Entrées/sorties

Affichage écran
Chaine de caractères: stockage, bonnes pratiques
Ecriture/Lecture texte
Ecriture/Lecture fichiers (ASCII)

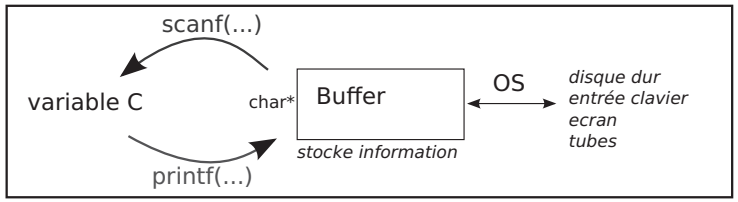
Entrees/sorties

Affichage/lecture écran/fichier

Ecran:
Information utilisateur
Debug d'un programme
Communication avec l'utilisateur

Fichier:
Suivi execution (fichier log)
Sauvegarde information (disque=mémoire non volatile)
Communication entre programme (disque=mémoire partagée)

En C: écriture fichier // écriture écran

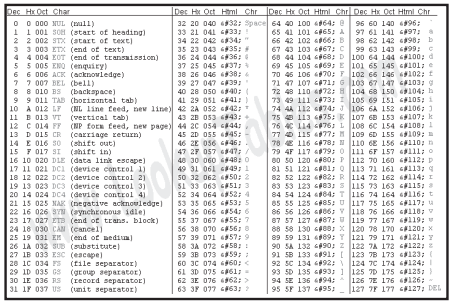


Entrees/sorties

Rappels caractère *char*

Un caractère (char) = nombre entre 0-256

1 octet
= 1 emplacement mémoire
= 2 caractères hexa



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

ex.

```
char c='M';
```

4D

```
int value=(int)c;
```

77

Caractère: entre ' '

Entrées/sorties

Affichage écran
→ **Chaine de caractères: stockage, bonnes pratiques**
Ecriture/Lecture texte
Ecriture/Lecture fichiers (ASCII)

Entrees/sorties

Rappels chaine de caractère

const char* , char []

Chaine de caractère = suite de caractères

```
char nom[10];
nom[0]='m';
nom[1]='a';
nom[2]='i';
nom[3]='s';
nom[4]='o';
nom[5]='n';
```

reserve 10 emplacements mémoire

'm'	'a'	'i'	's'	'o'	'n'	??	??	??	??

en mémoire

6D	61	69	73	6F	6E	??	??	??	??

Entrees/sorties

Rappels chaine de caractère

const char* , char []

Chaine de caractère = suite de caractères

'm'	'a'	'i'	's'	'o'	'n'	??	??	??	??

en mémoire

6D	61	69	73	6F	6E	??	??	??	??

nom correspond à la
première case de la chaine

nom = pointeur sur la 1ere case

```c
#include <stdio.h>

int main()
{
  char nom[10];
  nom[0]='m';
  nom[1]='a';
  nom[2]='i';
  nom[3]='s';
  nom[4]='o';
  nom[5]='n';

  printf("%s\n",nom);
  return 0;
}
```


Entrees/sorties

Rappels chaine de caractère

const char* , char []

Chaine de caractère = suite de caractères

'm'	'a'	'i'	's'	'o'	'n'	??	??	??	??

en mémoire

6D	61	69	73	6F	6E	??	??	??	??

Comment savoir où s'arrêter ?

```c
#include <stdio.h>

int main()
{
  char nom[10];
  nom[0]='m';
  nom[1]='a';
  nom[2]='i';
  nom[3]='s';
  nom[4]='o';
  nom[5]='n';

  printf("%s\n",nom);
  return 0;
}
```


Entrees/sorties

Rappels chaine de caractère

const char* , char []

Fin de chaine = principe de **sentinelle de fin**


```c
#include <stdio.h>

int main()
{
  char nom[10];
  nom[0]='m';
  nom[1]='a';
  nom[2]='i';
  nom[3]='s';
  nom[4]='o';
  nom[5]='n';
  nom[6]='\0';

  printf("%s\n",nom);
  return 0;
}
```

'm'	'a'	'i'	's'	'o'	'n'	'\0'	??	??	??

en mémoire

6D	61	69	73	6F	6E	00	??	??	??

Affiche maison

Comportement déterministe

Entrees/sorties

Rappels chaine de caractère

const char* , char []

Fin de chaine = '\0'

Toujours

Toujours

=> Toujours prévoir une case pour placer '\0'

Toujours

Fonction pour trouver la taille d'un mot:

```
int calcul_taille(char mot[])
{
  char *pointeur_debut=mot;
  char *pointeur_fin=pointeur_debut;

  while(*pointeur_fin!='\0')
    ++pointeur_fin;

  return (int)(pointeur_fin-pointeur_debut);
}
```


Entrees/sorties

Rappels chaine de caractère

const char* , char []

Fonction pour trouver la taille d'un mot:

```
int calcul_taille(char mot[])
{
  char *pointeur_debut=mot;
  char *pointeur_fin=pointeur_debut;

  while(*pointeur_fin!='\0')
    ++pointeur_fin;

  return (int)(pointeur_fin-pointeur_debut);
}
```

Attention!! Si '\0' est oublié !!!

=> Comportement indeterminé

Entrees/sorties

Rappels chaine de caractère

const char* , char []

Les fonctions de manipulation de chaine: #include <string.h>

str<...>

strcpy	strncpy	→ copie
strcmp	strncmp	→ compare
strcat	strncat	→ concatenation
...	...	
jusqu'à '\0'	*jusqu'à '\0'* *maximum n caractères*	

Bonnes pratiques: Chaine de caractères

Définir les chaines "inline" en tant que char[]

char[]="ma_chaine"

Définir les pointeurs (char*) en tant que constantes

const char*

Attention:

```
char mot[]="abricot";
mot[2]='l';
```

OK

```
char *mot="abricot";
mot[2]='l';
```

KO

pointeur vers
une chaine constante

Bonnes pratiques: Chaine de caractères

Toujours utiliser les fonctions à tailles limitées:

str**n**<...>

~~strcpy(mot_1,mot_2);~~
~~strcmp(mot_1,mot_2);~~
~~strcat(mot_1,mot_2);~~

strncpy(mot_1,mot_2,n_max);
strncmp(mot_1,mot_2,n_max);
strncat(mot_1,mot_2,n_max);

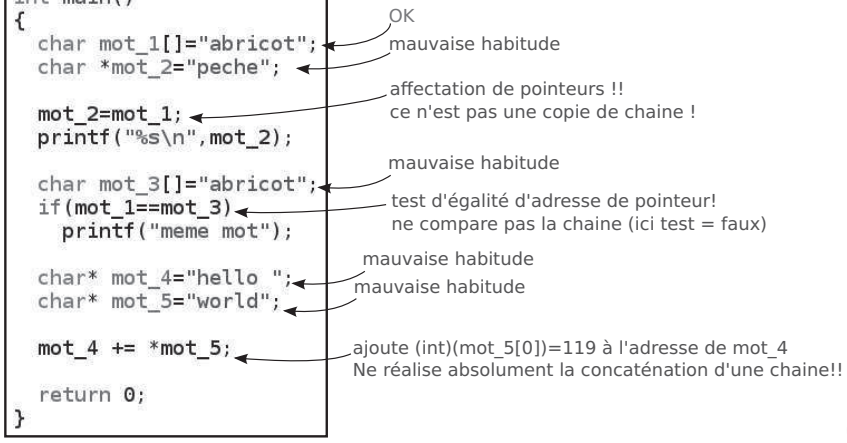
+ sécurisé
+ debug plus aisé

Note:
Le C n'est pas le langage approprié pour le traitement complexe de chaine de caractères

Bonnes pratiques: Chaine de caractères

Ne pas utiliser d'opérateurs sur des chaine de caractères !!!

```c
#include <stdio.h>

int main()
{
  char mot_1[]="abricot";
  char *mot_2="peche";

  mot_2=mot_1;
  printf("%s\n",mot_2);

  char mot_3[]="abricot";
  if(mot_1==mot_3)
    printf("meme mot");

  char* mot_4="hello ";
  char* mot_5="world";

  mot_4 += *mot_5;

  return 0;
}
```



Bonnes pratiques: Chaine de caractères

Ne pas utiliser d'opérateurs sur des chaine de caractères !!!

```c
#include <stdio.h>

int main()
{
  char mot_1[]="abricot";
  char *mot_2="peche";

  mot_2=mot_1;
  printf("%s\n",mot_2);

  char mot_3[]="abricot";
  if(mot_1==mot_3)
    printf("meme mot");

  char* mot_4="hello ";
  char* mot_5="world";

  mot_4 += *mot_5;

  return 0;
}
```

Attention!!!

gcc -Wall -Wextra
ne donne aucun Warning !

Pourtant ce programme ne fait probablement pas ce que vous souhaitez!

Bonnes pratiques: Chaine de caractères

Ne pas utiliser d'opérateurs sur des chaines de caractères !!!

```c
#include <stdio.h>

int main()
{
  char mot_1[]="abricot";
  char *mot_2="peche";

  mot_2=mot_1;
  printf("%s\n",mot_2);

  char mot_3[]="abricot";
  if(mot_1==mot_3)
    printf("meme mot");

  char* mot_4="hello ";
  char* mot_5="world";

  mot_4 += *mot_5;

  return 0;
}
```

OK

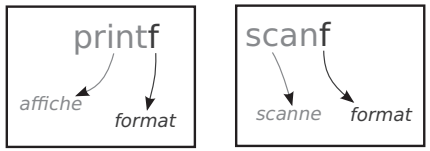
```c
#include <stdio.h>
#include <string.h>

#define N 10

int main()
{
  char mot_1[N]="abricot";
  char mot_2[N]="peche";

  strncpy(mot_1,mot_2,N);

  char mot_3[N]="abricot";
  if(strncmp(mot_1,mot_3,N)==0)
    printf("meme mot");

  char mot_4[N]="hello";
  char mot_5[N]=" world";
  strncat(mot_4,mot_5,N);

  return 0;
}
```

Entrées/sorties

Affichage écran
Chaine de caractères: stockage, bonnes pratiques
→ **Ecriture/Lecture texte**
Ecriture/Lecture fichiers (ASCII)

Printf / Scanf

Fonctions de conversions



Principe:

a=**printf**(<format>,variables ...)
- affiche sur la sortie standard (stdout) — *en général: la ligne de commande écran*
- renvoie le nombre de caractères formattés

a=**scanf**(<format>,variables ...)
- récupère de l'entrée standard (stdin) — *en général: la ligne de commande clavier*
- renvoie le nombre d'arguments formattés

Printf / Scanf

ex

```
int a=12;
printf("%d",a); //affiche: 12

int b=0xAAFF4E3D;
printf("%x",b); //affiche aaff4e3d

char c='e';
printf("%c",c); //affiche: e

char mot[]="bonjour a tous";
printf("%s",mot); //affiche: bonjour a tous

float x=1.25;
printf("%f",x); //affiche 1.250000
printf("%1.3f",x); //affiche 1.250
```


Printf / Scanf

ex

printf (et scanf) accepte un nombre d'argument variable
=> Fonction variadiques

```
printf("\n");
printf("Je suis %s a %c%c%c \n","etudiant",'C','P','E');
printf("j'ai %d ans\n",15);

printf("1.3+1.7=%1.2f\n",1.3+1.7);
```

```
int a=printf("\n");
int b=printf("Je suis %s\n","heureux");
int c=printf("%d-%d=%d\n",5,7,5-7);
int d=printf("%d+4=%d\n",3);


printf("%d %d %d %d\n",a,b,c,d);
```

```
Je suis heureux
5-7=-2
3+4=1522178016
1 16 7 15
```

oublie argument
=>comportement indeterminé
(=> Warnings!)

Printf / Scanf

ex

```
int main()
{

  int a=0;
  scanf("%d",&a);

  float x=0.0;
  scanf("%f",&x);

  char c='a';
  scanf(" %c",&c);

  char buffer[50];
  scanf("%50s",buffer);

  printf("%d\n%f\n%c\n%s\n",a,x,c,buffer);

  return 0;
}
```

$ ./mon_executable
> 1
> 1.25
> a
> bonjour

entrées utilisateur

affiche:

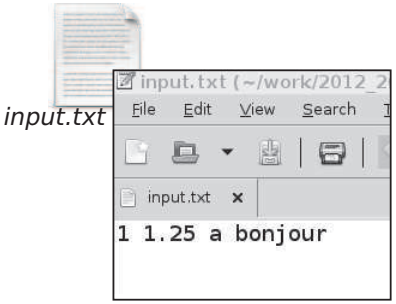
```
1
1.250000
a
bonjour
```


Printf / Scanf

ex

```
int main()
{
  int a=0;
  scanf("%d",&a);

  float x=0.0;
  scanf("%f",&x);

  char c='a';
  scanf(" %c",&c);

  char buffer[50];
  scanf("%50s",buffer);

  printf("%d\n%f\n%c\n%s\n",a,x,c,buffer);

  return 0;
}
```

Astuce:
entrée utilisateur:



$ ./mon_executable < input.txt

```
1
1.250000
a
bonjour
```

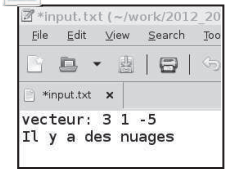
l'entrée standard deviens le fichier!
=> scanf viens "lire" dans le fichier

Printf / Scanf

ex

```
int main()
{
  int x=0,y=0,z=0;
  scanf("vecteur: %d %d %d\n",&x,&y,&z);

  char buffer[256];
  scanf("Il y a des %256s\n",buffer);

  printf("(%d,%d,%d) , %s\n",x,y,z,buffer);

  return 0;
}
```

input.txt

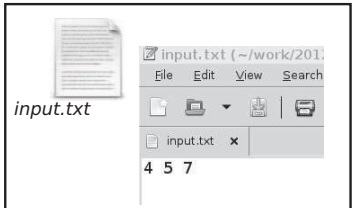


$ ./mon_executable < input.txt
> (3,1,-5) , nuages

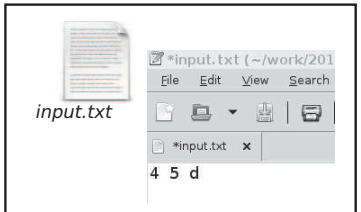
Printf / Scanf

ex

```
int main()
{
  int x=0,y=0,z=0;

  int valeur=scanf("%d %d %d",&x,&y,&z);

  printf("%d\n",valeur);

  return 0;
}
```



./ mon_executable
> 3



./ mon_executable
> 2

Printf / Scanf

Formatage sur d'autres entrées/sorties

sprintf / sscanf

string: chaine de caractères

Principe:

a=**sprintf**(buffer,<format>,variables ...)

affiche dans le buffer

renvoie le nombre de caractères formattés

a=**sscanf**(buffer,<format>,variables ...)

récupère du buffer

renvoie le nombre d'arguments formattés

Printf / Scanf

```
int main()
{
  char buffer[512];
  float x=cos(0.5);
  sprintf(buffer,"%s, (%d,%2.3f)","bonjour",1+4,x);

  printf("%s\n",buffer);

  return 0;
}
```

$ ./mon_executable
> bonjour, (5,0.878)

Printf / Scanf

Analyse d'une chaine de caractères

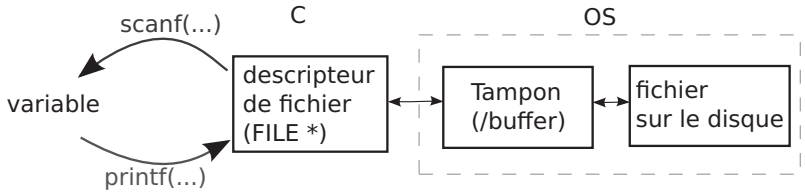
```
int main()
{
  char buffer[512]="Il fait beau ce matin. Il est 13 heures et fait 25.1 degres \n";

  char temps[25];
  int heure=0;
  float temperature=0.0;
  sscanf(buffer,"Il fait %s ce matin. Il est %d heures et fait %f degres \n",&temps,&heure,&temperature);

  printf("%s %d %1.2f\n",temps,heure,temperature);

  return 0;
}
```

$ ./mon_executable
> beau 13 25.10

Entrées/sorties

Affichage écran
Chaine de caractères: stockage, bonnes pratiques
Ecriture/Lecture texte
→ **Ecriture/Lecture fichiers (ASCII)**

Ecriture/lecture fichier



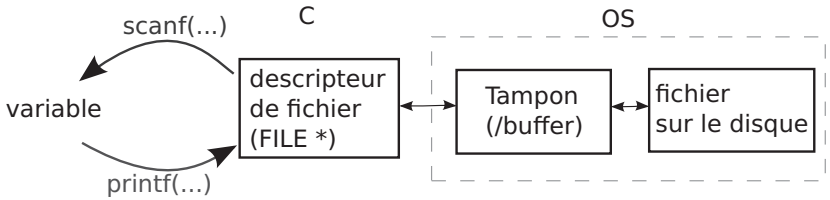
Ouvrir un fichier (pour l'écriture)

```
FILE* mon_descripteur=NULL;  //pointeur vers descripteur de fichier

mon_fichier=fopen("mon_fichier.txt","w"); //ouverture en écriture (w)
                                          //créé le fichier si il n'existe pas
if(mon_fichier==NULL) //gestion d'erreur
  {printf("Erreur ouverture fichier [mon_fichier.txt]\n");abort();}
```

Ouvrir un fichier (pour la lecture)

```
FILE* mon_descripteur=NULL;  //pointeur vers descripteur de fichier

mon_fichier=fopen("mon_fichier.txt","r"); //ouverture en lecture (r)
                                          //le fichier doit déjà exister
if(mon_fichier==NULL) //gestion d'erreur
  {printf("Erreur ouverture fichier [mon_fichier.txt]\n");abort();}
```

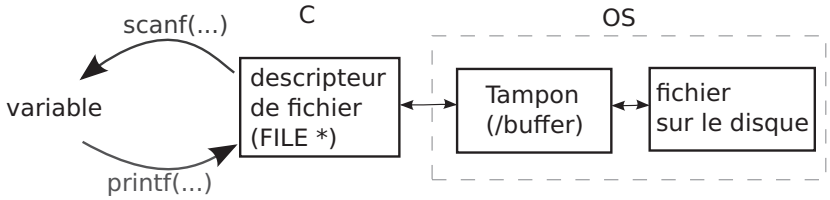

Ecriture/lecture fichier



Fermer un fichier

```
int valeur=fclose(mon_descripteur); //fermeture fichier
if(valeur!=0)//gestion d'erreur
  {printf("Erreur fermeture fichier \n");abort();}
```

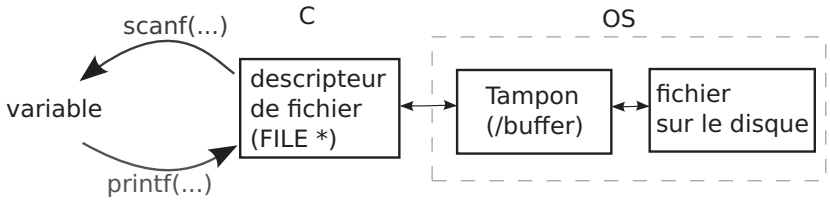

Ecriture/lecture fichier



Ecrire dans un fichier *fprintf(...)*

```
int valeur=fprintf(mon_descripteur,"%s","Mon texte a moi\n");

if(valeur!=1) //gestion d'erreur
  {printf("Erreur ecriture fichier\n");abort();}
```


Ecriture/lecture fichier



Lire dans un fichier *fscanf(...)*

```
char buffer[25]; //prepare un buffer de 25 cases

//lit au plus 25 caracteres
int valeur=fscanf(mon_descripteur,"%25s\n",buffer);

if(valeur!=1)//gestion d'erreur
  {printf("Erreur lecture fichier\n");abort();}
```

Ecriture/lecture fichier

Lecture/ecriture par fprintf(...) et fscanf(...)

Similaire à printf(...), scanf(...)

ecrit sur la sortie standard
(stdout => ecran)

lit sur l'entrée standard
(stdin => clavier)

Exemples minimalistes fonctionnels :
(Attention: sans gestion d'erreur)

```c
#include <stdio.h>

int main()
{
  FILE *fid=fopen("mon_fichier.txt","w");
  fprintf(fid,"prix carottes : 15 euros \n");
  fclose(fid);

  return 0;
}
```

lecture

```c
#include <stdio.h>

int main()
{
  char legume[128];
  int prix=0;


  FILE *fid=fopen("mon_fichier.txt","r");
  fscanf(fid,"prix %128s : %d euros \n",legume,&prix);
  fclose(fid);

  printf("les %s coutent %d euros\n",legume,prix);

  return 0;
}
```

ecriture

Ecriture/lecture fichier

Exemples minimalistes fonctionnels : Ecriture
(Attention: sans gestion d'erreur)

```c
#ifndef INCLUDE_GUARD_PERSONNE_H
#define INCLUDE_GUARD_PERSONNE_H

struct personne
{
  char nom[15];
  char fonction[15];
  int salaire;
  int anciennete;
};

#endif
```

personne.h

```c
#include <stdio.h>
#include "personne.h"


int main()
{
  //base de donnees
  struct personne employe[4]={{"Charlie","Directeur",55000,3},
                              {"Robert","DRH",45000,15},
                              {"Brigitte","R&D",38000,4},
                              {"Raymond","Technicien",23000,25}};

  //ouverture fichier
  FILE *fichier=fopen("ma_base.txt","w");

  //ecriture fichier
  int k=0;
  for(k=0;k<4;++k)
    {
      fprintf(fichier,"%s %s %d %d \n",employe[k].nom,
          employe[k].fonction,
          employe[k].salaire,
          employe[k].anciennete);
    }

  //fermeture fichier
  fclose(fichier);

  return 0;
}
```

ecrivain.h

Ecriture/lecture fichier

Exemples minimalistes fonctionnels : Ecriture
(Attention: sans gestion d'erreur)

```c
#ifndef INCLUDE_GUARD_PERSONNE_H
#define INCLUDE_GUARD_PERSONNE_H

struct personne
{
  char nom[15];
  char fonction[15];
  int salaire;
  int anciennete;
};

#endif
```

personne.h

```c
#include <stdio.h>
#include "personne.h"


int main()
{
  //base de donnees
  struct personne employe[4]={{"Charlie","Directeur",55000,3},
                              {"Robert","DRH",45000,15},
                              {"Brigitte","R&D",38000,4},
                              {"Raymond","Technicien",23000,25}};

  //ouverture fichier
  FILE *fichier=fopen("ma_base.txt","w");

  //ecriture fichier
  int k=0;
  for(k=0;k<4;++k)
    {
      fprintf(fichier,"%s %s %d %d \n",employe[k].nom,
          employe[k].fonction,
          employe[k].salaire,
          employe[k].anciennete);
    }

  //fermeture fichier
  fclose(fichier);

  return 0;
}
```

ecrivain.h

ma_base.txt

```
Charlie Directeur 55000 3
Robert DRH 45000 15
Brigitte R&D 38000 4
Raymond Technicien 23000 25
```


Ecriture/lecture fichier

Exemples minimalistes fonctionnels : Lecture
(Attention: sans gestion d'erreur)

```c
#ifndef INCLUDE_GUARD_PERSONNE_H
#define INCLUDE_GUARD_PERSONNE_H

struct personne
{
  char nom[15];
  char fonction[15];
  int salaire;
  int anciennete;
};

#endif
```

personne.h

```c
#include <stdio.h>
#include "personne.h"


int main()
{
  //base de donnees
  struct personne employe[4];

  //ouverture fichier
  FILE *fichier=fopen("ma_base.txt","r");

  //ecriture fichier
  int k=0;
  for(k=0;k<4;++k)
  {
      fscanf(fichier,"%15s %15s %d %d \n",employe[k].nom,
             employe[k].fonction,
             &employe[k].salaire,
             &employe[k].anciennete);
  }

  //fermeture fichier
  fclose(fichier);

  return 0;
}
```

Ecriture/lecture fichier

Exemple ecriture fichier avec gestion d'erreur

```c
int main()
{
  //base de donnees
  struct personne employe[4]={{"Charlie","Directeur",55000,3},
                              {"Robert","DRH",45000,15},
                              {"Brigitte","R&D",38000,4},
                              {"Raymond","Technicien",23000,25}};

  //ouverture fichier
  char nom_fichier[]="ma_base.txt";

  FILE *fichier=NULL;
  fichier=fopen(nom_fichier,"w");
  if(fichier==NULL)
  {printf("Erreur ouverture fichier %s\n",nom_fichier);abort();}

  //ecriture fichier
  int k=0;
  for(k=0;k<4;++k)
  {
      int verif=fprintf(fichier,"%s %s %d %d \n",employe[k].nom,
                        employe[k].fonction,
                        employe[k].salaire,
                        employe[k].anciennete);
      if(verif<=0)
      {printf("Erreur ecriture fichier %s\n",nom_fichier);abort();}
  }

  //fermeture fichier
  int ret=fclose(fichier);
  if(ret!=0)
  {printf("Erreur ouverture fichier %s\n",nom_fichier);abort();}
  fichier=NULL;

  return 0;
}
```


Ecriture/lecture fichier

Exemple lecture fichier avec gestion d'erreur

```c
int main()
{
  //base de donnees
  struct personne employe[4];

  //ouverture fichier
  char nom_fichier[]="ma_base.txt";
  FILE *fichier=NULL;
  fichier=fopen(nom_fichier,"r");
  if(fichier==NULL)
  {printf("Erreur ouverture fichier %s\n",nom_fichier);abort();}

  //ecriture fichier
  int k=0;
  for(k=0;k<4;++k)
  {
      int verif=fscanf(fichier,"%15s %15s %d %d \n",employe[k].nom,
                        employe[k].fonction,
                        &employe[k].salaire,
                        &employe[k].anciennete);
      if(verif!=4)
      {printf("Erreur lecture fichier %s\n",nom_fichier);abort();}
  }

  //fermeture fichier
  int ret=fclose(fichier);
  if(ret!=0)
  {printf("Erreur ouverture fichier %s\n",nom_fichier);abort();}
  fichier=NULL;

  return 0;
}
```


Bonnes pratiques Ecriture/Lecture

Toujours vérifier que fopen s'est bien déroulé

Toujours

Toujours

Toujours

* erreur nom
* chemin invalide
  (chemin locale dépend de l'endroit de l'execution!)
* fichier non existant
* fichier bloqué par une autre application

Bonnes pratiques Ecriture/Lecture

Toujours vérifier que fopen s'est bien déroulé

Toujours

Toujours

Toujours

```c
FILE *fid=NULL;
fid=fopen(...);
if(fid==NULL)
{
   ...
}
```

* erreur nom
* chemin invalide
  (chemin locale dépend de l'endroit de l'execution!)
* fichier non existant
* fichier bloqué par une autre application

Bonnes pratiques Ecriture/Lecture

scanf("%s",...) => s'arrête au premier espace rencontré

↳ Pour lire une ligne complète: fgets

fichier string

fgets(char buffer[], int taille_max, FILE *fichier)

lecture fichier

```c
#define TAILLE_MAX 128

int main()
{
  FILE *fichier=NULL;
  fichier=fopen("mon_fichier","r");

  //lit a partir d'un fichier
  char buffer_fichier[TAILLE_MAX];
  fgets(buffer_fichier,TAILLE_MAX,fichier);

  fclose(fichier);
}
```

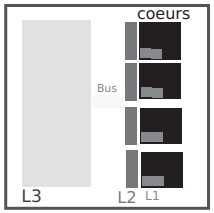
lecture clavier

```c
#define TAILLE_MAX 128

int main()
{
  //lit entree standard (clavier par defaut)
  char buffer_fichier[TAILLE_MAX];
  fgets(buffer_fichier,TAILLE_MAX,stdin);
}
```

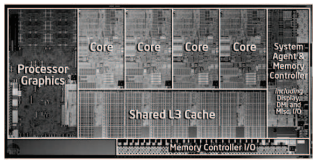
rem. Ne jamais utiliser ~~gets(...)~~
=> Non sécurisé!

Fichiers vitesse

	Transfert (Mo/s)	Tps Accès (ns)	Capacité (Mo)	Prix $/Go
L1	60 000	0.5	0.032	on chip
L2	25 000	7	0.256	
L3	10 000	20	12	20 000 000
RAM	5 000	60	6000	20
Disque dur	150	10 000 000	1000000	0.1



vue schématique
Processeur récent



http://www.ni.com

Fichiers vitesse

	Transfert (Mo/s)	Tps Accès (ns)	Capacité (Mo)	Prix $/Go
L1	60 000	0.5	0.032	on chip
L2	25 000	7	0.256	
L3	10 000	20	12	20 000 000
RAM	5 000	60	6000	20
Disque dur	150	10 000 000	1000000	0.1

vue schématique
Processeur récent

=> Ne pas abuser de lecture/ecriture nombreuses opérations

↳ Passer par un buffer, puis lire/ecrire par blocs dans le fichier

=> Ne pas lire caractères par caractères dans un fichier

↳ chargez l'ensemble dans un buffer
puis lire dans le buffer

=> Ouvrir/fermer un même fichier trop souvent

↳ Laisser le fichier ouvert tant que la fin de l'écriture n'a pas eu lieu

=> Ne pas utiliser de fichier pour des opérations critiques en temps

Chaine de compilation

Compilateur
Warnings
Compilation séparée
Précompilateur
Makefile
Edition de liens et librairies

Chaine de compilation

```
int main()
{
    int a=3;
    a++;
    int b=a+2;

    char texte[]="j aime l informatique";

    return 0;
}
```

Code (C)
(=langage humain)

Compilateur

Fichier executable
(=binaire)

(Windows: .exe
Linux: nom quelconque)

Compilateur

gcc

GNU Compiler Collection

http://gcc.gnu.org/
(C, C++,Obj-C,Fortran,Ada,Go)

Le + répendu sous Linux

D'autres existent:

Nwcc
Clang
Quick C (Microsoft)
Turbo C (Embarcadero)
XL C (IBM)

Note:
Compilateur C++ compatibles

BorlandC++
Intel C++ Compiler
Visual Studio
Turbo C++
ProDev
Solaris Studio

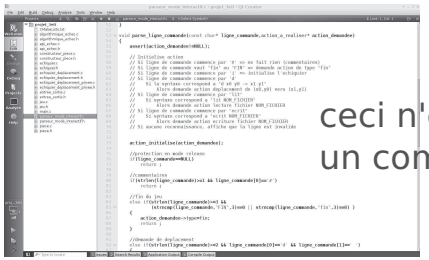
Compilateur

Remarque:

Ne pas confondre **IDE** et **Compilateur**

éditer du texte/code

Emacs
QtCreator
MS Visual C++
Dev C++
Eclipse
Code::Blocks
Anjuta
KDevelop



ceci n'est pas
un compilateur!

génère le binaire

Compilateur

En pratique

mon_fichier.c

mon_executable

```
Terminal - damien@garonne: ~
File  Edit  View  Terminal  Go  Help
[damien@garonne ~]$ gcc mon_fichier.c -o mon_executable
```

Puis
$ ./mon_executable

Compilateur: Paramètres de GCC

gcc possède des paramètres:

- Warnings
- Debug
- Optimisation

beaucoup d'options!

- Chemins d'acces
- Lien avec librairies
- Constantes
- ...

http://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html

Chaine de compilation

Compilateur

Ex: Warning

```c
#include <stdio.h>

int main()
{
    float *a;
    printf("%f",*a);

    return 0;
}
```

sans option de Warning

```
[damien@garonne ~/work/2012_2013_teaching/2012_3eti_projet_c/cours/src/code]$ gcc compilation.c
[damien@garonne ~/work/2012_2013_teaching/2012_3eti_projet_c/cours/src/code]$ gcc compilation.c -Wuninitialized
compilation.c: In function 'main':
compilation.c:7:17: warning: 'a' is used uninitialized in this function [-Wuninitialized]
```

avec option de Warning

en francais:
compilation.c, ligne 17. Warning 'a' est utilisé sans être initialisé dans cette fonction.

Warning = Aide pour le programmeur
= Lisible pour le programmeur
= Evite les erreurs "*silentieuses*"

Warnings typiques:

```c
int oublie_retour(int valeur)
{
    int valeur_a_retourner=0;
    valeur_a_retourner=valeur+1;
}

int* recupere_adresse_temporaire(int valeur)
{
    return &valeur;
}

int main()
{
    int a=oublie_retour(3);
    printf("%d\n",a);

    int tableau[2]={1,2,3};

    int b=3;
    int *p=recupere_adresse_temporaire(b);
    printf("%d\n",*p);

    int somme=0;
    unsigned int k=0;
    for(k=5;k>=0;--k)
        somme += 3;

    return 0;
}
```

Sans warnings

$ gcc mon_fichier.c

→ compile

Fonctionne?

Warnings typiques:

```c
int oublie_retour(int valeur)
{
    int valeur_a_retourner=0;
    valeur_a_retourner=valeur+1;
}

int* recupere_adresse_temporaire(int valeur)
{
    return &valeur;
}

int main()
{
    int a=oublie_retour(3);
    printf("%d\n",a);

    int tableau[2]={1,2,3};

    int b=3;
    int *p=recupere_adresse_temporaire(b);
    printf("%d\n",*p);

    int somme=0;
    unsigned int k=0;
    for(k=5;k>=0;--k)
        somme += 3;

    return 0;
}
```

pas de retour

adresse temporaire

depassement tableau

toujours vrai: boucle infinie

Ajout de warnings

```
$ gcc mon_fichier.c -Wall -Wextra
mon_fichier.c: In function 'oublie_retour':
mon_fichier.c:6:9: warning: variable 'valeur_a_retourner' set but not used [-Wunused-but-set-variable]
mon_fichier.c: In function 'recupere_adresse_temporaire':
mon_fichier.c:12:5: warning: function returns address of local variable [enabled by default]
mon_fichier.c: In function 'main':
mon_fichier.c:20:5: warning: excess elements in array initializer [enabled by default]
mon_fichier.c:20:5: warning: (near initialization for 'tableau') [enabled by default]
mon_fichier.c:20:9: warning: unused variable 'tableau' [-Wunused-variable]
mon_fichier.c:28:5: warning: comparison of unsigned expression >= 0 is always true [-Wtype-limits]
mon_fichier.c: In function 'oublie_retour':
mon_fichier.c:8:1: warning: control reaches end of non-void function [-Wreturn-type]
```


Warnings : Bonnes pratiques

Toujours activer un maximum de Warnings

conteneurs de multiples warnings

-Wall -Wextra Indispensable!!

=> Toujours compiler avec *$gcc -Wall -Wextra*

D'autres Warnings très utiles:

(non compris dans -Wall ni -Wextra)

-Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum
-Wwrite-strings -Wpointer-arith -Wcast-qual -Wredundant-decls
-Winit-self

Warnings : Bonnes pratiques

Toujours activer un maximum de Warnings

Ne **jamais** se priver du travail du compilateur!

Si après analyse des Warnings inutiles gachent la **lisibilité**
(ex. unused variables)

annule ce Warning spécifique

Option: -Wno-<nom_warning>

ex. -Wall -Wextra -Wno-unused-variable

Warnings : Bonnes pratiques

gcc -Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwrite-strings
-Wpointer-arith -Wcast-qual -Wredundant-decls -Winit-self

Trop long à écrire ?

1 unique fois dans un fichier
utilisez les scripts

script = lignes de commandes écrites dans un fichier
executées les unes derrières les autres

ex.

<nom_script>.sh

```
#!/bin/bash

<mes_lignes_de_scripts>
<...>
```

Script pour compiler

En pratique:

1. créez le fichier: mon_script.sh dans le même repertoire que: *mon_programme.c*

2. y inscrire:

```
#!/bin/bash

gcc mon_fichier.c -Wall -Wextra -o mon_executable
```

3. rendez le fichier executable: $ chmod +x mon_script.sh

4. lancez le: $ ./mon_script.sh

Script pour compiler

En pratique:

Avec l'ensemble des paramètre de compilation:

```
#!/bin/bash

CFLAGS="-Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwr
ite-strings"

gcc mon_fichier.c -o mon_executable ${CFLAGS}
```

déclaration d'une variable (+ lisible)

utilisation d'une variable

Note: En anglais: options de compilation = *flags*
options de compilation du compilateur C = *Cflags*

version générique:

```
#!/bin/bash

CFLAGS="-Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwr
ite-strings"

nom_de_fichier=mon_fichier.c

echo "Je compile" ${nom_de_fichier}
gcc ${nom_de_fichier} -o mon_executable ${CFLAGS}
```


Options de debug

`-g` **Toujours** compiler avec l'option de Debug lors du developpement

Permet l'utilisation des debuggers (gdb,kdbg,ddd,valgrind, ...)

Pour résumer, CFLAGS minimales à toujours utiliser:

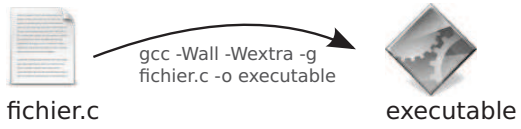
`-Wall -Wextra -g`

Chaine de compilation

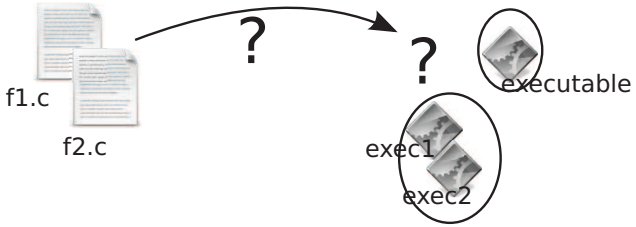
Compilateur
Warnings
→ **Compilation séparée**
Précompilateur
Makefile
Edition de liens et librairies

Compilation séparée

1 Fichier:



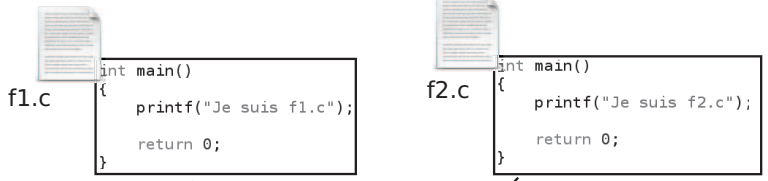
2 Fichiers:



Cas de 2 fichiers

Un executable => 1 point d'entrée (int main())

Cas 1:



$ gcc -Wall -Wextra -g f1.c -o executable_1
$ gcc -Wall -Wextra -g f2.c -o executable_2

2 créations d'executables distincts


executable_1

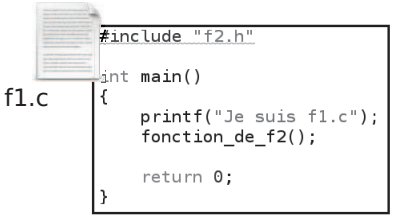
$ ./executable_1
> Je suis f1.c

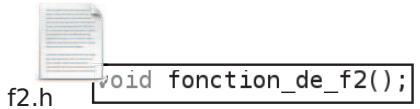

executable_2

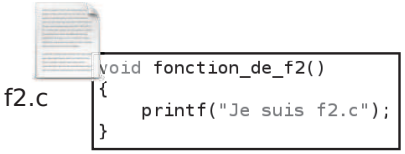
$ ./executable_2
> Je suis f2.c

Cas de 2 fichiers

Cas 2:





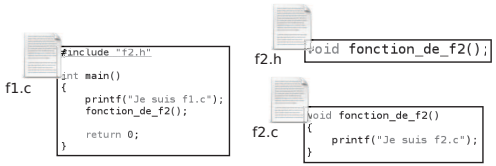


f2 ne contient pas de fonction: *int main()*
=> pas d'executable associé à f2 uniquement!

f2 agit en tant que "*librairie*" pour le code appelé dans f1

Cas de 2 fichiers: compilation séparée

Cas 2:

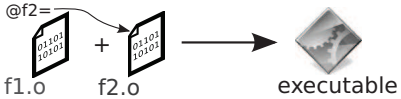


Principe:

- On compile tout ce qu'on peut de f1
  => cad: tout sauf la fonction de f2 (adresse à définir)

- On compile tout ce qu'on peut de f2
  => cad: tout

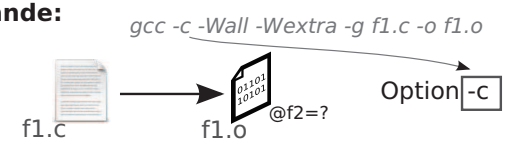
- On rassemble les 2 binaires + complète l'adresse de fonction définitive



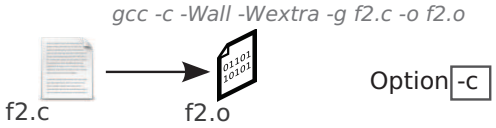
Cas de 2 fichiers: compilation séparée

Cas 2, ligne de commande:

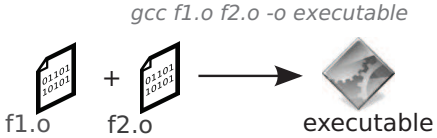
- Compilation de f1.c



- Compilation de f2.c



- Edition de liens, génération d'executable

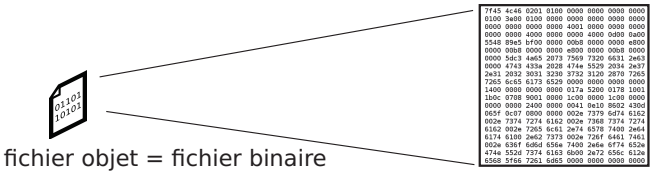


Fichier objet

gcc -c -Wall -Wextra -g f1.c -o f1.o

Option -c

indique:
génère un binaire du code
ne génère pas d'executable
s'arrête avant l'étape d'édition de lien



fichier non executable

peut contenir des liens vers des fonctions externes

Fichier objet

Pour visualiser un fichier binaire

ex.

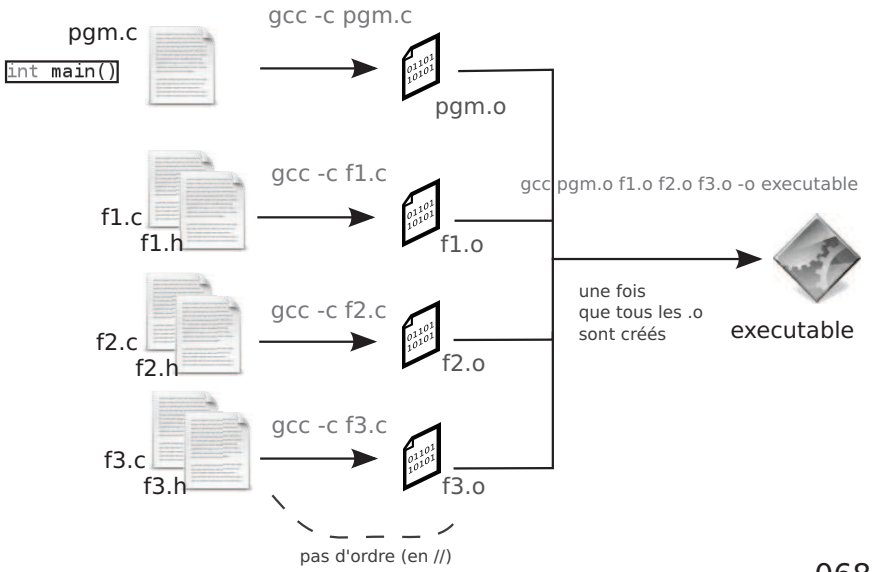
Ouvrir le fichier avec xemacs ou emacs
$ xemacs fichier.o

Appuyez sur **Alt+X**: cherchez *hexl-mode*

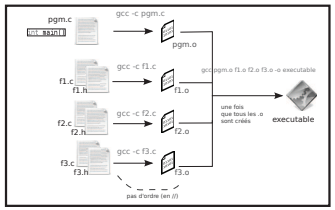
ou

```
$ od -x fichier.o
```


Plusieurs sources, un executable



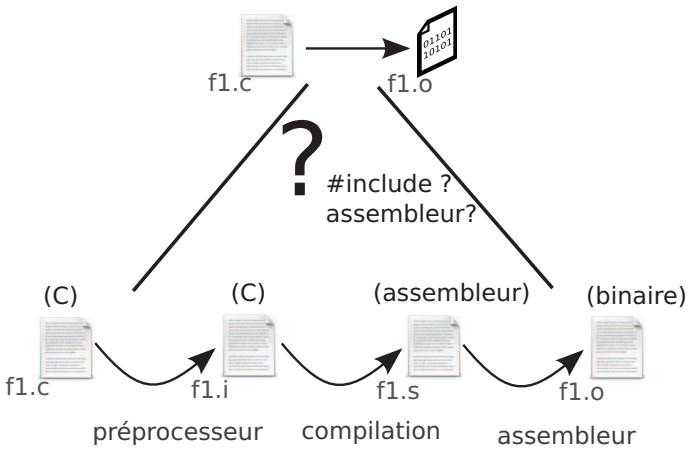
Plusieurs sources, un executable: script



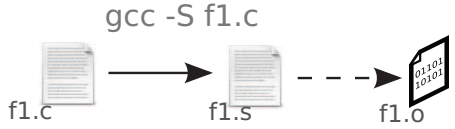
mon_script_de_compilation.sh

```
#!/bin/bash

CFLAGS="-Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwr
ite-strings -Wpointer-arith -Wcast-qual -Wredundant-decls -Winit-self -g"

#Compilation vers fichiers objets
gcc f1.c -o f1.o ${CFLAGS}
gcc f2.c -o f2.o ${CFLAGS}
gcc f3.c -o f3.o ${CFLAGS}
gcc pgm.c -o pgm.o ${CFLAGS}

#Edition de liens
gcc f1.o f2.o f3.o pgm.o -o executable
```


Construction d'un fichier objet



Fichier assembleur



Option -S

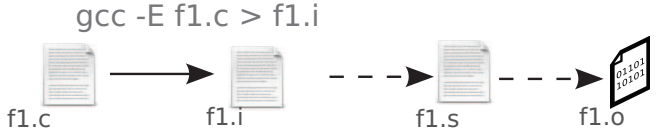
```
int main()
{
    int a=5;
    int b=6;
    int c=a+b;
}
```

```
        .file   "compilation.c"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $5, -4(%rbp)
        movl    $6, -8(%rbp)
        movl    -8(%rbp), %eax
        movl    -4(%rbp), %edx
        addl    %edx, %eax
        movl    %eax, -12(%rbp)
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 4.7.1 20120721 (prerelease)"
        .section        .note.GNU-stack,"",@progbits
```


Chaine de compilation

- Compilateur
- Warnings
- Compilation séparée
- → **Précompilateur**
- Makefile
- Edition de liens et librairies

Precompilateur



Option -E

Precompilateur =
Convertion du code C en un autre code C (plus simple à compiler)

Precompilateur

Elimine tous les commentaires
=> inutile pour générer de l'assembleur

f1.c

```
/**
   Ceci est un commentaire qui devrait
                décrire mon programme.
   J'aime l'informatique.
   J'aime le C.
*/
int main()
{
    // ceci est un commentaire
    int a=5;
    /** Ceci est un autre commentaire */
    int b=6;
    int c=a+b;
}
```

gcc -E f1.c > f1.i

f1.i

```
# 1 "compilation.c"
# 1 "<command-line>"
# 1 "compilation.c"

int main()
{

    int a=5;

    int b=6;
    int c=a+b;
}
```

Conclusion:
Ecrivez vos commentaires pour les humains, pas pour la machine!

Precompilateur

Remplace toute les *Macros*
#define

f1.c

```
#define MA_CONSTANTE 8
#define PI 3.14159


int main()
{
    int a=MA_CONSTANTE;

    int b=MA_CONSTANTE+8;

    int tableau[MA_CONSTANTE];

    float rayon=3.0;
    float aire=2*PI*RAYON;
}
```

gcc -E f1.c > f1.i

f1.i

```
# 1 "compilation.c"
# 1 "<command-line>"
# 1 "compilation.c"

int main()
{
    int a=8;

    int b=8 +8;

    int tableau[8];

    float rayon=3.0;
    float aire=2*3.14159*RAYON;
}
```

Conclusion:
Centralisez vos constantes (tailles tableaux) dans des macros.

Precompilateur

Remplace toute les *Macros*
#define

Attention: Les macros sont directement copiés collés!
=> Pas de points-virgules (;) !

f1.c

```
#define MA_CONSTANTE 8;
#define PI 3.14159;


int main()
{
    int a=MA_CONSTANTE;

    int b=MA_CONSTANTE+8;

    int tableau[MA_CONSTANTE];

    float rayon=3.0;
    float aire=2*PI*RAYON;
}
```

gcc -E f1.c > f1.i

f1.i

```
# 1 "compilation.c"
# 1 "<command-line>"
# 1 "compilation.c"


int main()
{
    int a=8;;

    int b=8;+8;

    int tableau[8;];

    float rayon=3.0;
    float aire=2*3.14159;*RAYON;
}
```

Conclusion:
Pas de points-virgules dans les macros!.

Precompilateur

Remplace toute les *Macros*
#define

Les macros peuvent être des fonctions!

f1.c

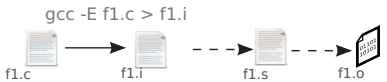
```
#define CARRE(x) x*x
#define NORME(x,y) sqrt(CARRE(x)+CARRE(y))

int main()
{

  float a=5;
  float b=CARRE(4);

  float d=NORME(a,b);
  float e=NORME(8,1);

}
```

gcc -E f1.c > f1.i

f1.i

```
# 3 "compilation.c" 2

int main()
{

  float a=5;
  float b=4*4;

  float d=sqrt(a*a+b*b);
  float e=sqrt(8*8 +1*1);

}
```


Precompilateur



Remplace toute les *Macros*
#define

Les macros peuvent être des fonctions!
Attention au principe du copié-collé des macros !!!

f1.c

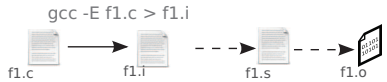
```
#define CARRE(x) x*x
#define NORME(x,y) sqrt(CARRE(x)+CARRE(y))

int main()
{

  float a=5;
  float b=CARRE(4+5);

  float d=NORME(a+b,b);
  float e=NORME(8+1,1-4);

}
```

gcc -E f1.c > f1.i

f1.i

```
int main()
{

  float a=5;
  float b=4+5*4+5;

  float d=sqrt(a+b*a+b+b*b);
  float e=sqrt(8+1*8+1 +1-4*1-4);

}
```


Precompilateur



Remplace toute les *Macros*
#define

Les macros peuvent être des fonctions!
Attention au principe du copié-collé des macros !!!

f1.c

```
#define CARRE(x) ((x)*(x))
#define NORME(x,y) (sqrt(CARRE(x)+CARRE(y)))

int main()
{

  float a=5;
  float b=CARRE(4+5);

  float d=NORME(a+b,b);
  float e=NORME(8+1,1-4);

}
```

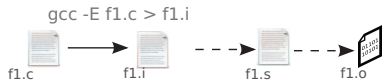
gcc -E f1.c > f1.i

f1.i

```
int main()
{

  float a=5;
  float b=((4+5)*(4+5));

  float d=(sqrt(((a+b)*(a+b))+((b)*(b))));
  float e=(sqrt(((8+1)*(8+1))+((1-4)*(1-4))));

}
```

=> Autour de chaque variable/expression: ajoutez des parenthèses

Precompilateur



Utilisation des conditionnelles+macro:

Pour la portabilitée entre systèmes

Pour le debug


```
#if SYSTEM=LINUX
#include <stdio.h>
#else if SYSTEM==WINDOWS
#include <stdafx.h>
#endif

#if ARCHI==x86
...
#else if ARCHI==itanium
...
#else if ARCHI==power_pc
...
#endif

#if NBITS==64
...
#else if NBITS==32
...
#endif
```

Attention: Ne **pas** utiliser pour l'optimisation

Réduit la lisibilité du code.

Si faisable avec une fonction: **utiliser une fonction**

Precompilateur

Macro: #include "fichier"

Copie-colle le contenu d'un fichier

mon_fichier

```
void ma_fonction(int a)
{
  printf("bonjour %d\n",a);
}

int ma_variable=36;
```

f1.c

```
#include "mon_fichier"

int main()
{
    ma_fonction(ma_variable);
    return 0;
}
```

f1.i

```
# 1 "compilation.c"
# 1 "<command-line>"
# 1 "compilation.c"

# 1 "mon_fichier" 1
void ma_fonction(int a)
{
  printf("bonjour %d\n",a);
}

int ma_variable=36;
# 3 "compilation.c" 2

int main()
{
    ma_fonction(ma_variable);
    return 0;
}
```

execution:

bonjour 36

gcc -E f1.c > f1.i

Precompilateur

Synthèse précompilateur

Potentiellement puissant (va au dela du langage C)

A utiliser avec précaution

=> rend rapidement le code peu standard
= peu lisible

Bonne pratique:
N'utilisez que les règles standards:

* include guard
* #ifdef => portabilité, debug.
* pas d'optimisation, préférez les fonctions aux macros