Le langage C

1 Présentation

C et UNIX sont etroitement lies.
Les grandes caracteristiques :

- la programmation structuree.
- la compilation separee des fonctions.
- la possibilite de travail a un bas niveau.

2 Organisation d'un programme C

premiere remarque :

on peut inserer des commentaires en
les mettant entre /* et */.

```
/* inclusion de fichiers textes par exemple des librairies etc...*/

#include <stdio.h>
#include "fichier"
```

la nuance entre ¡¿ et "" est dans le type de la recherche : avec ¡¿ recherche dans des endroits standarts (compilateur) avec "" recherche dans le repertoire source ou suivant des chemins normalises (PATH).

```
/* declaration de constantes*/

#define TAILLE 100
#define TEXTE "encore une erreur"

/* definition de types*/

typedef int tableau[TAILLE];

/* declaration de variables globales*/

int nombre,suite[20];
char caractere,*phrase;
tableau liste;
```

```
/*declaration de fonctions*/

int machin();
tableau *truc();

/* definition de fonctions*/

int machin(a,b)
char a,b;
{
.
.
.
}

/* le programme principal       */

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

/* fin                                                    */
```

3 Le strict minimum :

3.1 le plus petit programme :

```
main()
{
}
```

Ceci est un fichier texte, pour executer ce programme il faut le compiler (transformation du fichier source en fichier objet) (passage du fichier texte au code machine).

3.1.1 premierement :

on donne un nom au fichier texte avec une extension ".c" pour tre reconnu par le compilateur.

3.1.2 deuxiemement :

la compilation se fait avec la commande cc (C compiler) le fichier executable a par defaut a.out comme nom.

```
%cc rien.c
%a.out
%
```

Si on veut donner un nom au fichier executable on utilise l'option -o

```
%cc rien.c -o rien
%rien
%
```

Un exemple un peu plus explicite ;

```
main()
{
printf("ca marche\n");
}
```

Dans ce cas, on obtient :

```
%cc peu.c -o peu
%peu
ca marche
%
```

remarque :

printf est une fonction d'entree-sortie formatee qui se trouve dans la librairie stdio.h. Il est a noter que cette bibliotheque est implicitement liee au programme, un #include n'a pas ete necessaire. D'autre part
n est une constante caractere d'interligne.

```
#include <stdio.h>
int printf(format [ , arg ] ... )
char *format;
```

Pour plus de precisions, sous UNIX faire :

```
%man printf
```

(man comme manuel).

4 Les donnees :

les noms : ils commencent obligatoirement par une lettre. Ils sont composes de lettres, de chiffres et de _ (separateur).
Attention, C differencie majuscules et minuscules. Tres souvent les constantes sont en majuscules.

4.1 Les types standards :

char ; un octet representant un caractere, on le donne entre apostrophes , 'a' ou 'd' ou ' ',
char * ; cas particulier de la chaine de caracteres qui est donnee entre guillemets, "il fait beau", elle est stockee sous la forme d'un tableau (pointeur) de caracteres se terminant par '
0',

```
i       l               f       a       i       t               b       e
0       1       2       3       4       5       6       7       8       9
```

int ; entier ici sur 32 bits,de -2147483648 a 2147483647,
unsigned int ; entier non signe sur 32 bits, de 0 a 4294967295,
long int ; identique a int sur cette machine, sur d'autres int est sur 16 bits et long int sur 32,
float ; nombre en virgule flottante, ici sur 32 bits, entre 3.4e-38 et 3.4e38,
double ; nombre en virgule flottante sur 64 bits. de valeur absolue entre 1.7e-308 a 1.7e308.

4.2 declaration de constante :

```
#define UN 1                    /*entier donne en base dix*/

#define QUINZE 017      /* en octal 0 en debut de nombre*/

#define VINGT 0x14      /* en hexadecimal 0x en debut de nombre*/

#define QUATRE 4L               /* L en fin de nombre pour un long int*/


#define TOTO 't'
#define PHRASE "il fait beau"


#define PI 3.14
/*un point '.' pour signaler un nombre en virgule flottante, il est
considere automatiquement comme un double*/

#define NOMBRE 21.1234e-12
/*notation avec des puissances de dix*/
```

4.3 declaration de variables :

```
int i,j,k;
char *phrase;
double x,resultat;
```

4.4 notion de tableaux :

```
int serie[123];
/*tableau d'entiers de taille 123, les indices vont de 0 a 122*/
```

Attention : la declaration reserve une place memoire mais il n'y a pas de contrle de depassement. On peut tres bien ecrire serie[123123], il y a des risque d'ecriture dans des znes non reservees.

4.5 adresse d'une variable :

&i ou &resultat donnent les adresses des variables i et resultat. pour les variables du type tableau ou pointeur(*) le nom donne l'adresse
de debut, par exemple : phrase ou serie.
serie == &serie[0]
Autre remarque : il y a une difference entre :

```
char *string;
char string[20];
```

l'un est une chaine de caracteres sans place memoire allouee et l'autre est un tableau de caractere de taille maximale 20.

4.6 definition de type :

```
typedef char *string;
```

On vient de definir le type string comme etant une chaine de caracteres.
Ainsi :

```
string phrase,mots;
```

declare phrase et mots comme etant du type string

5 Les entrees-sorties standards :

On ne considere ici que la saisie clavier et l'affichage ecran.

5.1 les entrees :

```
#include <stdio.h>

  char getchar()

  scanf(format [ , pointer ] ... )
  char *format;

  char *gets(s)
  char *s;
```

5.1.1 des exemples :

```
main()

{
char s;

s=getchar();
printf("%c\n",s);
}
```

remarque : il est preferable de vider le buffer du clavier avec la commande fflush(stdin) avant d'utiliser la fonction gtetchar() .

```
main()

{
char *s,t[20];

gets(t);
s = t;
printf("%s\n",t);
printf("%s\n",s);
}

main()

{
char *s,t[20];
int i,j;

scanf("%s",t);
s = t;
printf("%s\n",t);
printf("%s\n",s);
}
```

5.1.2 Remarques :

Pour ces trois fonctions, l'entree se termine avec un retour a la ligne. gets permet de rentrer des chaines de caracteres comprenant des blancs. scanf permet via un formatage de rentrer plusieurs donnees separees par des blancs.

5.2 Les sorties :

```
#include <stdio.h>
 int printf(format [ , arg ] ... )
 char *format;

 int putchar(c)
```

```
char c;

puts(s)
char *s;
```

5.2.1 des exemples :

```
main()

{
char s;

s=getchar();
putchar(s);
}

main()

{
char s[20];

gets(s);
puts(s);
}

main()

{
char *s,t[20];
int i;
float k;

scanf("%d %f",&i,&k);
s=t;
printf("i = %5d\n",i);
printf("i = %-5d\n",i);
printf("i = %7.2f\n",k);
printf("i = %-7.2f\n",k);
}
```

5.2.2 Remarques :

gets renvoit la chaine de caracteres et un retour a la ligne, scanf renvoit le texte qui se trouve dans format avec des arguments inseres suivant un ordre donne.

5.3 Pour le format de scanf et de printf :

```
%d    A decimal int
```

```
          %u    An  unsigned  decimal  int
          %o    An octal integer
          %x    A hexadecimal integer
          %i    An integer
    interpreted according to C conventions: ‘‘0’’ implies octal; a
                ‘‘0x’’  implies  hexadecimal;  otherwise,
                  decimal.
          %e,%f
                A floating point number
          %s    A character string
          %c    A character
```

Les sequences d'echappement pour printf :

```
          \n              retour a la ligne
          \r              retour en debut de ligne
          \t              deplacement d'une tabulation
          \a              bip sonore.
```

6 Les expressions :

6.1 les operateurs :

```
=               assignation,
+               addition
-               soustraction
*               multiplication
/               division
%               reste division euclidienne
()              parentheses pour priorites des operations
```

6.2 Particularites du C :

```
--              decrementation de un
++              incrementation
op=     operation op et assignation
```

des exemples :

```
x=--t;  c'est-a-dire    x=t= t-1;
x=t++;  c'est-a-dire            x=t; t=t+1;
x*=t;   c'est-a-dire            x=x*t;
```

exemple :

```
main()

{
float x,t;

scanf("%f%f",&x,&t);
x*=t;
printf(" x = %f\n",x);
}
```

6.3 les relations :

```
<               strictement inferieur a
>               strictement superieur a
<=              inferieur ou egal a
>=              superieur ou egal a
==              egal a
!=              non egal a
```

exemple :

```
main()

{
printf(" faux = %d\n",(32 > 123));
printf(" vrai = %d\n",('a'=='a'));
}
```

On verifie qu'une expression vraie vaut 1 et qu'une expression fausse vaut 0.

6.4 operateurs logiques :

```
&&              et
||                      ou
!                       negation
```

exemples :

```
(x==y) && (x<=(t-1))
(x<0) || (t>= 23)
```

6.5 operateurs sur les bits :

```
>>              decalage a droite
<<                      decalage a gauche
&                       et
^                       ou exclusif
||                      ou
```

```
~                       negation
```

par exemple :

```
x=y<<2;                 equivaut a ,  x=y*4;
```

7 Les instructions :

7.1 generalites :

une instruction se termine par un ';' des instructions peuvent tre regroupees en une seule en utilisant des .

7.2 Les conditionnelles :

if(une expression) une instruction;
else une instruction;

exemple

```
main()

{
int i;

printf("entrez 1 ou 0 :");
scanf("%d",&i);
if (i==1)
printf(" faux = %d\n",(32 > 123));
else
printf(" vrai = %d\n",('a'=='a'));
}


switch(expression)
                {
                case valeur1:instruction;break;

                case valeuri:instruction;break;

                default:instruction;break;
                }
```

si expression=valeuri l'intruction correspondante est executee puis sortie, sinon execution de celle associee a default.

remarque :

s'il n'y a pas de break les autres cas sont examines.

exemple :

```
main()

{
int i;

printf("entrez 1 ou 0 :");
scanf("%d",&i);
switch(i)
        {
        case 1:
        printf(" faux = %d\n",(32 > 123));
        break;
        case 0:
        printf(" vrai = %d\n",('a'=='a'));
        break;
        default:printf("vous n'avez pas respecte les instructions");
        break;
}
}
```

7.3 les boucles :

while(expression)instruction;
tant que l'expression est vraie faire l'instruction.

exemple :

```
main()
{
char c;

c='o';
while((c=='o') || (c=='O'))
{
printf("encore o/n : ");
c=getchar();
getchar();
}
}
```

do instruction; while(expression);
faire l'instruction jusqu'a ce que l'expression soit fausse.

exemple :

```
main()
{
```

```
char c;

do
{
printf("encore o/n : ");
c=getchar();
getchar();
}
while((c=='o') || (c=='O'));
}
```

for(initialisation-contrle;test-contrle;modif-contrle) instruction;
peut se traduire par :

```
initialisation-contrle;
while(test-contrle)
        {
        instruction;
modif-contrle;
        }
```

exemple :

```
main()
{
char phrase[20];
int i;

printf("entrez une phrase :");
gets(phrase);
for(i=0;phrase[i]!='\0';++i)    putchar(phrase[i]);
}
```

8 les fonctions :

Pour simplifier la structure d'un programme il est agreable d'utiliser des fonctions

par exemple :

nous avons vu des fonctions predefinies d'entree-sortie

8.1 Declaration :

```
type nom(type1 var1,...);
```

8.2 le corps :

```
type nom(type1 var1,...)
{
declaration de variables locales;

instructions;
return(valeur);
}
```

remarque :

le type de la fonction est a prendre parmis les types de base;

exemple :

```
int addition();
int addition(x,y)
int x,y;
{
int c;

c=x+y;
return(c);
}

main()
{
int a,b;

printf("entrez un entier : ");
scanf("%d",&b);
a=addition(b,b);
printf("voici son double : %d \n",a);
}
```

exercice :

ecrire un programme comptant le nombre d'occurences des lettres d'une phrase.

8.3 Un peu plus loin, les arguments du main...

```
int main(argc,argv)
/*declaration des arguments*/

int argc;
char *argv[];
{
/*declaration d'une variable locale*/
int i;
```

```
printf("argc = %d\n",argc);
for(i=0;i<argc;++i)
        printf("argv[%d]=%s\n",i,argv[i]);
return(0);
}
```

Ce programme renvoit les arguments du programme principal. Par exemple :

```
% a.out titi tutu tyty toto
argc = 5
argv[0] = a.out
argv[1] = titi
argv[2] = tutu
argv[3] = tyty
argv[4] = toto
% echo $status
0
%
```

argv[] est un tableau de mots, argc donne le nombre de mots, le premier etant le nom de l'executable. Si on utilise la procedure return(), main()
renvoit un entier qui se loge dans une variable "status" du shell.Ceci peut permettre une verification de l'execution du programme.
Au sujet du printf(); dans le premier : on a "argc = %d
n" o %d represente le format d'ecriture (un entier en decimal) de argc. dans le second : on a "argv[%d] = %s
n" o %d represente le format d'ecriture de la variable i et %s represente le format d'ecriture de la
variable argv[i] .

8.4 etude d'un exemple :

```
/*****************************/
typedef int nombre;

nombre carre();
void square();
nombre cube();

nombre carre(a)
nombre a;
{
a=a*a;
return(a);
}

void square(pt_a)
nombre *pt_a;
```

```
{
*pt_a=(*pt_a)*(*pt_a);
}

nombre cube(a)
nombre a;
{
a=a*carre(a);
return(a);
}

main()
{
nombre x;

printf("entrez un nombre :");
scanf("%d",&x);
printf("\n carre(x) = %d  cube(x) = %d \n",carre(x),cube(x));
printf("et x = %d \n",x);
square(&x);
printf("apres square(x) x= %d \n",x);
}
/**********************************/
```

Dans la fonction carre le parametre est passe par valeur. La fonction utilise la valeur de la variable sans la modifier.
Le parametre de la fonction square est passe par adresse. La fonction square ne peut pas modifier l'adresse mais peut transformer le contenu.
Enfin, une fonction peut appeler d'autres fonctions, elle peut s'appeler
elle-mme, on parle d'appel recursif.
Dans le cadre d'appels recursifs il peut y avoir des probleme de taille de la pile (stack).

exemple :

```
int puissance(a,n)
int a,n;
{
int puis=1,parite;

parite = n&1;
if((n!=0)&&(n!=1))
{
n = (n>>1);
puis = puissance(a,n);
puis = puis * puis;
}
if(parite) puis = a * puis;
return(puis);
}
```

```
main()
{
int x,p,res;

printf("\nentrez un nombre :");
scanf("%d",&x);
printf("\nentrez un exposant : ");
scanf("%d",&p);
res = puissance(x,p);
printf("\nresultat = %d\n",res);
}
```

8.5 validite d'une variable :

Nous rencontrons des variables locales et des variables globales.
Une variable locale est declaree dans le corps d'une fonction. Cette variable existe lorsque l'on execute cette fonction. Sa duree de vie est
le temps d'execution de cette fonction.
Une variable globale est declaree en dehors de toute fonction. Sa duree de vie est celle de l'execution du programme.

exemple :

```
typedef double nombre;

nombre x;

nombre carre(a)
nombre a;
{
a=a*a;
return(a);
}

void square(pt_a)
nombre *pt_a;
{
*pt_a=(*pt_a)*(*pt_a);
}

void puis2()
{
x=x*x;
}

main()
{
nombre i;
```

```
printf("entrez un nombre : ");
scanf("%d",&i);
/* les trois solutions suivantes sont identiques*/
x = i;
x = carre(x);
printf("\n x = %d\n",x);
x = i;
square(x);
printf("\n x = %d\n",x);
x = i;
puis2();
printf("\n x = %d\n",x);
}
```

Dans la fonction "carre" l'argument est passe par valeur. La variable globale x est modifiee par assignation dans le programme principal.
Dans la fonction "square" l'argument est passe par adresse. La variable globale x est modifiee en tant que parametre.
Enfin la fonction "puis2" n'a pas d'argument, elle agit directement sur la variable globale x.