 2008

Préambule

Développement d'une application

Symboles

Mémoire

Cadre de pile

La fonction main

Traitement des options

Programme

L'objectif de ce cours est de présenter quelques notions avancées concernant la programmation en langage C.

- `gcc`, `gdb`, `make`
- mémoire, adresse et cadre de pile.
- ligne de commande, entrées/sortie sur flux.
- allocation dynamique, liste chainée.

Pérequis : une connaissance des aspects syntaxiques du langage C.

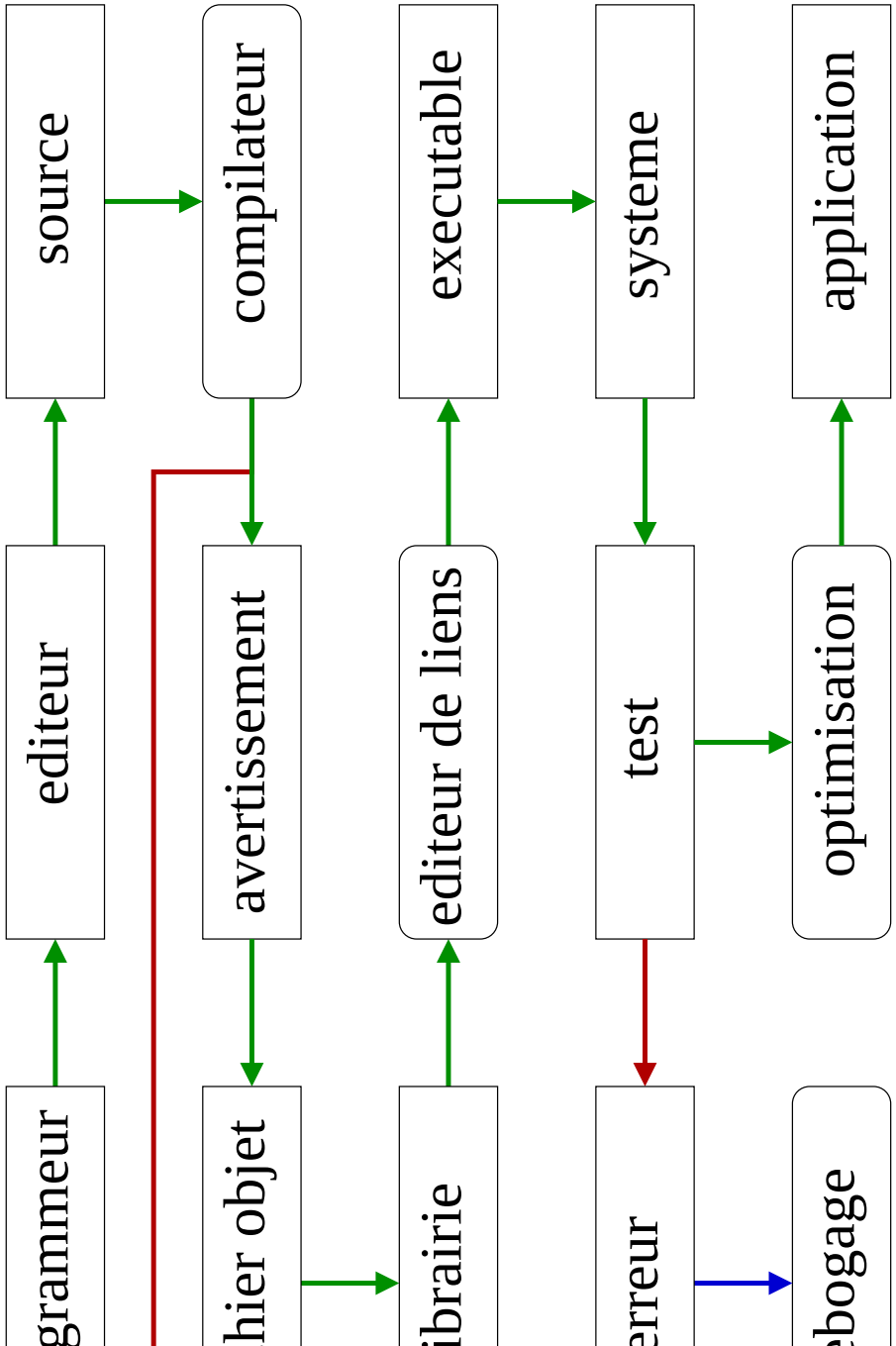
Travaux-Pratiques

[-n:]
Au cours des séances de travaux-pratiques, vous développerez une commande `stat.exe` pour faire des statistiques sur un fichier texte.

```
stat.exe [ options ]
```

- [-h] aide
- [-i] fichier source
- [-o] fichier destination
- [-f] compter les mots les plus fréquents
- [-g] mode graphique
- etc . . .

Développement d'une application



compilation

Le compilateur `gcc` traduit le *code source* dans un langage de plus bas niveau. Il détecte les *erreurs de syntaxes* et certaines *erreurs sémantiques*.

```
gcc -Wall -g prog.c -o prog.exe
```

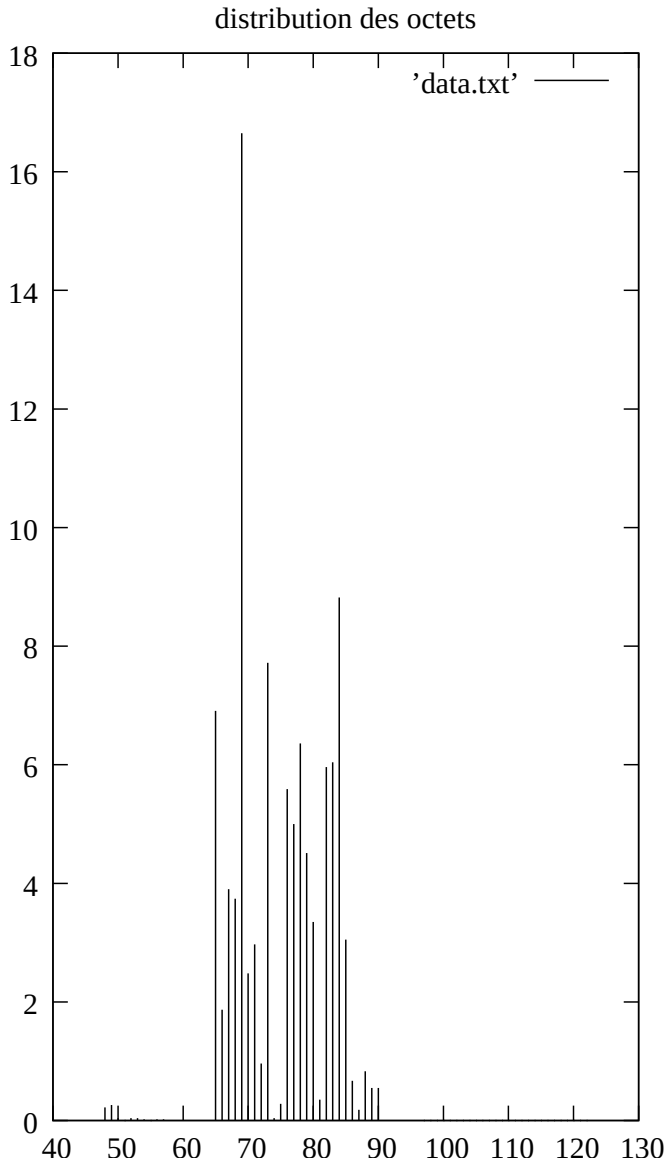
provoque la compilation de la source `prog.c`, pour construire un *exécutable* `prog.exe`. L'option -Wall est fondamentale, elle permet de détecter la plupart des erreurs sémantiques. Le programme est compilé avec l'option -g de déboggage. Il pourra etre exécuté pas á pas avec le deguggueur `gdb`.

makefile

```
all : data.pdf

prog.exe : prog.c
 gcc -Wall -g prog.c -o prog.exe

data.txt : ../prog.tex  prog.exe
 ./prog.exe -s ../prog.tex -au > data.txt

data.fig  : data.txt
 gnuplot data.plot

data.pdf : data.fig
 fig2dev -Lpdf data.fig data.pdf
clean :
 rm data.txt
```

Par défaut, la commande `make` recherche un fichier `makefile` dans le répertoire courrant puis exécute les commandes pour satisfaire la premiere cible, `all` dans l'exemple.

Exemple de projet



Symboles

Un fichier *source* en langage C est un ensemble de symboles représentants des :

- constante
- mot clef du langage
- variable
- fonction

Variable

Une variable est caractérisée par

- type
- adresse

Le type renseigne sur la taille t en octets des variables, la valeur correspondant à la paire (adresse, type) est la suite des t octets à l'adresse spécifiée.

Exemple

```
#include <stdio.h>
#define BASE 256
int main( int arc, char* argv[] )
{
   int    z;
   char *c;
   z=BASE * BASE * BASE + 2 * BASE * BASE + 3 * BASE + 4;
   printf("\nadresse de z : %p", &z);
   printf("\nvaleur  de z : %d",  z);
   c = ( char*) &z;
   printf("\n  adresse %p : %d", c, *c);
   c++;
   printf("\n  adresse %p : %d", c, *c);
   c++;
   printf("\n  adresse %p : %d", c, *c);
   c++;
   printf("\n  adresse %p : %d", c, *c);
   return 0;
}
```

Exemple

```
#define BASE 256
  z = BASE * BASE * BASE + 2 * BASE * BASE + 3 * BASE +
---

adresse de z : 0xbfd65d3c
valeur  de z : 16909060
  adresse 0xbfd65d3c : 4
  adresse 0xbfd65d3d : 3
  adresse 0xbfd65d3e : 2
  adresse 0xbfd65d3f : 1
```

Exemple sous gdb

```
break 8
run
x/4d &z
x/4c &z
x/4t &z
continue
quit
```

var.gdb.out

```
[?1034hBreakpoint 1 at 0x804836c: file var.c, line 8.

Breakpoint 1, main () at var.c:8
8    printf("\nadresse de z : %p", &z);
0xbffff43c: 16909060 8644304 -1073744800 -1073744712
0xbffff43c: 4 '\004' 3 '\003' 2 '\002' 1 '\001'
0xbffff43c: 00000100 00000011 00000010 00000001

adresse de z : 0xbffff43c
valeur  de z : 16909060
  adresse 0xbffff43c : 4
  adresse 0xbffff43d : 3
  adresse 0xbffff43e : 2
  adresse 0xbffff43f : 1
Program exited normally.
```

Zones mémoires

Un fichier exécutable comprend 4 zones mémoires

- le segment des données qui contient les variables dont les adresses sont déterminées lors de la compilation.
- le segment de code qui contient la liste des instructions.
- le segment de pile utilisé pour passer les arguments des procédures et fonctions.
- le tas qui contient les variables crées dynamiquement au cours de l'exécution du programme par un appel au gestionnaire de tas.

Cadre de Pile

- Appel :
  - PUSH : Empilement des arguments de la fonction
  - CALL : Empilement de l'adresse de retour.
  - Saut vers le code de la fonction.
- Prologue :
  - PUSH BP : Sauvegarde du registre de pile
  - BP = SP : Affectation du registre de base
  - DEC SP : Allocation des variables locales
- *Exécution de la fonction.*
- LEAVE : récupération du cadre de pile
- RET : retour à l'appelant
- Dépilage des arguments.
- Récupération du résulat.

Exemple de fonction

```
int proc( int x, int y, int z)
{
   int    tmp;
   tmp = x+y+z;
   return tmp;
}

int main( int argc, char* argv[] )
{ int t;

   t = proc( 1, 2, 3);

   return 0;
}
```

Code de proc

```
int proc( int x, int y, int z)
{
 8048348: 55                          push   %ebp
 8048349: 89 e5                       mov    %esp,%ebp
 804834b: 83 ec 10                    sub    $0x10,%esp
  int    tmp;
  tmp = x+y+z;
 804834e: 8b 45 0c                    mov    0xc(%ebp),%eax
 8048351: 03 45 08                    add    0x8(%ebp),%eax
 8048354: 03 45 10                    add    0x10(%ebp),%eax
 8048357: 89 45 fc                    mov    %eax,0xfffffffc(%
  return tmp;
 804835a: 8b 45 fc                    mov    0xfffffffc(%ebp),%
```

Appel de proc

```
  t = proc( 1, 2, 3);
 804837b: 6a 03                  push   $0x3
 804837d: 6a 02                  push   $0x2
 804837f: 6a 01                  push   $0x1
 8048381: e8 c2 ff ff ff         call   8048348 <proc>
 8048386: 83 c4 0c               add    $0xc,%esp
 8048389: 89 45 fc               mov    %eax,0xfffffffc(%

  return 0;
 804838c: b8 00 00 00 00         mov    $0x0,%eax
}
```

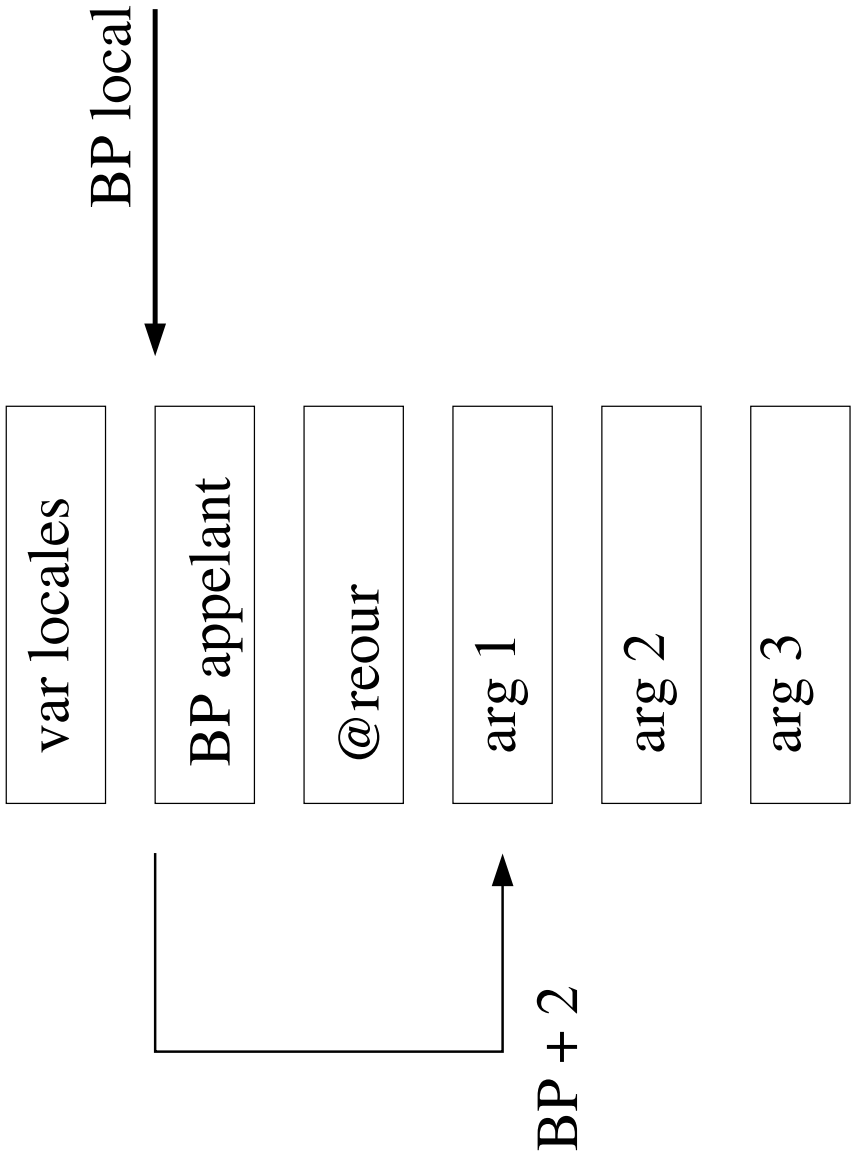
Inspection du code

```
break proc
run
print   &tmp
print   &x
print   &y
print   &z
x/6x    &tmp
print   $pc
print   $ebp
print  *( 0 + (int*) $ebp )
print  *( 1 + (int*) $ebp )
print  *( 2 + (int*) $ebp )
print  *( 3 + (int*) $ebp )
print  *( 4 + (int*) $ebp )
quit
```

Inspection du code

```
GNU gdb Red Hat Linux (6.3.0.0-1.21rh)
GDB is free software, covered by the GNU General Public
Breakpoint 1 at 0x804834e: file proc.c, line 5.
Breakpoint 1, proc (x=1, y=2, z=3) at proc.c:5
5     tmp = x+y+z;
$1 = (int *) 0xbfef35b8
$2 = (int *) 0xbfef35c4
$3 = (int *) 0xbfef35c8
$4 = (int *) 0xbfef35cc
0xbfef35b8: 0xbfef3811  0xbfef35f8  0x08048386  0x000000
0xbfef35c8: 0x00000002  0x00000003
$5 = (void *) 0x804834e
$6 = (void *) 0xbfef35bc
$7 = -1074842120
$8 = 134513542
$9 = 1
$10 = 2
$11 = 3
```

Cadre de pile



main.c

```
1  #include <stdio.h>
2
3
4  int main( int argc, char* argv[] )
5  {
6     int i;
7
8     printf("\nNombre d'arguments : %d", argc );
9
10    for( i = 0 ; i < argc ; i++)
11       printf("\n       argv[ %d ] = %s", i, argv[ i ] );
12
13    return 9;
14 }
```

Exemple d'exécution

```
./main.exe -a 1 -b2 -c
 ret=$?
 echo -e \\nretour=$ret
```

```
Nombre d'arguments : 5
        argv[ 0 ] = ./main.exe
        argv[ 1 ] = -a
        argv[ 2 ] = 1
        argv[ 3 ] = -b2
        argv[ 4 ] = -c
retour=9
```

getopt.c

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <unistd.h>
 4 int main( int argc, char *argv[] )
 5 { int opt;
 6   char *optlist ="n:b:h";
 7   int  base, val;
 8   while ( ( opt = getopt( argc, argv , optlist) ) > 0
   )
 9     switch( opt ){
10     case 'n': val  = atoi( optarg ); break;
11     case 'b': base = atoi( optarg ); break;
12     case 'h': printf("\nhelp:");
13     default : printf("\n%s:%s\n", argv[0], optlist);
14               exit ( 1 );
15     }
16
17   printf("n=%d b=%d", val, base);
18   printf("\n");
19   return 0;
```

Exemples d'exécution

```
./getopt.exe -n123 -b16
n=123 b=16
./getopt.exe -n -16
n=-16 b=1133391860
./getopt.exe -M5 -16

usage ./getopt.exe -> n:b:h
```