Mise à niveau en C

INFRES 1A

Enseignant: P. Bertin-Johannet

Introduction

Le langage C

Le langage C est utilisé lorsqu'on cherche à minimiser:

- La vitesse d'execution
- L'utilisation mémoire
- Le contrôle des instructions éxécutées

Il est courament utilisé dans:

- Les systèmes d'exploitation
- Les drivers
- Les navigateurs
- Les microcontroleurs
- Les jeux vidéos

Figure 1: Exemples d'utilisation du C

Pourquoi apprendre la programmation en C

- Le C est utilisé dans pratiquement tous les composants informatiques

Pourquoi apprendre la programmation en C

- Le C est utilisé dans pratiquement tous les composants informatiques

- La plupart des langages de programation sont interprétés par un programme écrit en C (Python, Java, C#, bash, etc...)

Pourquoi apprendre la programmation en C

- Le C est utilisé dans pratiquement tous les composants informatiques

- La plupart des langages de programation sont interprétés par un programme écrit en C (Python, Java, C#, bash, etc...)

- Une grande partie des failles de sécurité critiques originent de programmes écrits en C (près de 70% d'après une étude menée par Google)

Pourquoi apprendre la programmation en C

- Le C est utilisé dans pratiquement tous les composants informatiques

- La plupart des langages de programation sont interprétés par un programme écrit en C (Python, Java, C#, bash, etc...)

- Une grande partie des failles de sécurité critiques originent de programmes écrits en C (près de 70% d'après une étude menée par Google)

- Un programme en C est plus rapide et utilise moins de mémoire en général qu'un programme ecrit dans un autre langage

Pourquoi apprendre la programmation en C

- Le C est utilisé dans pratiquement tous les composants informatiques

- La plupart des langages de programation sont interprétés par un programme écrit en C (Python, Java, C#, bash, etc...)

- Une grande partie des failles de sécurité critiques originent de programmes écrits en C (près de 70% d'après une étude menée par Google)

- Un programme en C est plus rapide et utilise moins de mémoire en général qu'un programme ecrit dans un autre langage

- Une majeure partie des langages utilisés aujourd'hui sont inspirés du C

Comparaison d'utilisation des ressources

On considère ici un programme qui demande un nombre à l'utilisateur et affiche deux de ses facteurs

Langage	Temps d'execution	Nombre d'appels système	Mémoire utilisée	Mémoire disque nécessaire
C	8 ms	52	2.5 Mo	24 Ko
Java	86 ms	185	623 Mo	124 000 Ko
Python	1 400 ms	670	13 Mo	6 000 Ko

Particularités de la programmation en C

- On contrôle précisément la mémoire que l'on souhaite utiliser ainsi que les instructions executées
- Il faut donc souvent se poser les questions suivantes :
  - Quelles instructions binaires sont executées et quel est leur cout
  - Quelle quantité de mémoire est utilisée
  - Quels appels systèmes sont effectués
  - Comment sont organisées les données dans la mémoire

Cela demande donc un effort supplémentaire

Les bases du C

Compilation

- Le langage C peut être traduit en langage binaire pour être executé
- On appelle ce procédé **Compilation**. Le programme effectuant la traduction est appelé **Compilateur**

```c
#include "stdlib.h"

int main(){
  int a = 5;

  if (a * 4 > a * a){
    printf("hello world !");
  }
}
```

Programme en C

Compilateur

```
01101100
01101111
01110110
01100101
```

Fichier binaire

Compilation

- Le compilateur que nous utiliserons dans ce cours s'appelle *gcc*
- On lui passe le nom du fichier que l'on désire compiler puis le nom du fichier binaire à créer

gcc main.c -o main.out

Fichier .c à compiler

Fichier binaire à créer

Les variables

- En C, une variable est définie par :
  - Un nom
  - Un type
- Elle permet d'associer un nom à un emplacement mémoire.
- Pendant l'execution du programme, on pourra modifier la valeur située dans cet emplacement mémoire
- Exemple: une variable **a** de type entier qui contiendra d'abord 5 puis 4

```c
int a = 5;
a = a - 1;
```

Types de données

- En C nous sommes obligés de préciser le type de chaque variable utilisée par notre programme
- Le type d'une variable indique :
  - La quantité de mémoire qui lui est reservée
  - Les opérations qu'on peut lui appliquer

Les types de base

Durant cette séance nous utiliserons les types suivants:

Nom	Représentation mémoire	Mémoire utilisée	Intervale de valeurs
Int	Nombre entier	minimum 2 octets	si 2 octets: [−65536 ; 65536]
Char	Nombre entier	1 octet (8bits)	0 ; 255
Float	Réel (Nombre à virgule)	2 octets (16 bits)	$3.4 * 10^{-38}$; $3.4 * 10^{38}$

Déclaration de variable

- Afin d'utiliser une variable il faut toujours la déclarer et lui donner un type, on écrit :

  ```
  type nom_variable;
  ```

- Pour une variable apellée b et de type entier on écrira :

  ```
  int b ;
  ```

- Pour une variable apellée c et de type charactère on écrira :

  ```
  char c ;
  ```

Déclaration de variable

- On peut aussi donner une valeur à notre variable au moment de la déclaration :
- `type nom_de_la_variable = valeur;`

Par exemple :

- `ip ip_client = "192.168.1.1;"`

- `int age = 52;`

Exemple

Programme:

```
int a = 5;
int b;
char c = 10;
float f = -1.8;
b = a + c;
```

Mémoire:

Valeurs	Octets
	Octet 1
	Octet 2
	Octet 3
	Octet 4
	Octet 5
	Octet 6
	Octet 7
	Octet 8
	Octet 9

Exemple

Programme:

```
int a = 5; <
int b;
char c = 10;
float f = -1.8;
b = a + c;
```

Mémoire:

Valeurs	Octets
5	Octet 1
	Octet 2
	Octet 3
	Octet 4
	Octet 5
	Octet 6
	Octet 7
	Octet 8
	Octet 9

Exemple

Programme:

```
int a = 5;
int b; <
char c = 10;
float f = -1.8;
b = a + c;
```

Mémoire:

Valeurs	Octets
5	Octet 1
	Octet 2
??	Octet 3
	Octet 4
	Octet 5
	Octet 6
	Octet 7
	Octet 8
	Octet 9

Exemple

Programme:

```
int a = 5;
int b;
char c = 10;
float f = -1.8;
b = a + c;
```

Mémoire:

Valeurs	Octets
5	Octet 1
	Octet 2
??	Octet 3
	Octet 4
10	Octet 5
	Octet 6
	Octet 7
	Octet 8
	Octet 9

Exemple

Programme:

```
int a = 5;
int b;
char c = 10;
float f = -1.8;
b = a + c;
```

Mémoire:

Valeurs	Octets
5	Octet 1
	Octet 2
??	Octet 3
	Octet 4
10	Octet 5
−1.8	Octet 6
	Octet 7
	Octet 8
	Octet 9

Exemple

Programme:

```
int a = 5;
int b;
char c = 10;
float f = -1.8;

b = a + c; <
```

Mémoire:

Valeurs	Octets
5	Octet 1
	Octet 2
15	Octet 3
	Octet 4
10	Octet 5
−1.8	Octet 6
	Octet 7
	Octet 8
	Octet 9

Les charactères

- Le type char permet aussi de représenter des charactères (lettre/symbole).
- On l'écrit donc entre deux apostrophes:

Par exemple :

- `char a = 'i'`
- `char b = '$'`
- `char c = '*'`

La table ASCII (American Standard Code for Information Interchange)

- La table ASCII associe un nombre à tous les caractères usuels (états-uniens) sur 7 bits (128 positions)
- Lorsqu'on écrit un charactères dans un programme C, le nombre correspondant dans la table ascii sera utilisé

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
32	20	[SPACE]	64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(	72	48	H	104	68	h
41	29	)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	-	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p
49	31	1	81	51	Q	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s

Exemple

Programme:

```
char a = 56;
char b = '8';
char c = 'a' + 4;
char d = 'e';
```

Mémoire:

Valeurs	Octets
	Octet 1
	Octet 2
	Octet 3
	Octet 4
	Octet 5
	Octet 6

Exemple

Programme:

```
char a = 56; <
char b = '8';
char c = 'a' + 4;
char d = 'e';
```

Mémoire:

Valeurs	Octets
56	Octet 1
	Octet 2
	Octet 3
	Octet 4
	Octet 5
	Octet 6

Exemple

Programme:

```
char a = 56;
char b = '8';
char c = 'a' + 4;
char d = 'e';
```

Mémoire:

Valeurs	Octets
56	Octet 1
56	Octet 2
	Octet 3
	Octet 4
	Octet 5
	Octet 6

Exemple

Programme:

```
char a = 56;
char b = '8';
char c = 'a' + 4; <
char d = 'e';
```

Mémoire:

Valeurs	Octets
56	Octet 1
56	Octet 2
101	Octet 3
	Octet 4
	Octet 5
	Octet 6

Exemple

Programme:

```
char a = 56;
char b = '8';
char c = 'a' + 4;
char d = 'e';
```

Mémoire:

Valeurs	Octets
56	Octet 1
56	Octet 2
101	Octet 3
101	Octet 4
	Octet 5
	Octet 6

Opérations simples sur les entiers

Le C propose les opérateurs suivants sur les entiers :

- Les opérations classiques : (*, +, /, -)
  - $1 + 2 * 3 - 4 = 3$
- Le reste de la division euclidienne (ou “modulo”) : %
  - $18 \% 14 = 4$

Opérations binaires sur les entiers

Le C propose aussi:

- des opérateurs binaires qui appliquent une opération booléenne à chaque bit des deux entiers donnés (|, &, ^)
  - $1 \mid 2 = 0$
  - 31 & 16 = 16
  - 5 ^ 30 = 25
- Des opérateurs qui déplacent les bits d'une variable dans un sens (>>, <<)
  - $3 \ll 2 = 12$
  - $40 \gg 3 = 5$
- Le complément à 1(négation bit à bit) ou "tilde"
  - $\sim 7 = 248$ (sur 8 bits)

Écrire dans le terminal

- Pour afficher dans le terminal on utilise la fonction printf (pour print format)
- La syntaxe est la suivante : `printf(format, variables) ;`
- Exemple: afficher un jour et une température

Programme:

```
int jour = 27;
float temperature = 25.4;
printf("il fait %f le %d\n", temperature, jour);
```

Sortie:

> il fait 25.4 le 27

Écrire dans le terminal

Durant cette séance, nous utiliserons les formats suivants :

- `%d` : `int`
- `%f` : `float`
- `%c` : `char` (affiche le charactère dans la table ASCII)

> ⚠ **Attention**
>
> Sur la plupart des environnements, le texte ne sera pas affiché tant que vous n'aurez pas envoyé un saut de ligne : `"\n"`

Exemple

Programme:

```
char a = 56;
float b = 8.1;
int c = 4;
printf("%c -- %f -- %d\n", a, b, c);
```

Exemple

Programme:

```
char a = 56;
float b = 8.1;
int c = 4;
printf("%c -- %f -- %d\n", a, b, c);
```

Sortie:

```
> 8 -- 8.1 -- 4
```

Lire depuis le terminal

- Pour lire dans le terminal on utilise la fonction `scanf (pour scan format)`
- La syntaxe est la suivante : `scanf(format, &variable) ;`
- Exemple: lire un age et une taille

```
int age;
float taille;
scanf("%d %f", &age, &taille);
```

Formats de scanf

Les formats de scanf sont les mêmes que printf

- `%d` : `int`
- `%f` : `float`
- `%c` : `char` (scanne le charactère dans la table ASCII)

> ⚠️ **Attention**
>
> un espace dans scanf permet de sauter tous les charactères “blancs” (tabs, espaces, sauts de lignes) jusqu’au prochain charactère non “blanc”

La condition if

- L'instruction `if` (en anglais : "si") permet d'executer du code uniquement si une condition est vraie.

```
if(condition){
        code
}
```

- Les lignes de code entre les accolades ne seront executées que si la condition est vraie

Exemple

```c
int temperature;
printf("Quelle est la température dehors ?\n");
scanf("%d", &temperature);

if (temperature < 7){
     printf("Il vaut mieux se couvrir !\n");
}
```

La condition else

L'instruction `else` ("sinon" en anglais) placée après un `if` permet d'executer du code si la condition était fausse.

```
if(condition){
        code_a
}
else{
        code_b
}
```

Le code code_b sera executé uniquement si la condition est vraie

Exemple

```
if (age > 17){
      printf("Tarif : 8€ \n");
}
else {
      printf("Tarif : 6€ \n");
}
```

Les conditions

- En C, une condition est un entier. On la considère comme vraie ou fausse ainsi :
  - 0 => FAUX
  - N'importe quoi d'autre => VRAI
- Les opérations suivantes renvoient 1 ou 0 pour vrai ou faux :
  - Comparaison de nombres : >, <, ==, >=, <=
  - Algebre de bool : || (ou), && (et), ! (not)

Les conditions

Les opérateurs d'affectation permettent de changer la valeur d'une variable et de renvoyer le résultat

- `c = b` => affecte la valeur de b à c et renvoie la nouvelle valeur
- `c += b` => ajoute b à c (existe aussi pour *, /, -)
- `c++` => ajoute 1 à c et renvoie l'ancienne valeur
- `c--` => ajoute 1 à c et renvoie l'ancienne valeur
- `++c` => ajoute 1 à c et renvoie la nouvelle valeur
- `--c` => enlève 1 à c et renvoie la nouvelle valeur

Exemple

Programme:

```
char a = 5;
char b = (a = 2);
b *= 2;
a = --b + 4;
char c = (b = a) + 1
```

Mémoire:

Valeurs	Octets
	Octet 1
	Octet 2
	Octet 3
	Octet 4
	Octet 5

Exemple

Programme:

```
char a = 5; <
char b = (a = 2);
b *= 2;
a = --b + 4;
char c = (b = a) + 1
```

Mémoire:

Valeurs	Octets
5	Octet 1
	Octet 2
	Octet 3
	Octet 4
	Octet 5

Exemple

Programme:

```
char a = 5;
char b = (a = 2);
b *= 2;
a = --b + 4;
char c = (b = a) + 1
```

Mémoire:

Valeurs	Octets
2	Octet 1
2	Octet 2
	Octet 3
	Octet 4
	Octet 5

Exemple

Programme:

```
char a = 5;
char b = (a = 2);
b *= 2;
a = --b + 4;
char c = (b = a) + 1
```

Mémoire:

Valeurs	Octets
2	Octet 1
4	Octet 2
	Octet 3
	Octet 4
	Octet 5

Exemple

Programme:

```
char a = 5;
char b = (a = 2);
b *= 2;
a = --b + 4; <
char c = (b = a) + 1
```

Mémoire:

Valeurs	Octets
7	Octet 1
3	Octet 2
	Octet 3
	Octet 4
	Octet 5

Exemple

Programme:

```
char a = 5;
char b = (a = 2);
b *= 2;
a = --b + 4;
char c = (b = a) + 1
```

<

Mémoire:

Valeurs	Octets
7	Octet 1
7	Octet 2
8	Octet 3
	Octet 4
	Octet 5

Priorité des opérateurs

- Les opérateurs avec la priorité la plus élevée sont évalués en premier
- Par exemple dans `2 + 3 * 5` on évalue la multiplication en premier : `2 + (3 * 5)`
- Tous les opérateurs C ont une priorité associée

Priorité des opérateurs

Du plus prioritaire au moins prioritaire:

Parenthèses	()
Opérateurs unaires	++ − -
Multiplication et division	/ * %
Addition et soustraction	+ -
Déplacement de bits	<< >>
Comparaison	> >= < <=
Égalité	!=
Opérateurs Binaires (chacun une priorité différente)	& puis ^ puis |
Logique booléenne	|| &&
Affectation	+= -= *= /=

Programme complet en C

- Dans cette séance les programmes seront de la forme suivante :

```
#include "stdio.h"
int main(){
      code
 }
```

Exemple

```c
#include "stdio.h"
int main(){
        printf("C'est la fin ! \n");
}
```

Mise en pratique

```
printf("

          _____
        < TP 1 >
          -----
                \   ^__^
                 \  (oo)\_______
                    (__)\       )\/\
                        ||----w |
                        ||     ||
        ")
```