Structure d'un programme

```
#include <stdio.h>
▪ int main(void){ ... }
▪ int main(){ ... }
▪ int main(int argc, char *argv[]){ ... }
```

Compilation gcc =>Exécution

```
gcc monsource.c                 =>./a.out
gcc monsource.c –o monProg      => ./monProg
```

Déclarations

Variables

```
nonType nomVariable;
int nb;
float f1,f2;
char c[20];
```

Constantes

```
#define nomConstante valeurConstante
#define tva 19.6
```

Affectations

```
nomVariable=uneValeur
nb=5;
f1+=f2;
```

Opérateurs arithmétiques

+	addition
-	soustraction
*	multiplication
/	division
%	modulo

Opérateurs relationnels

==	égalité
!=	différence
<=	inférieur ou égal
>=	supérieur ou égal
<	inférieur
>	supérieur

Opérateurs logiques

&&	et
||	ou
!	non

Opérateurs d'affectation

=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
++	x++	x = x +1
--	x--	x = x – 1

Entrées/sorties

Ecriture

printf(format, exp_1, exp_2, ..., exp_n);

```
printf("%d",unEntier);
printf("%d\n%d",entier1,entier2);
```

%d	entier signé
%c	caractère
%s	chaîne
%f	réel (float ou double)

Lecture

int scanf(format, &donnee_1, &donnee_2, ..., &donnee_n);

&donnee correspond à l'adresse de donnee

```
scanf("%d %d %d", &entier1, &entier2, &entier3);
scanf("%c%*c",&char1)
```

si plusieurs scanf consécutifs, pour exclure le ENTREE

Lecture d'une chaîne

char *gets(char *s);

```
char ligne[10];
printf("Votre texte ? ");
gets(ligne);
```

Structure conditonnelle

```
si condition alors
        instructions
sinon
        instructions
Fsi
```

```
if(condition){
   instruction1;
   ...
   instructionN;
}
else{
   instruction1;
   ...
   instructionN;
}
```

```
if (a<b)
     a++;
else
     b++;
équivaut à
(a<b)?a++:b++
```

Structures itératives

Tant que condition répéter ...

```
while (condition) {
  instructions;
}
```

Répéter ... tant que condition (Exécuté au moins 1 fois)

```
do {
  instructions;
} while (condition);
```

Pour...(itérations à bornes connues)

```
for (expressionDepart ; expressionArrivee ; expressionPas) {
  instructions;
}
```

Débogage avec gdb

Compilation non optimisée

```
gcc –g monSource.h –o monProgramme
```

lancement

```
gdb monProgramme
```

Commandes

run	Exécution normale
break main	point d'arrêt sur la fonction main
break 11	point d'arrêt ligne 11
continue	poursuite du programme
next	pas à pas principal
step	pas à pas détaillé
print maVariable	affiche maVariable

Options de compilation pour GCC

-Wall -W provoque l'affichage de warnings supplémentaires. Les warnings sont une façon pour le compilateur de vous indiquer qu'il y a peut-être une erreur (mais le code compile quand même).
Il faut toujours essayer d'éliminer les warnings, même ceux qui sont un peu embêtants

-Wuninitialized, -O -Wuninitialized est une option inclue dans -Wall (donc pas besoin de l'activer si vous utilisez -Wall) mais qui a aussi besoin de -O. -Wuninitialized permet de générer des avertissements quand vous utilisez une variable avant de lui avoir donné une valeur. Malheureusement, il arrive que gcc se trompe (lire la man page pour mieux comprendre pourquoi et un example de cas).
-pedantic, -ansi Ces options activent la norme C89
-std=c99 Cette option active la norme C99.
-g pour une compilation sans optimisation
-lm avec math.h par exemple

Fichiers

Ouverture/Fermeture

Includes nécessaires

```
#include <stdlib.h>
#include <stdio.h>
```

Ouverture

```
FILE* fopen(const char* filename, const char* mode)
```

Mode peut prendre les valeurs :

r	Lecture seule
w	Ecriture seule
a	Ajout (à la fin)
r+	Lecture/écriture
w+	Lecture/écriture avec suppression au préalable

Exemple :

```
FILE* monFichier=NULL ;
monFichier=fopen("dossierLinux/monfichier.zzz","r+") ;
If (monFichier !=NULL)
{
        //Lecture ou écriture
        fclose(monFichier);
}
else
        printf("ouverture impossible");
```

Ecriture dans un fichier

fputc	Un caractère
fputs	Une chaîne
fprintf	Chaîne formatée (comme printf)

Exemple :

```
//on suppose monFichier ouvert en écriture
fputc('A',monFichier) ;
fputs("une chaîne\nde caractères",monFichier) ;
fprintf(monfichier,"x :%d,y :%d",x,y) ;//x et y ont deux entiers déclarés
```

Lecture d'un fichier

fgetc	Un caractère
fgets	Une chaîne
fscanf	Chaîne formatée (voir scanf)

Attention, avec les chaînes, il vaut mieux utiliser fgets que fscanf, fscanf s'arrête en effet au premier espace trouvé…

Exemple : Lecture complète d'un fichier caractère par caractère

```
//on suppose monFichier ouvert en écriture
do{
        currentChar=fgec(monfichier) ;
        printf("%c",currentChar);
}while (currentChar!=EOF);
fclose(monFichier);
```

Exemple : Lecture complète d'un fichier ligne par ligne

```
#define MAX_SIZE 500
…
char chaine[MAX_SIZE]=””;

//on suppose monFichier ouvert en écriture
while(fgets(chaine,MAX_SIZE,fichier) !=NULL){
        printf("%s",chaine) ;
}
fclose(monFichier);
```

Exemple : Exemple de lecture d'un fichier formaté avec fprintf

```
//on suppose monFichier ouvert en écriture
int count=0 ;
int i ;int* tabInt=NULL ;char* tabString=NULL ;
FILE* monFichier=NULL ;
monFichier =fopen("file.zzz","r") ;
if(monFichier !=NULL){
        fscanf(monfichier,"%d",&count) ;//lecture du nombre de valeurs
        tabint=malloc(count*sizeof(int)) ;
        tabString=malloc(count*sizeof(char));
        for(i=0 ;i<count ;i++){
                fscanf(monfichier,“%d %s”,&tabInt[i],&tabString[i]);
        }
        fclose(monFichier);
}
fclose(monFichier);
free(tabint);free(tabString);
```

Déplacement de la position dans le fichier :

```
int fseek(FILE* file,long position,int origine)
```

Permet de se déplacer de position dans file par rapport à origine.
Origine peut prendre les valeurs suivantes :
SEEK_SET Début du fichier
SEEK_CUR Position courante
SEEK_END Fin du fichier

Exemple :
//Se déplace de 10 caractère en arrière de la position courante

```
Fseek(monFichier,-10,SEEK_CUR) ;
```

Structures

Exemple :

//Définition d'une structure personne :

```
struct Personne{
        char nom[20] ;
        int age ;
} ;
```

//Utilisation :

```
struct Personne unePersonne ;
unePersonne.nom="toto" ;
unePersonne.age="5" ;
```

L'utilisation de typedef permet d'alléger l'utilisation :

```
Typedef struct Personne Personne ;
//Définition de la structure personne :
struct Personne{
        char nom[20] ;
        int age ;
} ;
```

//Utilisation simplifiée :

```
Personne unePersonne ;
//Autre possibilité d'affectation :
unePersonne={"toto",5} ;
```

Création d'une procédure d'initialisation :

```
Void initPersonne(Personne* p){
        p->nom="nobody";
        p->age=0;
}
```

// la notation p->nom équivaut à (*p).nom

Appel de la procédure :

```
Personne unePersonne ;
initPersonne(&unePersonne) ;
```

Enumérations

Elles permettent de créer un nouveau type à partir de listes de valeurs possibles.

Exemple :

```
typedef enum Periode Periode ;
enum Periode{
        MATIN,MIDI,SOIR
} ;
```

Utilisation :

```
Periode unePeriode=MATIN ;
```

Inclusions dans fichier header

```
#ifndef MONFICHIER_H
#define MONFICHIER_H

// contenu du fichier h
//prototypes
#endif
```

Allocation dynamique de mémoire pour un tableau :

```
int size=0;
int *tab=NULL ;
printf("Entrez la taille :") ;
scanf("%d",&size);
tab=malloc(size*sizeof(int));
if(tab==NULL)
        exit(0);//sortie du prog si allocation non réussie
//Utilisation du tableau
for…
…
free(tab) ;//libération de la mémoire allouée
```

Utiliser calloc pour intialiser la mémoire du tableau

Date et heure

Date et heure courante :

```
include  <time.h>
time_t t=time(NULL);
```

Décomposition d'une date :

```
Time_t ts=time(NULL) ;
struct tm * t;
t=gmtime(&ts);
t->tm_sec ;//secondes
t->tm_min ;//minutes
t->tm_hour ;//heures
t->tm_mday ;//numéro du jour dans le mois
t->tm_mon ;//mois
t->tm_year ;//année
t->tm_wday ;//jour de la semaine
t->tm_tm_yday ;//jour depuis le 1er janvier
```

Formater une date (tm) :
Size_t strftime(char *s,size_t max,const char *format, const struct tm *tm) ;
max correspond à la taille maximale de la chaîne remplir
Exemples pour format :

%A %d %B %y	Vendredi 4 décembre 2009
%X	11 :50 :27

Convertir une date en chaîne :

```
Ctime($ts) ;
```

Convertir une structure tm en time_t :

```
Time_t t=Mktime(&tm) ;
```

Calculs sur dates (time_t):

```
difftime(t1,t2) ;//Calcule la différence en secondes entre t2 et t1
```