Yantra Technologies

www.yantra-technologies.com

Le Langage C

carole.grondein@yantra-technologies.com
david.palermo@yantra-technologies.com

https://wiki.waze.com/wiki/Your_Rank_and_Points

Programmation en C : Sommaire

www.yantra-technologies.com

1 - Généralités

david.palermo@yantra-technologies.com

1 - Généralités

1 - Historique

2 - Les styles de programmation

3 - Avantages du langage C

4 - Inconvénients du langage C

5 - Environnement de développement

1a- Historique

- Le langage C a été créé au début des années 1970 par Dennis Ritchie (Laboratoires Bell AT&T). Son but était de ré-écrire le système d'exploitation Unix en langage evolué.
  En 1978, le premier standard est publié ("The C Programming Language") par B. W. Kernighan et D.M. Ritchie.

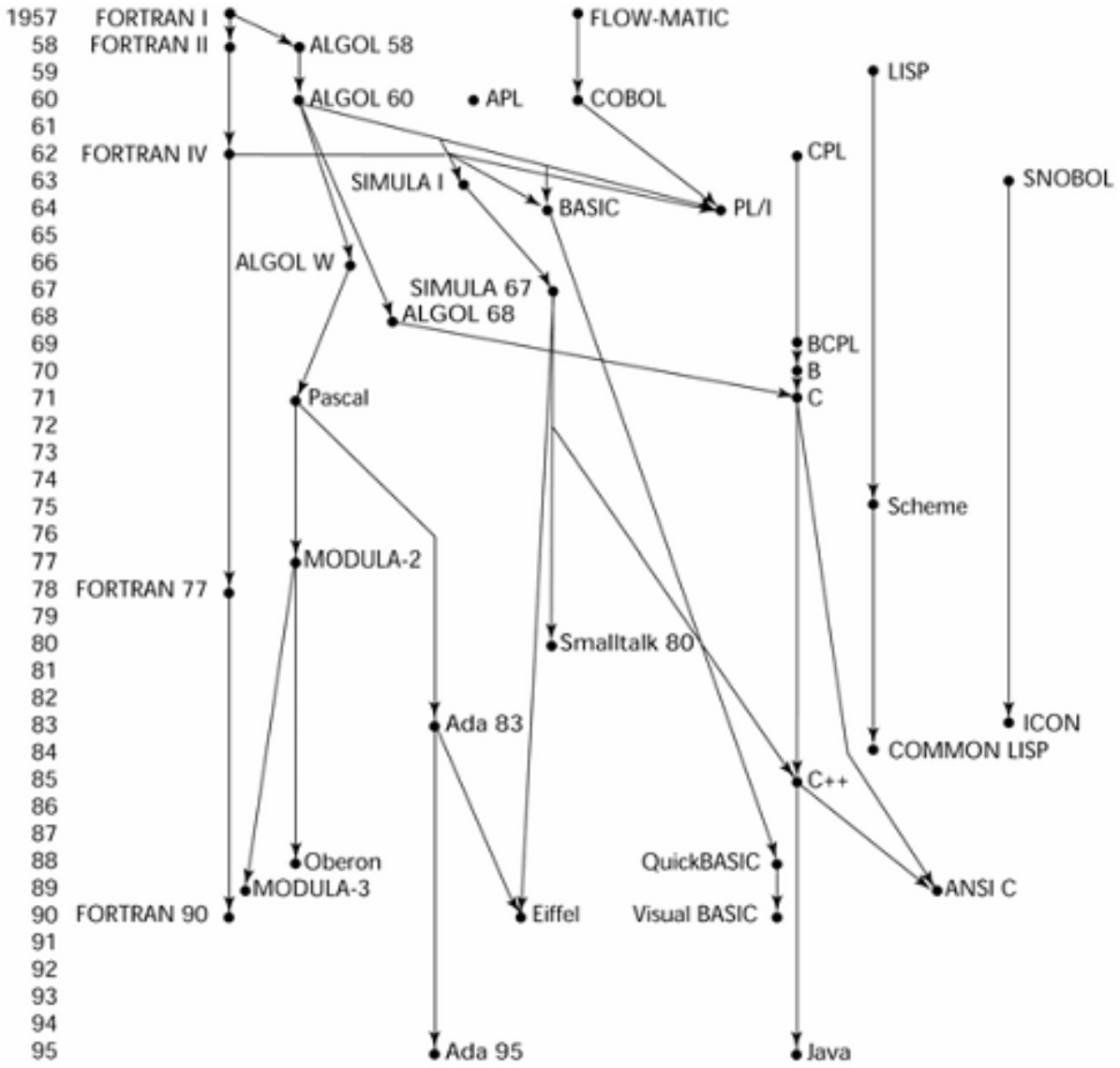
- En 1983, l‘ "American National Standards Institute" (ANSI) définit le standard ANSI-C qui est une définition explicite et indépendante de la machine pour le langage C".

- Le langage C++ est né en 1983 et a pour but de développer un langage qui garderait les avantages de ANSI-C (portabilité, efficacité) et qui permettrait en plus la programmation orientée objet. Premier standard ANSI-C++ en 1998 corrigée en 2003,

- Nouvelle version 12/08/2011 nouvelle norme du C++ C++11

.

1b - Historique



2 - Les styles de programmation

- **Fonctionnels : évaluation d 'expressions** : Lisp (1958), APL (récursivité) (1962), Caml (1985)

- **Procéduraux ou impératifs**: exécution d 'instructions

  Cobol (1959), *Fortran (1954), C (1970), Pascal (1970),*

- **Objet** : *Simula (1964-1967) , Smalltalk (1976), C++ (1980), Java (1995), ADA 95 (1990), C# (2001)*
  - ensemble de composants autonomes (objets) qui disposent de moyens d 'interaction,
  - utilisation de classes,
  - échange de message.

3 – Avantages du langage C

1. **Universel** : C n'est pas orienté vers un domaine d'applications spéciales
2. **Compact** : C est basé sur un noyau de fonctions et d'opérateurs limités qui permet la formulation d'expressions simples mais efficaces.
3. **Moderne** : C est un langage structuré, déclaratif et récursif. Il offre des structures de contrôle
4. **Près de la machine** : C offre des opérateurs qui sont très proches de ceux du langage machine (manipulations de bits, pointeurs…). C'est un atout essentiel pour la programmation des systèmes embarqués.
5. **Rapide** : C permet de développer des programmes concis et rapides.
6. **Indépendant de la machine** : C est un langage près de la machine (microprocesseur) mais il peut être utilisé sur n'importe quel système ayant un compilateur C. Au début C était surtout le langage des systèmes UNIX mais on le retrouve comme langage de développement aujourd'hui de tous les systèmes d'exploitation
7. **Portable** : En respectant le standard ANSI-C, il est possible d'utiliser (théoriquement ☺) le même programme sur tout autre système simplement en le recompilant. Il convient néanmoins de faire attention dans le cas d'un système embarqué qui n'inclut pas toujours un système d'exploitation…
8. **Extensible** : C peut être étendu et enrichi par l'utilisation de bibliothèque de fonctions achetées ou récupérées (logiciels libres…).

4 - Inconvénients du langage C

1. **Efficacité et compréhensibilité** : C autorise d'utiliser des expressions compactes et efficaces. Les programmes sources doivent rester compréhensibles pour nous-mêmes et pour les autres. ***Attention : La programmation efficace en langage C nécessite beaucoup d'expérience. Sans commentaires ou explications, les fichiers sources C peuvent devenir incompréhensibles et donc inutilisables (maintenance ?).***

2. **Portabilité et bibliothèques de fonctions :**
   - La portabilité est l'un des avantages les plus importants de C : en écrivant des programmes qui respectent le standard ANSI-C, nous pouvons les utiliser sur n'importe quelle machine possédant un compilateur ANSI-C.
   - Le nombre de fonctions standards ANSI-C (d'E/S) est limité. La portabilité est perdue si l'on utilise une fonction spécifique à la machine de développement.

3. **Discipline de programmation :**
   - C est un langage près de la machine donc dangereux bien que C soit un langage de programmation structuré.
   - C n'inclut pas de gestion automatique de la mémoire comme Java. ***La gestion mémoire en C est la cause de beaucoup de problèmes (***certains ont peut être eu une mauvaise expérience avec les *malloc()/free()*)…
   - L'instruction *goto* existe en C.

5a - Environnement de développement

Compilateurs gratuits :

Eclipse CDT : (http://www.eclipse.org/cdt/) fournit un environnement de développement moderne et modulaire pour la réalisation de projets C et C++.

Sous Windows :

Cygwin (http://www.cygwin.com/). Un environnement qui permet d'émuler Linux sous Windows, et donc en particulier de disposer du compilateur Gcc.

Code :: Blocks (http://www.codeblocks.org/) est un environnement de développement en C++ entièrement configurable et extensible à l'aide de nombreux plugins. A la fois complet et simple d'utilisation, il supporte divers compilateurs : GCC, Microsoft Visual C++ Toolkit 2003, Microsoft Visual C++ Express 2005, Borland C++ 5.5, Intel C++ compiler, etc.

Dev-C++ (http://www.bloodshed.net/devcpp.html). c'est un système complet de développement C/C++, tournant cette fois sous Windows directement. Comme Djgpp, il comprend un éditeur multi-fichiers, un compilateur, un gestionnaire de projet et un débogueur. Le compilateur de base est d'ailleurs le même : Gcc

MinGW (http://www.mingw.org/) C' est un compilateur C/C++ qui produit des exécutables Win32. Il constitue une alternative au Visual C++ de Microsoft. Le package comprend un pré-processeur, un compilateur, un linker de même qu'une très grande partie des fichiers en-têtes pour programmer sous Win32. MinGW s'exécute par la ligne de commande

Sous Unix : **Gcc/G++.** Les sources sont téléchargeables à partir de http://gcc.gnu.org/ avec un débogueur graphique (ddd) à l'adresse http://www.gnu.org/software/ddd/.

5b - Environnement de développement

Compilateurs C gratuits pour Windows

- MinGw (http://www.mingw.org/)
- Borland C++ Compiler
- Digital Mars

Environnements de développement intégrés C gratuits

- **Visual Studio 2015 :** https://msdn.microsoft.com/fr-fr/vstudio/
- CodeBlocks Studio **(http://www.codeblocks.org/)**
- NetBeans I.D.E.
- Anjuta
- KDevelop
- Glade

Compilateurs / EDI C commerciaux disponibles en évaluation

- Microsoft Visual Studio .NET

Divers

- Doxygen
- Valgrind
- Intel VTune Performance Analyzer
- Intel Parallel Studio
- TotalView Debugger

Voir : http://c.developpez.com/compilateurs/

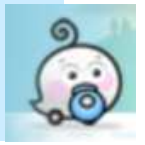
www.yantra-technologies.com

3 – Le Langage C

david.palermo@yantra-technologies.com

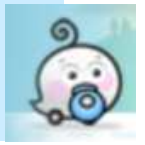
Sommaire

0 - Mots-clés du C++/C



alignas (depuis C++11)	char32_t (depuis C++11)	enum	namespace	return	try
alignof (depuis C++11)	class	explicit	new	short	typedef
and	compl	export	noexcept (depuis C++11)	signed	typeid
and_eq	const	extern	not	sizeof	typename
asm	constexpr (depuis C++11)	false	not_eq	static	union
auto (depuis C++11)	const_cast	float	nullptr (depuis C++11)	static_assert (depuis C++11)	unsigned
bitand	continue	for	operator	static_cast	using
bitor	decltype (depuis C++11)	friend	or	struct	virtual
bool	default	goto	or_eq	switch	void
break	delete	if	private	template	volatile
case	do	inline	protected	this	wchar_t
catch	double	int	public	thread_local (depuis C++11)	while
char	dynamic_cast	long	register	throw	xor
char16_t (depuis C++11)	else	mutable	reinterpret_cast	true	xor_eq
override (C++11)	final (C++11)				

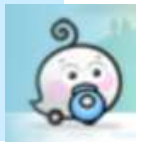
1 - Les commentaires



Commentaire C :

```
/* Ceci est un commentaire C */
```

2a - Types prédéfinis



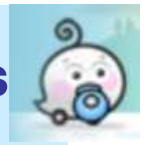
Type de donnée	Signification	Taille (en octets)	Plage de valeurs acceptée
char	Caractère	1	-128 à 127
unsigned char	Caractère non signé	1	0 à 255
short int	Entier court	2	-32 768 à 32 767
unsigned short int	Entier court non signé	2	0 à 65 535
int	Entier	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	-32 768 à 32 767 -2 147 483 648 à 2 147 483 647
unsigned int	Entier non signé	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	0 à 65 535 0 à 4 294 967 295
long int	Entier long	4	-2 147 483 648 à 2 147 483 647
unsigned long int	Entier long non signé	4	0 à 4 294 967 295
float	Flottant (réel)	4	$-3.4*10^{-38}$ à $3.4*10^{38}$
double	Flottant double	8	$-1.7*10^{-308}$ à $1.7*10^{308}$
long double	Flottant double long	10	$-3.4*10^{-4932}$ à $3.4*10^{4932}$
void	Type vide		

- les tableaux à une dimension, dont les indices sont spécifiés par des crochets (’[’ et ’]’).
- les structures, unions et énumérations, les fonctions

Remarque :

- la taille des types peut dépendre des compilateurs,
- en hexadécimal, ils s’écrivent de la manière suivante : 127 -> 0x7f
- un bit est réservé pour le signe

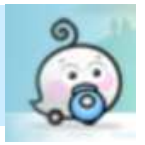
2b - Types prédéfinis : Quelques constantes caractères



le type char est un type entier particulier

CARACTERE	VALEUR (Code ASCII)	NOM ASCII	
'\n'	interligne	0x0a	LF
'\t'	tabulation horizontale	0x09	HT
'\v'	tabulation verticale 0x0b	VT	
'\r'	retour charriot 0x0d	CR	
'\f'	saut de page 0x0c	FF	
'\\'	backslash 0x5c	\	
'\''	cote 0x2c	'	
'\"'	guillemets 0x22	"	
'\a'	signal alerte		
'\b'	retour arriere		

2c- Types prédéfinis : Déclaration type réel



Forme :

<+/-> mantisse * [E,e] <exposant>

Composés :

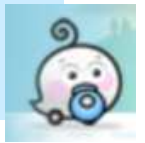
- d'un signe : + ou -
- d'une mantisse : chiffre positif à 1 chiffre avant la virgule
- d'un exposant : entier relatif

3 principaux types : float, double, long float

Exemples :

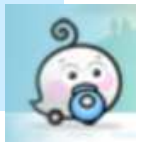
- double d = -1.E5;

3 - Déclarer une variable



- Forme
  - type nomdevariable;
- Exemples
  - int i; /* déclaration de la variable i */
  - char c; /* déclaration du caractère c */

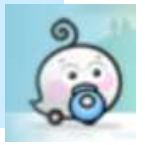
4 - Initialiser une variable



C permet l'initialisation des variables dans la zone de déclaration

Affecter		Initialiser
char c; c='A';	abouti au même résultat que	char c='A';
int i; i=50;	abouti au même résultat que	int i=50;

5 – Affichages : printf



Fonction de la bibliothèque stdio.h
Format : printf("%format",variable);

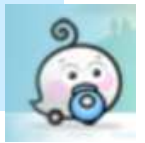
- d pour les décimaux et entiers,
- f pour les réels
- x pour les hexadécimaux,
- c pour un caractère, etc.

Exemples :

```
printf("bonjour");
printf("la valeur est : %d ",i);
printf("les valeurs sont : %d et %d",i,j);
```

6 – Les opérateurs

6.1 - Les opérateurs : arithmétiques



- Sur les réels : + - / *
- Sur les entiers : + - / (quotient de la division), % (reste de la division)

l'affectation se fait grâce au signe '='

Hiérarchie habituelle (* et / prioritaires par rapport aux + et -)

6.2a - Les opérateurs : binaires

- &, et (and)

Table de vérité AND

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

- |, ou, (OR)

Table de vérité OR

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

- ^, ou exclusif (XOR)

Table de vérité XOR

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

6.2b - Les opérateurs : binaires

- ~, complément à 1

Table de vérité NOT

A	NOT A
0	1
1	0

- <<, décalage à gauche
- >>, décalage à droite

Exemple : p = n « 3;

p égal n décalé de 3 bits sur la gauche

6.3 - Les opérateurs : conditionnels

expr1 ? expr2 : expr3

si (expr1) alors expr2 sinon expr3

Exemple : fonction min

```
if (y<z)  x=y; else x=z;
```

Equivalent à

```
x = ( y<z) ? y : z ;
```

6.4 - Les opérateurs : simplifiés

- i=i+1 peut s'écrire i++; (++i;)
- i=i-1 peut s'écrire i--; (--i;)
- a=a+b peut s'écrire a+=b;
- a=a-b peut s'écrire a-=b;
- a=a&b peut s'écrire a&=b;

6.5 - Les opérateurs : logiques

6.5.1 - Quelle utilisation ?

6.5.2 - Opérateurs classiques

6.5.3 - Les expressions

6.5.4 - Et - Ou - !

6.5.1 - Quelle utilisation ?

- Tous les tests :
  - si (expression vraie) ...
  - tant que (expression vraie) ...

- Valeurs booléennes
  - Vrai (1)
  - Faux (0)

6.5.2 - Opérateurs classiques

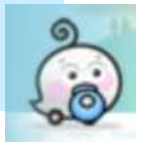
- Le 'ET'

A	B	ET
0	0	0
0	1	0
1	0	0
1	1	1

- Le 'OU'

A	B	OU
0	0	0
0	1	1
1	0	1
1	1	1

6.5.3 - Les expressions



- Egalité

  a == b

- Inégalité

  a != b

- Relations d'ordre

  a < b a <= b a > b a >= b

- Expression

  a est vraie si a est différent de 0

6.5.4 - Et - Ou - !

- Et Logique
  - expression1 && expression2
  - si expression1 et expression2 sont vraies
- Ou Logique
  - expression1 || expression2
  - si expression1 ou expression2 est vraie
- ! (Non)
  - !expression
  - si expression est fausse

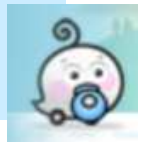
6.5.5 - Priorité des operateurs

priorité	opérateur	parité	associativité	description
1	( )		gauche vers la droite (GD)	parenthésage
2	() [] . ->		GD	appel de fonction, index de tableau, membre de structure, pointe sur membre de structure
3	!	unaire	droite vers la gauche (DG)	négation booléenne
4	~	unaire	DG	négation binaire
5	++ --	unaire	DG	incrémentation et décrémentation
6	-	unaire	DG	opposé
7	(type)	unaire	DG	opérateur de transtypage (cast)
8	*	unaire	DG	opérateur de déréférençage
9	&	unaire	DG	opérateur de référençage
10	sizeof	unaire	DG	fournit la taille en nombre de "char" de l'expression (souvent en octet mais pas toujours, mais sizeof(char) == 1 par définition, voir Caractères)
11	* / %	binaire	GD	multiplication, division, modulo (reste de la division)
12	+ -	binaire	GD	addition, soustraction

6.5.5 - Priorité des operateurs

priorité	opérateur	parité	associativité	description
13	>> <<	binaire	GD	décalages de bits
14	> >= < <=	binaire	GD	comparaisons
15	== !=	binaire	GD	égalité/différence
16	&	binaire	GD	et binaire
17	^	binaire	GD	ou exclusif binaire
18	|	binaire	GD	ou inclusif binaire
19	&&	binaire	GD	et logique avec séquencement
20	||	binaire	GD	ou logique avec séquencement
21	? :	ternaire	DG	si...alors...sinon
22	= += -= *= /= %= ^= &= |= >>= <<=	binaire	DG	affectation
23	,	binaire	GD	séquencement

7 - Constantes



2 solutions :

- const type variable = valeur;
  - exemple : const float PI = 3.14;

- #define PI 3.14
  - utilisation du préprocesseur du compilateur. Le symbole PI sera remplacé par 3.14 dans tout le code.

8 - Conversions de types

Opérateur de cast '(type)'

Exemple :

```
int i;
float f;
i=(int) f;
f=(float)i;
```

9 - Saisie de nombres et de caractères

9.0 – Les Chaine de Caractère

Une chaîne de caractères (appelée *string* en anglais) est une suite de caractères, défini par le code ASCII. (http://www.table-ascii.com/)

Remarque : Il n'existe pas de type spécial chaîne ou string en C

En langage C, une chaîne de caractères est un tableau, comportant plusieurs données de type **char**, dont le dernier élément est le caractère **nul '\0'**, c'est-à-dire le premier caractère du **code ASCII** (dont la valeur est 0).

Il existe des notations particulières et des fonctions spéciales pour le traitement de tableaux de caractères.

9.1a - Fonction getchar

- Saisie d'un caractère
- Prototype : int getchar();
- Bibliothèque stdio.h
- Moins gourmande que scanf
- Utilise aussi le flux d'entrée

9.1b - Fonction getchar

- Prototype : int getchar()
- Utilisation
  - avec retour de variable
  - sans retour de variable
- Exemple
  - char alpha;
  - alpha=getchar();

9.2a - Fonction scanf

- Définie dans stdio.h
- Prototype :

  int scanf(const char* format, &var)
- Variables à saisir sont formatées
- Les variables modifiées sont précédées de &
- Arrêt de la saisie avec "RETURN"

9.2b - Fonction scanf : Exemple d'utilisation

```
char car;
int i;
float r;
char chaine[100];

scanf("%c",&car);  /* saisie d'un caractère */
scanf("%d",&i);     /* saisie d'un entier */
scanf("%f",&r);      /* saisie d'un nombre réel */
scanf("%10s",chaine);      /* saisie d'une chaine de caractère  */
```

Remarque:
format non respecté, saisie ignorée

9.2c - Fonction scanf

%[*][width][modifiers]type

http://www.cplusplus.com/reference/clibrary/cstdio/scanf/

where:

*	An optional starting asterisk indicates that the data is to be retrieved from `stdin` but ignored, i.e. it is not stored in the corresponding argument.
width	Specifies the maximum number of characters to be read in the current reading operation
modifiers	Specifies a size different from `int` (in the case of `d`, `i` and `n`), `unsigned int` (in the case of `o`, `u` and `x`) or `float` (in the case of `e`, `f` and `g`) for the data pointed by the corresponding additional argument: **h :** `short int` (for `d`, `i` and `n`), or `unsigned short int` (for `o`, `u` and `x`) **l :** `long int` (for `d`, `i` and `n`), or `unsigned long int` (for `o`, `u` and `x`), or `double` (for `e`, `f` and `g`) **L :** `long double` (for `e`, `f` and `g`)
type	A character specifying the type of data to be read and how it is expected to be read. See next table.

scanf type specifiers:

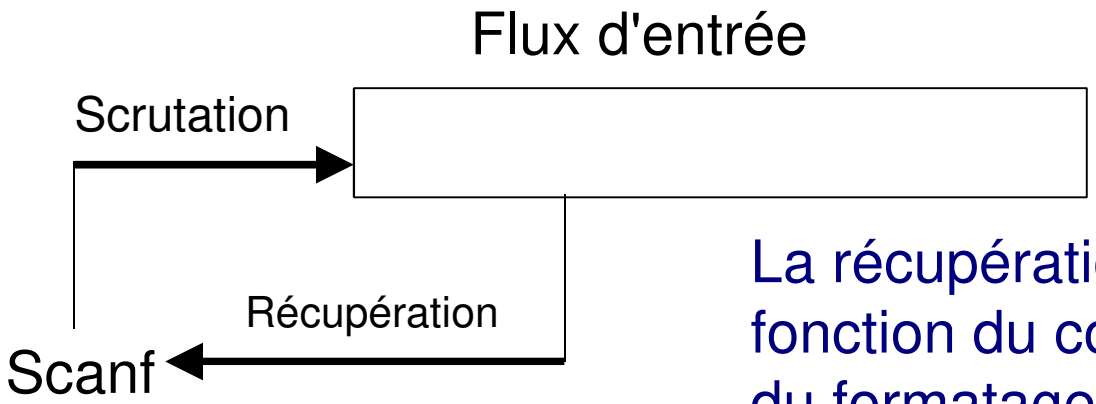
type	Qualifying Input	Type of argument
`c`	**Single character:** Reads the next character. If a *width* different from 1 is specified, the function reads *width* characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	`char *`
`d`	**Decimal integer:** Number optionally preceded with a + or - sign.	`int *`
`e,E,f,g,G`	**Floating point:** Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally folowed by the `e` or `E` character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	`float *`
`o`	**Octal integer.**	`int *`
`s`	**String of characters.** This will read subsequent characters until a whitespace is found (whitespace characters are considered to be blank, newline and tab).	`char *`
`u`	Unsigned decimal integer.	`unsigned int *`
`x,X`	Hexadecimal integer.	`int *`

9.3 - Flux d'entrée

- Saisie clavier -> stockage dans buffer
- Buffer de type FIFO (First In First Out)
- Dernier caractère stocké : LF (0x0A) (\n)
- Buffer appelé Flux D'entrée
- Compilation -> vide le flux d'entrée

9.4 - Scanf et Flux d'entrée

- À l'appel, la fonction scanf scrute le flux



La récupération se fait en fonction du contenu du buffer et du formatage du scanf

- Problèmes liés à LF !!
  - Exemples : saisie de 2 caractères à la suite, caractère puis entier, etc.

10 - Les structures de contrôle

10.1 - Si … alors … sinon … => if () {} else {}
10.2 - Au cas … ou faire … => switch () { case }
10.3 - Tant que … faire … => while () {}
10.4 - Répéter … tant que …=> do {} while ()
10.5 - Boucle Pour … faire … => for () {}
10.6 - Saut => goto
10.7- Rupture de séquence => break, continue, return

10.1 - si … alors … sinon … => if () {} else {}

Si (expression conditionnelle vraie)
alors { instructions 1 }
sinon { instructions 2 }

```
if (expression) {
   instructions 1;
}
else {
   instructions 2;
}

if (expression) {
   instructions 1;    /* bloc else
     optionnel */
}
```

```
if (expression)
  instruction;    /* bloc d'1
     instruction */

if (expression) instruction1;
else instruction2;
```

10.1 - Exemple

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

int main(int argc, char *argv[]) {
   char quit = '\0';
   printf(" Bonjour ! \n");
   printf(" Appuyer sur la touche q pour quitter \n");
   quit=getchar();
   if ( quit == 'q' ) return 0;

   if ( quit == 'Q' ) {
    printf(" Appuyer sur la touche q et pas Q pour quitter \n");
    quit=getchar();
   }
   else
   {
    printf("  Appuyer sur la touche q pour quitter \n");
    quit=getchar();
   }

   if ( quit == 'q' ) return 0;
   else printf("Dernier essai, Appuyer sur la touche q pour quitter \n");

   quit=getchar();
   return 0;
}
```

10.2 - Au cas … ou faire …=> switch () { case }

Au cas ou la variable vaut :
valeur 1 : faire … ;
valeur 2 : faire … ; etc.

```
switch (variable de type char ou int)  {
    case valeur 1 : …;
            …;
            break;
    case valeur 2 : …;
            …;
            break;
    default :          …;
            …;
}
```

10.2 - Exemple

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

int main(int argc, char *argv[])
{
   char quit = '\0';
   printf ( " Bonjour ! \nAppuyer sur la touche q pour quitter \n");
   quit=getchar();
   switch ( quit )
   {
      case 'q' :
            return 0; break;
      case 'Q' :
            printf ( " Appuyer sur la touche q et pas %c pour quitter \n",quit);
          quit=getchar();
            break;
      case 'r':case 'R':case 'S' : case 'T' :
            printf ( " Appuyer sur la touche q et pas %c pour quitter \n",quit);
          quit=getchar();
            break;
      default :
            printf ( " Appuyer sur la touche q et pas %c pour quitter \n",quit);
            quit=getchar();
            break;
   }
   if ( quit == 'q' ) return 0;
   else printf ( "Dernier essai, Appuyer sur la touche q pour quitter \n");
   quit=getchar();
   return 0;
}
```

10.3 - Tant que … faire => while () {}

Tant que (expression vraie) faire
{bloc d 'instructions}

```
while (expression vraie)  {
     bloc d 'instructions }
```

10.3 - Exemple

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

int main(int argc, char *argv[])
{
    char quit = '\0';
    while ( quit != 'q' )
    {
      printf ( " Bonjour ! \n Appuyer sur la touche q pour quitter \n");
      quit=getchar();
    }
    return 0;
}
```

10.4 - Répéter … tant que …=> do {} while ()

Répéter {bloc d 'instructions}
tant que (expression vraie)

```
do {bloc d 'instructions} while
(expression vraie);
```

10.4 - Exemple

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

int main(int argc, char *argv[])
{
    char quit = '\0';
    do
    {
      printf ( " Bonjour ! \n Appuyer sur la touche q pour quitter \n");
      quit=getchar();
    }
    while ( quit != 'q' );

    return 0;
}
```

10.5 - Boucle Pour … faire … => for () {}

Pour (initialisation;condition;modification) faire {
bloc d 'instructions}

```
for (initialisation;condition;modification)  {
    bloc d 'instructions}
```

10.5 - Exemple

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

int main(int argc, char *argv[])
{
    char quit = '\0';
    int i;
    printf ( " Bonjour ! \n Appuyer sur la touche q pour quitter \n");
    quit=getchar();
    for ( ; quit!='q'; )
    {
        for (i = 0 ; i < 5; i++) printf ( "Bonjour ! \n");
        printf ( " Appuyer sur la touche q pour quitter => ");
        scanf("%c",&quit);getchar();
     }
    return 0;
}
```

10.6 - Saut => goto

goto étiquette ;
étiquette :

```
goto etiquette ;
etiquette :
```

10.6 - Exemple

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

int main(int argc, char *argv[])
{
    char quit = '\0';

    reprise :

    printf ( " Bonjour ! \n Appuyer sur la touche q pour quitter \n");
     scanf("%c",&quit);getchar();
     if ( quit!='q' ) goto reprise;

    return 0;
}
```

10.7 - Rupture de séquence => break, continue, return

- ***Return[valeur]*** : permet de quitter immédiatement la fonction en cours.

- ***break*** : permet de passer à l'instruction suivant l'instruction *while*, *do*, *for* ou *switch* la plus imbriquée.

- ***continue*** : saute directement à la dernière ligne de l'instruction *while*, *do* ou *for* la plus imbriquée.

10.7 – Exemple

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

int main(int argc, char *argv[])
{
   char quit = '\0';
     int i;
   while ( quit!='q' )
   {
      for (i=0;i< 5;i++)
      {
               printf ( " Bonjour ! : num => %i \n",i);
               printf ( " Appuyer sur la touche 'o' pour quitter la boucle\n");
               scanf("%c",&quit);getchar();
               if ( quit=='o' ) break;
      }
   printf ( " Appuyer sur la touche 'r' pour recommencer la boucle\n");
   printf ( " Appuyer sur la touche 'q' pour quitter \n");
   scanf("%c",&quit);getchar();

   if ( quit=='r' ) continue;
   if ( quit!='q' ) return 0;
   }
   return 0;
}
```

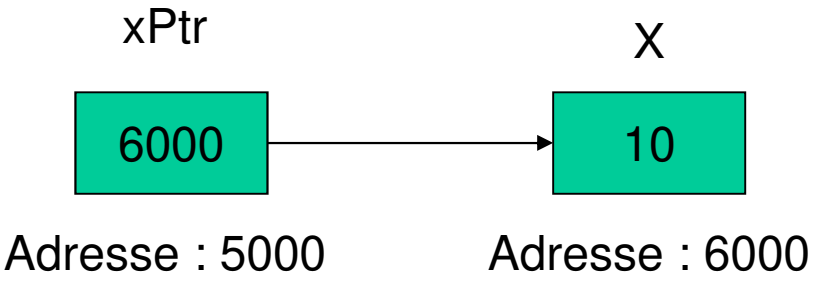
11 - Les pointeurs

11.1a - Définition Pointeur

- Tout emplacement mémoire a une adresse
- Un pointeur est une variable qui contient une adresse mémoire
- On dit que le pointeur pointe sur cette adresse

Déclaration pointeur
pointeur = type * **p**
Récupération valeur pointée
* **p**

```
int main ()
{
  int x = 10;
  int * xPtr;  /* xPtr pointeur pointant sur type int */
  xPtr = &x;
}
```



5000 6000 10
6000 10

11.1b - Définition Adresse

Notion :

- Un objet manipulé est stocké dans sa mémoire.
- Une mémoire est constituée d'une série de « cases », ou sont stockées des valeurs (variables ou instructions)
- Pour accéder au contenu de la case mémoire où l'objet est enregistré, il faut connaître l'emplacement en mémoire de l'objet à manipuler.

Cet emplacement est appelé l'adresse de la case mémoire, ou l'adresse de la variable ou l'adresse de la fonction.

Toute case mémoire a une adresse unique.

int x = 10

X

10

Adresse : 6000

Notation adresse

adresse x = **&x**

11.2 - Opérateur &

- Retourne l ’adresse d’une variable en mémoire
- cf scanf

  – Exemple :

```
int i=8;
int* iptr = &i;
printf(“Voici i et son adresse %d , %p“,i,&i);
printf(“Voici iptr et son adresse %p , %p, %d “,iptr,&iptr,*iptr);
*iptr=5;
printf(“Voici i et son adresse %d , %p“,i,&i);
printf(“Voici iptr et son adresse %p , %p, %d “,iptr,&iptr,*iptr);
```

11.3 - Allocation dynamique de mémoire

L'allocation dynamique de mémoire permet de réserver de la mémoire (appelée encore *allocation*) pendant l'exécution d'un programme. La quantité de mémoire utilisée par le programme est variable.

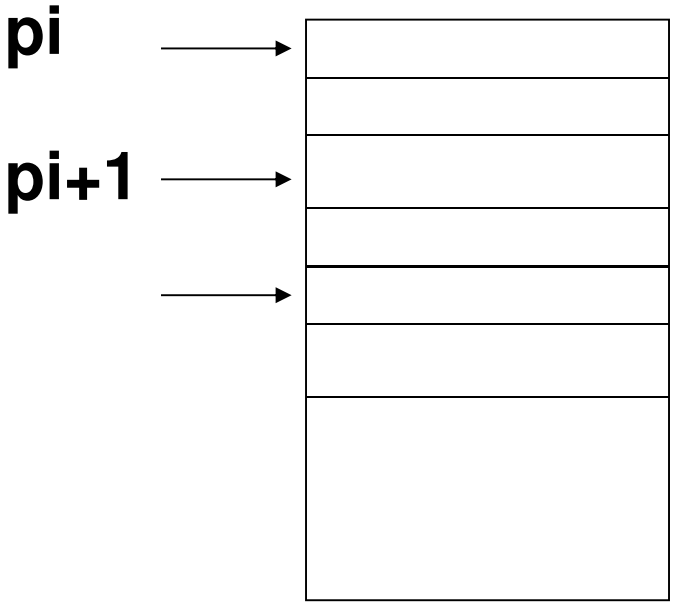
11.4a - Arithmétique des pointeurs

- Déplacement du pointeur dans l'espace mémoire
- Déplacement multiple du type de variable pointée

  – Exemples :

```
int i = 8;
int *pi = NULL;
pi = &i ;
*pi=12;
/*  *(pi+1)=3; */
printf(‘’%d %d %p %p %p \n’’, i, *pi, &i, pi , &pi);
```

11.4b - Arithmétique des pointeurs




```
int*pi;
*pi=0;
```

1 entier = 2 octets

11.5 - Opérateur de Déréférencement et d'indirection

```
int i = 0;
int *pi = 0;
pi = &i;
*pi = *pi+1;
```

Attention *pi+1 != *(pi+1)

11.6 - Allocation dynamique

- Réservation obligatoire pour pérennité
- Allocation par la fonction malloc
- Prototype dans alloc.h
- Exemples :

```
int i=0;
char * pc=(char*)malloc(10);
int* pj= 0 ;
/*pi=(int*)malloc(16);*/
int* pi = (int*) malloc(4*sizeof(int));
for (i=0;i < 4 ;i++) {
        pi[i]=i; /* *(pi+i)=i;*/
        printf("%d \n",pi[i]);
}
pj=(int*)malloc(sizeof(int));
```

11.7 - Libération de la mémoire

- Fonction free(pointeur)
- Evite l'encombrement mémoire par la destruction des pointeurs

- Exemples :

```
free(pi);
free(pc);
```

11.8 - Allocation dynamique en C

Allocation dynamique : malloc

- signature de la fonction :
  - void* malloc(unsigned int) ;
- Prototype dans alloc.h
- Exemples :
  - char* pc;
  - pc=(char*) malloc(10);
  - int * pi;
  - pi=(int*) malloc(sizeof(int)*16);

Libération mémoire : free

- Libération par la fonction
  - free(pointeur)
- Évite l'encombrement mémoire par la destruction des pointeurs
- Exemples :
  - free(pi);
  - free(pc);

11.9 - Attention

- Ne pas affecter directement une valeur dans un pointeur !
  - Exemple : **char *pc;**

pc=0xfffe;

11.10 - Pointeurs constants ou volatiles

```
int const *pi;
const int *pi = (int*)malloc(sizeof(int));  /* pi est un pointeur sur un entier
                                               constant. */
/* pi = 2; erreur */
pi = (int*)malloc(sizeof(int));

int j;
int * const pj=&j; /* pi est  un pointeur constant sur un entier non constant */

*pj = 2;
/* pj = (int*)malloc(sizeof(int));  erreur*/

const int * const pc = &j;

const int *pi[10]; /* pi est un tableau de 10 pointeurs d'entiers constants */
```

11.11 - Fonction scanf

- Utilisation : scanf(“%d“,&i);
- Le paramètre de la fonction est l’adresse de la variable
- Passage par adresse = modification possible du paramètre

11.12 - Passage de paramètre par variable ou par valeur

Passage de paramètre par valeur

```
double mult( double x1, double y1 )
{
     y1 = x1 * y1;
     return y1;
}

void main()
{
     double x = 2;
      double y = 3;
     double z = mult(x,y);
    /*en sortie z = 6 , x=2 , y=3*/
}
```

Passage de paramètre par variable

```
double mult( const double* const  x1,
     double* y1 )
{
     *y1 = (*x1) * (*y1);
     return *y1;
}

void main()
{
     double x = 2;
      double y = 3;
     double z = mult(&x,&y);
    /*en sortie z = 6 , x=2 ,
      Attention y a été modifié y=6*/
}
```

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

void construire(int** pi,int taille) {
 int i;
if ( pi==NULL) exit(1);
*pi=(int*)malloc(sizeof(int)*taille);
if ( (*pi)==NULL) exit(2);
for ( i=0;i< taille;++i) (*pi)[i] = 0 ;
}
void afficher(const int * pi,int taille) {
 int i;
 if ( pi==NULL)  return;
 for ( i=0;i< taille;++i) printf("tab[%d]=%d\n",i,pi[i]);
}
void liberer (int ** pi) {
if ( pi==NULL) return ;
if (*pi == NULL) return;
free(*pi);
*pi=NULL
}

int main() {
        int* pi = NULL;
        construire(&pi,16);printf("%p\n",pi);
        afficher(pi,16); printf("%p\n",pi);
        liberer(&pi); printf("%p\n",pi);
        afficher(pi,16);
        return 0;
}
```

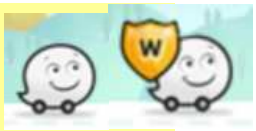
11.13 - Pointeur de fonctions

type (*identificateur) (paramètres);

Exemple :

```
double (*pf)(double, double);
pf =  mult;
double z = pf (2.,3.);

/* pf est un pointeur de fonction */.

typedef int (*PtrFonct)(int, int);
PtrFonct pf;
```

11.14 - Conclusion



Les pointeurs sont très utilisés en C, mais ils sont très dangereux, quelques règles de survie :

Règle 1 : TOUJOURS INITIALISER LES POINTEURS QUE VOUS UTILISEZ.

Règle 2 : VÉRIFIEZ QUE TOUTE DEMANDE D'ALLOCATION MÉMOIRE A ÉTÉ SATISFAITE.

Règle 3 : LORSQU'ON UTILISE UN POINTEUR, IL FAUT VÉRIFIER S'IL EST VALIDE.

12 - Les tableaux et les chaînes de caractères

12.1 – Les Tableaux

Tableau caractérisé par
- sa taille
- ses éléments repéré par son indice
- commence à 0 finit à dim-1

Tableau à une dimension
- Déclaration : Type nom[dim]; /* réservation de dim places de grandeur Type */
- Exemples
  - int Tab[10];
  - float vecteur[20];

Tableau à deux dimensions
- Déclaration : type nom[dim1][dim2];
  - int compteur[4][5];
  - float nombre[2][10];
- Utilisation : nom[indice1][indice2];
  - compteur[3][4]=3;
  - printf("%d",compteur[1][2]);

12.2 - Déclaration

On peut initialiser au moment de la déclaration

- Exemple
  - **int liste[10]={1,2,45,5,67,120,32,48,3,10};**
  - **int tab[2][3]={{1,4,8},{8,5,3}}**

Remarque

- Consomment beaucoup de place
- Nécessité de les dimensionner au plus juste

12.3 - Conversions des tableaux en pointeurs

Accès aux éléments d'un tableau par pointeur

```
int tableau[100];
int *pi=tableau;

tableau[3]=5;     /* Le 4ème élément est initialisé à 5 */
*(tableau+2)=4; /* Le 3ème élément est initialisé à 4 */
pi[5]=1;          /* Le 6ème élément est initialisé à 1 */
```

12.4 - Pointeurs, tableaux

Déclaration d'un pointeur (adresse du premier élément) + allocation de l 'espace mémoire
On peut donc déclarer nous-même un pointeur et allouer la mémoire ' manuellement '

```
int *tab = (int*) malloc(sizeof(int)*5)
```

Avantage : allocation de l 'espace mémoire au moment de l 'exécution, attention ne pas oublier de détruire la mémoire:

```
free(tab) ;
```

Il est possible de pointer sur un élément particulier d'un tableau :

```
int tab[5];
int *ptr = &tab[2];    /* ptr pointe sur le 3eme élément du tableau */
```

Un nom de tableau utilisé dans une expression est converti en une adresse du début de ce tableau :

```
int tab[5];
int *ptr = tab;      /* équivalent a : "ptr = &tab[0]" */
```

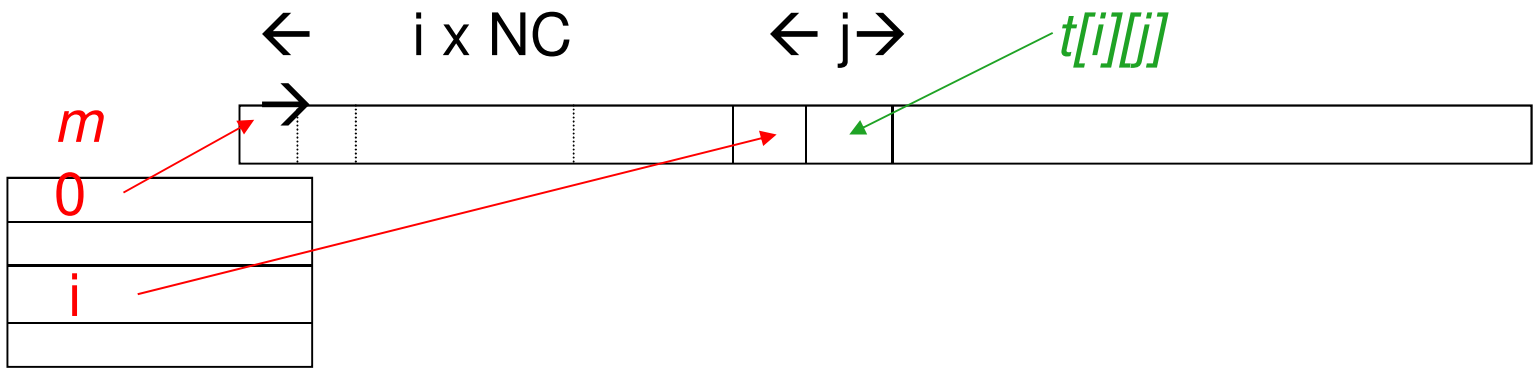
12.5 - Tableaux à 1 dimension

- Equivalences :
  - » **int *tableau**
- int tableau[10];

  tableau=(int*)malloc(sizeof(int)*10);

  - » ***tableau**
    tableau[0]
  - » ***(tableau+i)**
    tableau[i]
  - » **tableau**
    &tableau[0]
  - » **(tableau+i)**
    &tableau[i]

12.6 - Tableaux à 2 dimensions



- Equivalences :
  - » t[0] &t[0][0]
  - » t[1] &t[1][0]
  - » t[i] &t[i][0]
  - » t[i]+1 &(t[i][0])+1

t[NL][NC] , *m[NL]

t[i][j]-> t+i*NBCOL+j;

12.7 - Les chaînes de caractères

- Une chaîne = un tableau de caractères
- Déclaration
  - **char nom[dim]; ou char* nom;**
    nom=(char*)malloc(dim);
  - **exemple : char texte[10]; ou char* texte;**
    texte=(char*)malloc(10);
- Seulement dim-1 caractères disponibles
- Caractère NUL (‘ \0 ’) à la fin

12.8 - Affichage et saisie d 'une chaîne

- printf et format %s
  - Exemple : **char texte[10]=“bonjour“;**
    printf(“voici le texte %s“,texte);
- scanf et format %s
  - Exemple : **char texte[10];**
    scanf(“%s“,texte);

13 – Les fonctions

13.1 - Généralités

- En C procédures = fonctions
- Pas de déclaration de fonction dans une autre fonction
- Possible appel d ’une fonction dans une autre fonction
- Variables locales
- Variables globales

La déclaration d'une fonction

```
type_de_retour Nom_de_la_Fonction
( type1 argument1, type2 argument2, ... )
{
        liste d'instructions ;
}
```

```
int add( int i, int j) {
        int k = i + j ;
        return k;
}
```

13.2 - Variables globales

- Déclarées avant le main
- Eviter le surnombre

- Exemple :

```
#include <stdio.h>
int i;
int main () { ...
```

13.3 - Variables locales

- Locales aux fonctions : connues uniquement à l ’intérieur de la fonction.

- Locales au main : connues dans le main mais pas à l ’intérieur des fonctions

13.4 - Fonctions typées sans arguments

- Fonction renvoyant une valeur
- Type de la valeur déclaré avec la fonction
- Valeur retournée spécifiée grâce à return

13.5 - Fonctions avec arguments

- Type void ou non
- Utilisation des valeurs de certaines variables du programme
- En-tête : variables arguments
  - Exemple : **int truc(int); /* prototype réduit */**
    int truc(int x); /* prototype complet */

13.6- Arguments par adresse

- Nécessaire pour modifier la variable
- Déclaration en-tête : utiliser les pointeurs
- Tableaux et Pointeurs

- Exemples :
  - Déclaration

    void Calcul(int x,int* y);
  - Appel
  - int j = 0; **int * k = &j;**

    Calcul(i,&j);

    Calcul(i,k);

13.7 - Utilisation dans le programme

```
#include <stdio.h>
#include "produit.h"
 /*int produit (int, int); */

 int main () {
    int a, b,c;
    scanf("%d",&a); scanf("%d",&b); c = produit(a,b);
    printf("\nle produit vaut %d\n",c);
    return 0;
 }
/* produit.c
 int produit (int a, int b){return a*b;}*/
```

13.8 - Utilisation des entêtes .h

```
/****************************/
/***     fichier: produit.h      ***/
/***   produit de 2 entiers    ***/
/****************************/
#ifndef PRODUIT_H
#define PRODUIT_H
   int produit (int, int);

   int plus(int,int);
#endif  /* PRODUIT_H */
```

```
/****************************/
/***     fichier: produit.c      ***/
/***   produit de 2 entiers    ***/
/****************************/
#include  « produit.h »
int produit (int a, int b)
 {
    return a*b;
 }
```

```
/****************************/
/***     fichier: plus.c      ***/
/***   produit de 2 entiers    ***/
/****************************/
#include  « produit.h »
int plus(int a, int b)
 {
    return a+b;
  }
```

13.9 - Fonction main

```
int main() /* Plus petit programme C. */
{
    return 0;
}
```

La fonction main est une fonction spéciale, lorsqu'un programme est chargé, son exécution commence par l'appel de celle-ci, elle marque le début du programme.

Remarque :

Elle ne peut pas être récursive.

13.10 - main : passage de paramètres

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

int main (int argc, char* argv [], char* envp[] )
{
  int i=0;

  printf("Nb argument : %d \n",argc);

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

  printf("\n variables d'environnement.\n");

  for (i=0;envp[i] != NULL ;i++)
          printf("envp[%d]= %s \n",i,envp[i]);
  return 0;
}
```

14 - Types de variables complexes

14.1 - Types synonymes : typedef

- Création de ses propres types
- Déclaration en début de programme

- Exemple
  - **typedef int entier; /* type entier = int */**
  - **typedef int Type_tab[5]; /* type Type_tab = tableaux de 5 entiers */**
  - **typedef char* fcar; /* type fcar = pointeur de char */**
  - **typedef int* vecteurInt;**
  - **typedef vecteurInt* matriceInt;**

14.2 - Structures

- Types particuliers
- Composition d'un ensemble de types

  typedef struct {

  types; } nomdutype;

- Exemple
  - **typedef struct { char* nom; char* prenom;} personne;**
  - **personne toto;**

14.3 - Accéder aux champs

- Syntaxe : objet.champ
  - Exemple :

```
personne toto;
toto.nom = (char*) malloc(sizeof(char)*10);
scanf("%9s",toto.nom);
free(toto.nom);
```

- Pas de protection sur les champs

14.4 - Structures et pointeurs

- Pointeurs sur des structures
  - allocation dynamique de structures
  - modification d'une structure dans une fonction
  - Exemple :
  - **typedef struct { char* nom; char* prenom; unsigned short int age} personne;**
  - **personne* toto;**
  - **toto = (personne*)malloc(sizeof(personne));**
  - **(*toto).age=12;**
  - **free(toto);**

- Notation équivalente : ->
  - Exemple : **(*toto).age** **toto->age**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct { char* nom; char* prenom; unsigned short int age;} personne;

int main() {
personne* toto;
toto = (personne*)malloc( sizeof(personne));
toto->nom = (char* ) malloc(sizeof(char)*256);
toto->nom[0]='\0';
printf ( "Rentrer le nom\t:\t");scanf("%255s",toto->nom);
printf("Rentrer age\t: \t);scanf("%hu",& ( (*toto).age ) );
printf(":%s:%hu:\n",toto->nom,toto->age);
free(toto->nom);
free(toto);
return 0;
}
```

main.c

```
#include "personne.h"

int main() {
personne* toto = personne_creation(); /* personne* personne_creation(); */
personne_initialiser(toto); /* void personne_initialise( personne* );  */
personne_afficher(toto);  /* void personne_afficher( const  personne* );  */
personne_free(&toto); /*  void personne_free ( personne** ) ;*/
return 0;
}
```

personne.h

```
#ifndef PERSONNE_H
#define PERSONNE_H

typedef struct { char* nom;
                 char* prenom;
                  unsigned short int age;} personne;

typedef enum {CREATION_ERR01=1, CREATION_ERR02=2) Erreur ;

personne* personne_creation();
void personne_initialise( personne* );
void personne_afficher( const  personne* );
void personne_free ( personne** ) ;

#endif
```

personne.c

```
#include "personne.h"
#include <stdlib.h>


personne* personne_creation() {
        personne* toto = NULL;
        return toto;
}
void personne_initialise( personne* toto) {
}
void personne_afficher( const  personne* toto) {
}
void personne_free ( personne** toto ) {
}
```

personne_creation

```
personne* personne_creation() {
        personne* toto = (personne*) malloc(sizeof(personne));
        if ( toto == NULL ) exit (CREATION_ERR01);
        toto->nom = (char* ) malloc(sizeof(char)*256);
        if (toto->nom == NULL ) exit (2);
        toto->nom[0]='\0';
        toto->prenom = (char* ) malloc(sizeof(char)*256);
        if ( toto->prenom == NULL) exit (3);
        toto->prenom[0]='\0';
        toto->age = 0;
        return toto;
}
```

personne_free

```
void personne_free ( personne** toto ) {
        if ( NULL == toto ) return;
        if ( NULL == *toto) return;
        if ( NULL != (*toto)->nom ) free( (*toto)->nom);
         if ( NULL != (*(*toto)).prenom ) free((*(*toto)).prenom);
        free(*toto);
        *toto=NULL;

}
```

personne_initialiser

```
void personne_initialiser(personne *toto) {
if ( NULL == toto ) exit (10);
printf ( "Rentrer le nom\t:\t");
if ( scanf("%255s",toto->nom) != 1 ) exit (11) ;
printf ( "Rentrer le prenom\t:\t");
if ( scanf("%255s",toto->prenom) != 1 ) exit (12) ;
printf("Rentrer age\t: \t);
if ( scanf("%hu",& ( (*toto).age ) ) !=1) exit (13) ;
}
```

personne_afficher

```
void personne_afficher( const  personne* toto) {
        if (NULL==toto) return;
        printf("nom : %s \nprenom : %s \nage : %hu \n",
                toto->nom,
                toto->prenom,
                toto->age);
}
```

14.5 – Le type union

Syntaxe :

```
union <nom de l'union>
    {
    <type de donnée> <nom de la variable>;
    ...
    <type de donnée> <nom de la variable>;
    };
```

Description:

Le type union permet de définir des variables qui occupent le **même emplacement mémoire.**

La taille d'une union est celle de la donnée la plus grande de cette union.

L'écriture dans une variable **écrase la valeur** des autres variables de l'union.

Il faut utiliser le sélecteur d'enregistrement (.) pour accéder aux éléments d'une union.

14.6 - Exemple : Type union

```
union Nombre
    {
    int i;
    double d;
    char cNombre[12];
    };

Nombre nb;
nb.d = 2.3
```

14.7 – Le type enum

Syntaxe :

enum [<nom de l'énumération>] {<nom d'une constante> [= <valeur>], ...} [<liste de variables>];

Description:

Le mot clé enum permet de définir un ensemble de constantes de type entier(int).
Une énumération fournit des identificateurs mnémoniques pour un ensemble de valeurs entières et finies
Une variable énumérée ne peut se voir affecter qu'un de ses énumérateurs(<constante>).
En l'absence d'une <valeur>, la première constante prend la valeur zéro.
Toute constante sans <valeur> sera augmentée d'un par rapport à la constante précédente.

Exemple :

```
enum jour { Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche};
enum jour { Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche} variable_jour;
enum { Lundi=1, Mardi=2, Mercredi=3, Jeudi=4, Vendredi=5, Samedi=6, Dimanche=7};
```

15a – Les Directives de Précompilation

Toutes les commandes du préprocesseur commencent :

- en début de ligne,
- par un signe dièse (#).

#include :

#include "fichier" ou #include <fichier>

#define POSIX_VERSION 1001 (en C++ const int POSIX_VERSION = 1001;)

Le préprocesseur définit un certain nombre de constantes :

- __LINE__ : donne le numéro de la ligne courante
- __FILE__ : donne le nom du fichier courant
- __DATE__ : renvoie la date du traitement du fichier par le préprocesseur
- __TIME__ : renvoie l'heure du traitement du fichier par le préprocesseur
- __cplusplus : définie uniquement dans le cas d'une compilation C++

15b – Les Directives de Précompilation

Compilation conditionnelle :

#ifdef identificateur

...

#endif

d'autres directives de compilation conditionnelle :

- #ifndef (if not defined ...)
- #elif (sinon, si ...)
- #if (si ...)

autres commandes :

- # : ne fait rien (directive nulle) ;
- #error message : permet de stopper la compilation en affichant le message d'erreur donné en paramètre ;
- #line numéro [fichier] : permet de changer le numéro de ligne courant et le nom du fichier courant lors de la compilation ;
- #pragma texte : permet de donner des ordres spécifiques à une implémentation du compilateur

15c - Les Directives de Précompilation : Macro

#define macro(paramètre[, paramètre [...]]) définition

Exemple :

```
#define MAX(x,y) ((x)>(y)?(x):(y))
#define MIN(x,y) ((x)<(y)?(x):(y))
#define CHAINE(s) #s // transforme son argument en chaîne de caractères.
#define NOMBRE(chiffre1,chiffre2) chiffre1##chiffre2 // permet de construire un nombre à deux chiffres.
```

```
#ifdef MON_DEBUG
#include <stdio.h>
#define TRACE(mes) { \
printf("Trace : %s [%d] : %s\n",__FILE__,__LINE__,mes); \
}
#else
#define TRACE(mes) {}
#endif
int main() {
TRACE("Message");
/*
{ printf("Trace : %s [%d] : %s\n",__FILE__,__LINE__,"Message); }
*/
return 0;
}
```

16 - Les Fichiers

16.1 - Généralités

- Fichier = ensemble d'informations stocké sur une mémoire de masse
- En C un fichier = une suite d'octets
- Les informations sont localisées par un pointeur
- 2 types d'accès : séquentiel et direct
- 2 types de codage : binaires et ASCII

16.2 - Manipulation de fichiers

- Bibliothèque stdio.h
- opérations : créer, ouvrir, fermer, lire, écrire, détruire, renommer
- fonctions de base en C : accès séquentiel

16.3 - Déclaration d'un fichier

- FILE* fichier;
- définition d'un pointeur sur le fichier

16.4 - Ouverture

- FILE* fopen(char* nom,char* mode)
- modes texte : “r“, “w“, “w+“, “r+“, “a+“
- modes binaire : “rb“, “wb“, “wb+“, “rb+“, “ab+“
- ouverture = positionnement début de fichier (sauf pour “a+“ et “ab+“
- Exemple : fichier=fopen(“toto.txt“, “r“);

16.5 - Fermeture

- int fclose(FILE* fichier)
- Retourne 0 si la fermeture s'est bien passée
- Exemple :

```
fclose(fichier);
```

16.6 - Destruction

- int remove(char* nom)
- Retourne 0 si la destruction s'est bien passée
- Exemple :

```
remove("toto.txt");
```

16.7 - Renommer

- int rename(char* ancien_nom,char* nouv_nom)
- Retourne 0 si l'opération s'est bien passée
- Exemple :

```
rename("toto.txt", "tutu.txt");
```

16.8a - Ecriture

- Fichiers binaires
- int fwrite(void* p,int taille_bloc,int nb_bloc,FILE* fic)
- Ecrit les nb_blocs de taille taille_bloc pointés par p dans le fichier fic
- Exemple : int tab[3];

```
fwrite(tab,sizeof(int),3,fichier);
```

16.8b - Ecriture

- Fichiers textes
- int fprintf(FILE* fic,char* chaîne_format,liste de données)
- Ecrit la chaîne_format (avec les données) dans le fichier fic
- Exemple : fprintf(fichier,“Salut“);

  fprintf(fichier, “coucou %d“,i);

16.9a - Lecture

- Fichiers binaires
- int fread(void* p,int taille_bloc,int nb_bloc, FILE* fic)
- Lit les nb_blocs de taille taille_bloc pointés par p dans le fichier fic
- retourne le nombre de blocs lus et 0 si fin de fichier
- Exemple : int tab[3];

```
fread(tab,sizeof(int),3,fichier);
```

16.9b - Lecture

- Fichiers textes
- int fscanf(FILE* fic,char* format,liste d'adresses)
- Lit les données en suivant le format dans le fichier fic
- Exemple : fscanf(fichier,"%d",&i);

16.10 Gestion des erreurs

- fopen retourne NULL si erreur
- int feof(FILE* fichier) = 0 tant que la fin de fichier n'est pas atteinte

17 - Utilisation d'une bibliothèque

17.1 - Notion de prototype

- Fichiers .h
- Fichiers d'en-tête
- prototypes de fonctions
- Vérification de syntaxe

- exemple : prototype de getchar est int getchar()

17.2 - Fonction ne renvoyant rien

- Fonctions de type void
- On l ’appelle tel quel
  - exemple :
    - perror
    - prototype : void perror(const char*);
    - prototype dans stdio.h

17.3 - Fonction renvoyant une valeur

- Fonctions de type int ou char, ou long, etc.
- On récupère sa valeur dans une variable
  - exemple :
    - getchar
    - prototype : int getchar();
    - prototype dans stdio.h

17.4 - Fonction avec passage de paramètres

- Fonctions utilisant le paramètre pour fonctionner
- Utilisation d'une même fonction pour plusieurs données
  - exemple :
    - log
    - prototype : double log(double);
    - prototype dans math.h

18 - librairies standard

18.1a - Entrées-sorties <stdio.h>

Manipulation de fichiers	
fonction	**action**
fopen	ouverture d'un fichier
fclose	fermeture d'un fichier
fflush	écriture des buffers en mémoire dans le fichier

Entrées et sorties formatées		
fonction	**prototype**	**action**
fprintf	int fprintf(FILE *stream, char *format, ...)	écriture sur un fichier
fscanf	int fscanf(FILE *stream, char *format, ...)	lecture depuis un fichier
printf	int printf(char *format, ...)	écriture sur la sortie standard
scanf	int scanf(char *format, ...)	lecture depuis l'entrée standard
sprintf	int sprintf(char *s, char *format, ...)	écriture dans la chaîne de caractères s
sscanf	int sscanf(char *s, char *format, ...)	lecture depuis la chaîne de caractères s

18.1b - Entrées-sorties <stdio.h>

Impression et lecture de caractères		
fonction	**prototype**	**action**
fgetc	int fgetc(FILE *stream)	lecture d'un caractère depuis un fichier
fputc	int fputc(int c, FILE *stream)	écriture d'un caractère sur un fichier
getc	int getc(FILE *stream)	équivalent de fgetc mais implémenté par une macro
putc	int putc(int c, FILE *stream)	équivalent de fputc mais implémenté par une macro
getchar	int getchar(void)	lecture d'un caractère depuis l'entrée standard
putchar	int putchar(int c)	écriture d'un caractère sur la sortie standard
fgets	char *fgets(char *s, FILE *stream)	lecture d'une chaîne de caractères depuis un fichier
fputs	int *fputs(char *s, FILE *stream)	écriture d'une chaîne de caractères sur un fichier
gets	char *gets(char *s)	lecture d'une chaîne de caractères sur l'entrée standard
puts	int *puts(char *s)	écriture d'une chaîne de caractères sur la sortie standard

18.2 - Manipulation de caractères <ctype.h>

fonction	renvoie 1 si le caractère est
isalnum	une lettre ou un chiffre
isalpha	une lettre
iscntrl	un caractère de commande
isdigit	un chiffre décimal
isgraph	un caractère imprimable ou le blanc
islower	une lettre minuscule
isprint	un caractère imprimable (pas le blanc)
ispunct	un caractère imprimable qui n'est ni une lettre ni un chiffre
isspace	un blanc
isupper	une lettre majuscule
isxdigit	un chiffre hexadécimal

18.3 - Manipulation de chaînes de caractères <string.h>

fonction	prototype	action
strcpy	char *strcpy(char *ch1, char *ch2)	copie la chaîne ch2 dans la chaîne ch1 ; retourne ch1.
strncpy	char *strcpy(char *ch1, char *ch2, int n)	copie n caractères de la chaîne ch2 dans la chaîne ch1 ; retourne ch1.
strcat	char *strcat(char *ch1, char *ch2)	copie la chaîne ch2 à la fin de la chaîne ch1 ; retourne ch1.
strncat	char *strncat(char *ch1, char *ch2, int n)	copie n caractères de la chaîne ch2 à la fin de la chaîne ch1 ; retourne ch1.
strcmp	int strcmp(char *ch1, char *ch2)	compare ch1 et ch2 pour l'ordre lexicographique ; retourne une valeur négative si ch1 est inférieure à ch2, une valeur positive si ch1 est supérieure à ch2, 0 si elles sont identiques.
strncmp	int strcmp(char *ch1, char *ch2, int n)	compare les n premiers caractères de ch1 et ch2.
strchr	char *strchr(char *chaine, char c)	retourne un pointeur sur la première occurence de c dans chaine, et NULL si c n'y figure pas.
strrchr	char *strchr(char *chaine, char c)	retourne un pointeur sur la dernière occurence de c dans chaine, et NULL si c n'y figure pas.
strstr	char *strchr(char *ch1, char *ch2)	retourne un pointeur sur la première occurence de ch2 dans ch1, et NULL si ch2 n'y figure pas.
strlen	int strlen(char *chaine)	retourne la longueur de chaine.

18.4 - Fonctions mathématiques <math.h>

fonction	action
acos	arc cosinus
asin	arc sinus
atan	arc tangente
cos	cosinus
sin	sinus
tan	tangente
cosh	cosinus hyperbolique
sinh	cosinus hyperbolique
tanh	tangente hyperbolique
exp	exponentielle
log	logarithme népérien
log10	logarithme en base 10
pow	puissance
sqrt	racine carrée
fabs	valeur absolue
fmod	reste de la division
ceil	partie entière supérieure
floor	partie entière inférieure

18.5a - Utilitaires divers <stdlib.h>

Allocation dynamique	
fonction	**action**
calloc	allocation dynamique et initialisation à zéro.
malloc	allocation dynamique
realloc	modifie la taille d'une zone préalablement allouée par calloc ou malloc.
free	libère une zone mémoire

Conversion de chaînes de caractères en nombres		
fonction	**prototype**	**action**
atof	double atof(char *chaine)	convertit chaine en un double
atoi	int atoi(char *chaine)	convertit chaine en un int
atol	long atol(char *chaine)	convertit chaine en un long int

18.5b - Utilitaires divers <stdlib.h>

Génération de nombres pseudo-aléatoires		
fonction	**prototype**	**action**
rand	int rand(void)	fournit un nombre entier pseudo-aléatoire
srand	void srand(unsigned int germe)	modifie la valeur de l'initialisation du générateur pseudo-aléatoire utilisé par rand.

Arithmétique sur les entiers		
fonction	**prototype**	**action**
abs	int abs(int n)	valeur absolue d'un entier
labs	long labs(long n)	valeur absolue d'un long int
div	div_t div(int a, int b)	quotient et reste de la division euclidienne de a par b. Les résultats sont stockés dans les champs quot et rem (de type int) d'une structure de type div_t.
ldiv	ldiv_t ldiv(long a, long b)	quotient et reste de la division euclidienne de a par b. Les résultats sont stockés dans les champs quot et rem (de type long int) d'une structure de type ldiv_t.

18.5c - Utilitaires divers <stdlib.h>

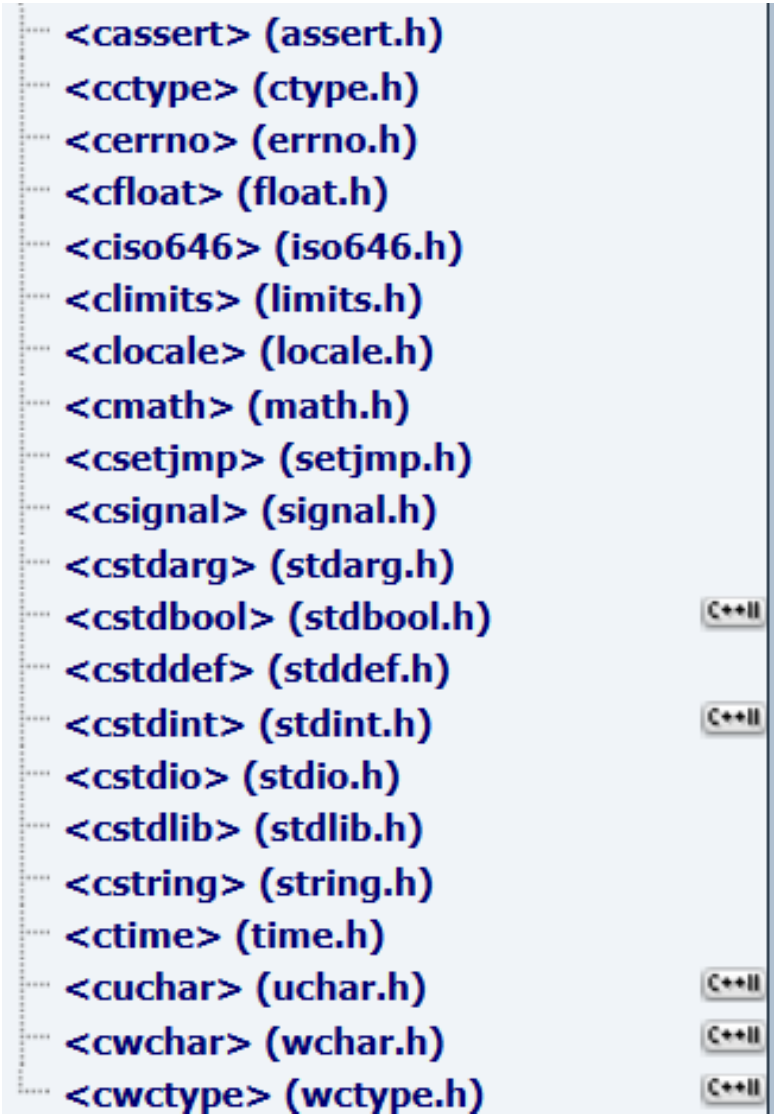
Recherche et tri	
fonction	**action**
qsort	permet de trier un tableau
bsearch	permet de rechercher un élément dans un tableau déjà trié

Communication avec l'environnement		
fonction	**prototype**	**action**
abort	void abort(void)	terminaison anormale du programme
exit	void exit(int etat)	terminaison du programme ; rend le contrôle au système en lui fournissant la valeur etat (la valeur 0 est considérée comme une fin normale).
system	int system(char *s)	exécution de la commande système définie par la chaîne de caractères s.

18.6 - Date et heure <time.h>

fonction	prototype	action
time	time_t time(time_t *tp)	retourne le nombre de secondes écoulées depuis le 1er janvier 1970, 0 heures G.M.T. La valeur retournée est assignée à *tp.
difftime	double difftime(time_t t1, time_t t2)	retourne la différence t1 - t2 en secondes.
ctime	char *ctime(time_t *tp)	convertit le temps système *tp en une chaîne de caractères explicitant la date et l'heure sous un format prédéterminé.
clock	clock_t clock(void)	retourne le temps CPU en microsecondes utilisé depuis le dernier appel à clock.

18.7a – C librairies



http://www.cplusplus.com/reference/new/

18.7b – C librairies : <cwctype> (wctype.h)

Functions

Character classification functions

They check whether the character passed as parameter belongs to a certain category:

iswalnum	Check if wide character is alphanumeric (function)
iswalpha	Check if wide character is alphabetic (function)
iswblank	Check if wide character is blank (function)
iswcntrl	Check if wide character is a control character (function)
iswdigit	Check if wide character is decimal digit (function)
iswgraph	Check if wide character has graphical representation (function)
iswlower	Check if wide character is lowercase letter (function)
iswprint	Check if wide character is printable (function)
iswpunct	Check if wide character is punctuation character (function)
iswspace	Check if wide character is a white-space (function)
iswupper	Check if wide character is uppercase letter (function)
iswxdigit	Check if wide character is hexadecimal digit (function)

http://www.cplusplus.com/reference/new/

18.7c – C librairies : <cwctype> (wctype.h)

Character conversion functions

Two functions that convert between letter cases:

towlower	Convert uppercase wide character to lowercase (function)
towupper	Convert lowercase wide character to uppercase (function)

Extensible classification/conversion functions

iswctype	Check if wide character has property (function)
towctrans	Convert using transformation (function)
wctrans	Return character transformation (function)
wctype	Return character property (function)

Types

wctrans_t	Wide character transformation (type)
wctype_t	Wide character type (type)
wint_t	Wide character integral type (type)

Constants

WEOF	Wide End-of-File (constant)

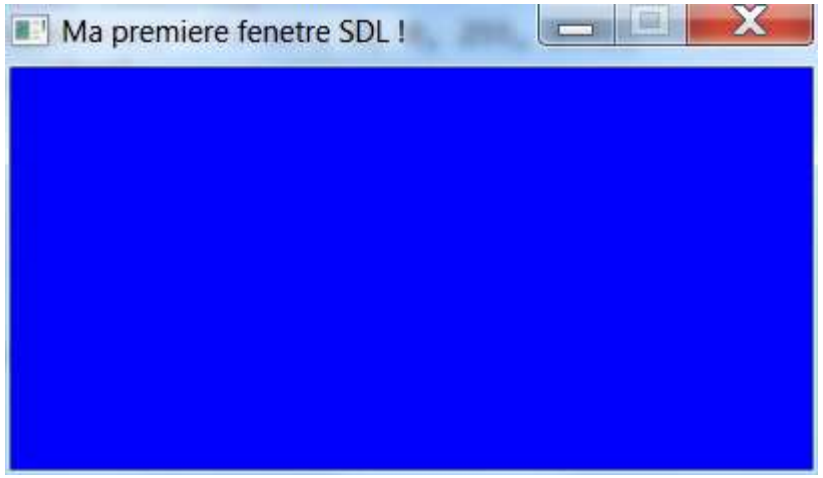
http://www.cplusplus.com/reference/new/

5 - Environnement Graphique : SDL

- Premier programme graphique

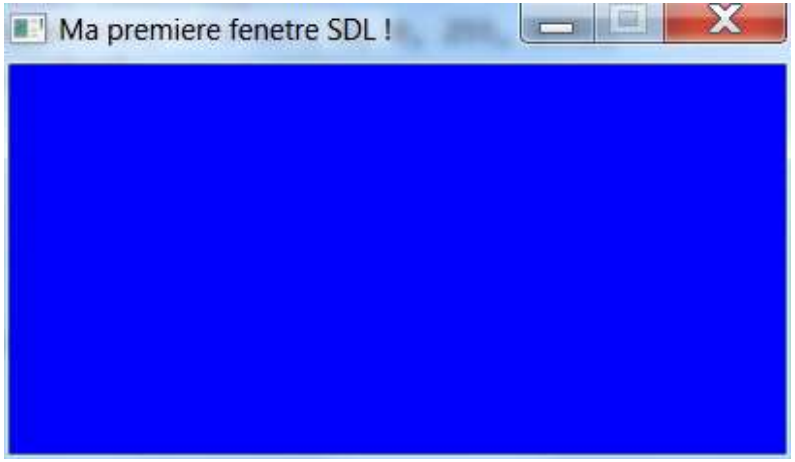
5.1a – Premier programme graphique

```c
int main(int argc, char* args[]) {
    SDL_Event evt;
    SDL_Window* fenetre = NULL;
    SDL_Surface* surfaceEcran = NULL;

    if (!SDL_creerWindow(&fenetre, &surfaceEcran)) {
        printf("Erreur de creation!\n");
    } else {
        SDL_UpdateWindowSurface(fenetre);
        while (1) {
            //Handle events on queue
            if (SDL_PollEvent(&evt) != 0) {
                //User requests quit
                if (evt.type == SDL_QUIT)
                    break;
            }
        }
    }
    SDL_libererMemoire(&fenetre);
    return 0;
}
```

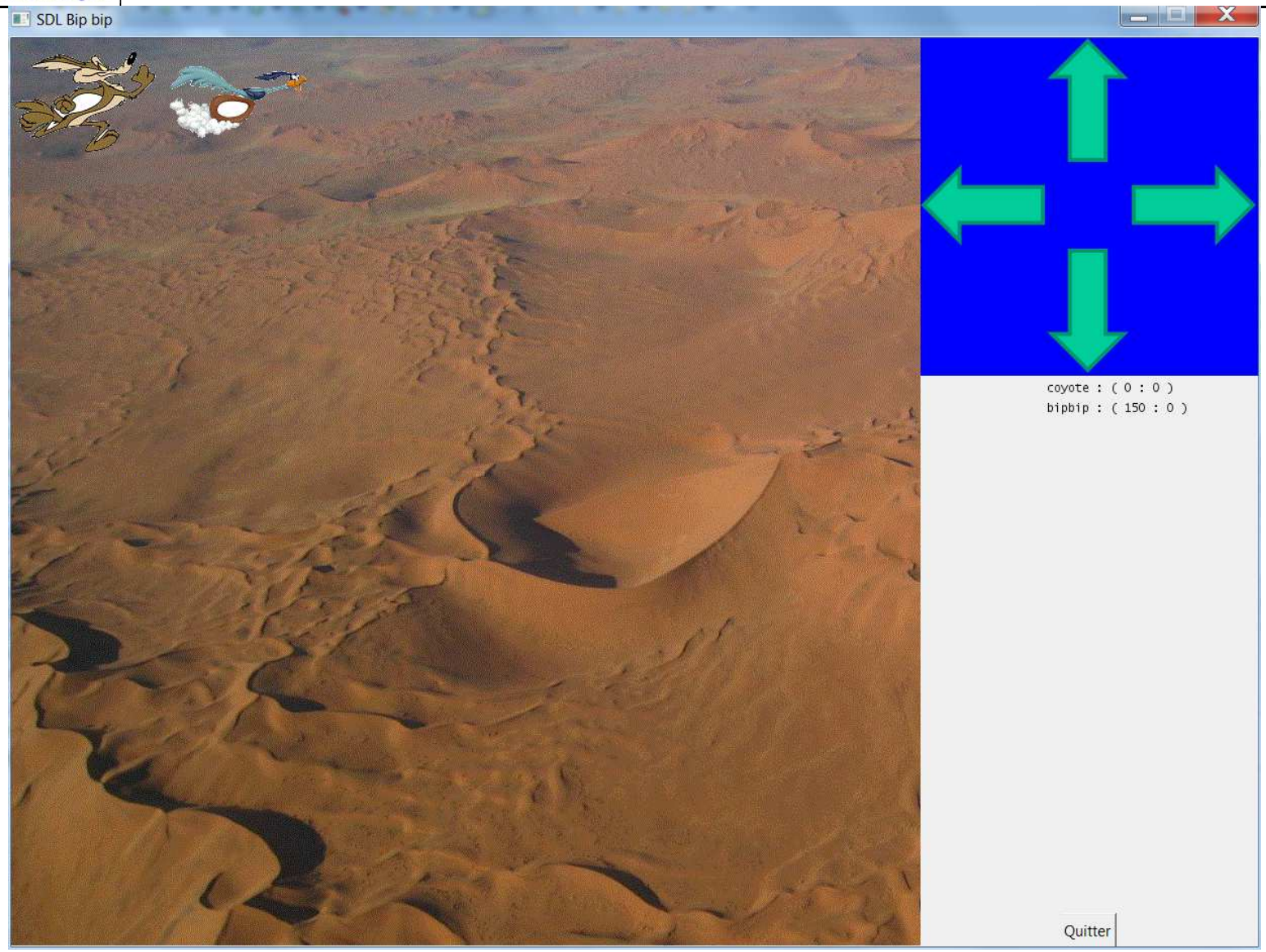


5.1b - Premier programme graphique

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

bool SDL_creerWindow(SDL_Window** fenetre, SDL_Surface** surfaceEcran) {
    if (SDL_Init( SDL_INIT_VIDEO) < 0) {
        printf("SDL_Error: %s\n", SDL_GetError());
        return false;
    } else {
        *fenetre = SDL_CreateWindow("Ma premiere fenetre SDL !",SDL_WINDOWPOS_CENTERED,SDL_WINDOWPOS_CENTERED, 400, 200, 0);
        if (fenetre == NULL) {
            printf("SDL_Error: %s\n", SDL_GetError());
            return false;
        } else {
            *surfaceEcran = SDL_GetWindowSurface(*fenetre);
            Uint32 color = SDL_MapRGBA((*surfaceEcran)->format, 0, 0, 255, 255);
            SDL_FillRect(*surfaceEcran, NULL, color);
            return true;
        }
    }
    return false;
}

void SDL_libererMemoire(SDL_Window** fenetre) {
    SDL_DestroyWindow(*fenetre);
    *fenetre = NULL;
}
```

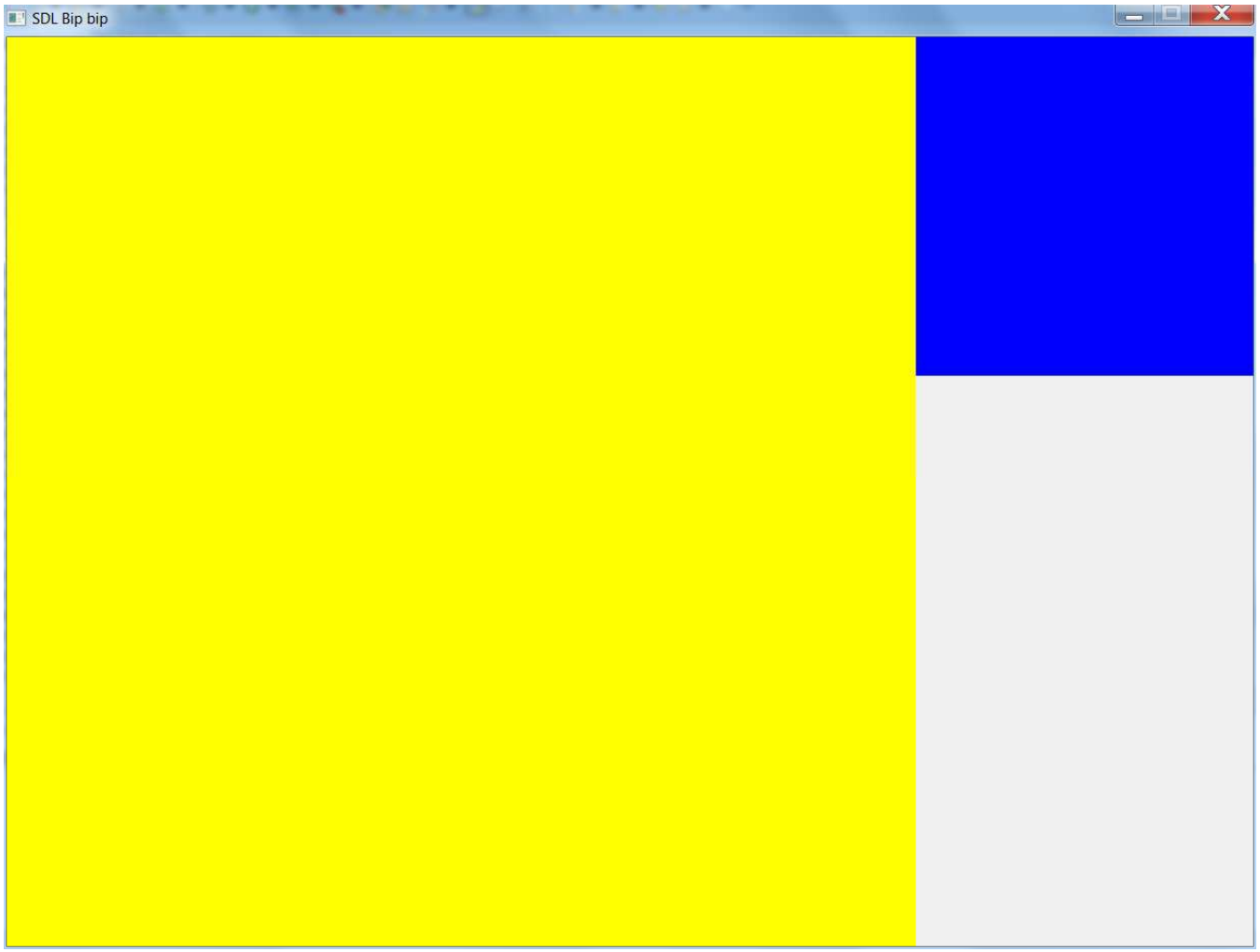


5.2 - Exemple : Bip Bip et Coyote



Cadre





Cadre



```c
int main(int argc, char* args[]) {
    BOOL quit = FALSE;
    SDL_Event evt;
    SDL_Window* fenetre = NULL;
    SDL_Surface* surfaceEcran = NULL;

    if (!SDL_creerWindow(&fenetre, &surfaceEcran)) {
        printf("Erreur d'initialisation!\n");
    } else {
        //Load media
        if (!SDL_Initialisation_Media(fenetre, surfaceEcran)) {
            printf("Erreur de lecture des media !\n");
        } else {
            SDL_UpdateWindowSurface(fenetre);
            while (!quit) {
                //Handle events on queue
                while (SDL_PollEvent(&evt) != 0) {
                    quit = SDL_evenenement(&evt);
                }
            }
        }
    }
    SDL_libererMemoire(&fenetre);
    return 0;
}
```

Cadre



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

#define TRUE 1
#define FALSE 0
typedef short unsigned int BOOL;

#define ERREUR_STL() fprintf(stderr,"SDL_Error: %s\n", SDL_GetError());exit(EXIT_FAILURE);
#define grisclaire SDL_MapRGBA((*surfaceEcran)->format, 240, 240, 240,255)
#define bleue SDL_MapRGBA((*surfaceEcran)->format, 0, 0, 255,255)
#define jaune SDL_MapRGBA((*surfaceEcran)->format, 255, 255, 0,255)

const int TAILLE = 150;
const int RMAX = 6;
const int TMAX = TAILLE * RMAX;
const int SCREEN_WIDTH = TMAX + 125 * 2 + 84;
const int SCREEN_HEIGHT = TMAX;

int xb = TAILLE;
int yb = 0;
int xc = 0;
int yc = 0;
```

Cadre



```
BOOL SDL_creerWindow(SDL_Window**fenetre, SDL_Surface** surfaceEcran) {
    BOOL success = TRUE;
    SDL_Rect r;

    if (SDL_Init( SDL_INIT_VIDEO | SDL_INIT_AUDIO) < 0) {
        ERREUR_STL();
    } else {
        *fenetre = SDL_CreateWindow("SDL Bip bip", SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,SCREEN_WIDTH, SCREEN_HEIGHT,0 );
        if (*fenetre == NULL) {
            ERREUR_STL();
        } else {
            *surfaceEcran = SDL_GetWindowSurface(*fenetre);

            SDL_FillRect(*surfaceEcran, NULL, grisclaire);
            r.x = TMAX;
            r.y = 0;
            r.w = 125 * 2 + 85;
            r.h = 125 * 2 + 85;
            SDL_FillRect((*surfaceEcran), &r, bleue);
            r.x = 0;
            r.y = 0;
            r.w = TMAX;
            r.h = TMAX;
            SDL_FillRect((*surfaceEcran), &r, jaune);

        }
    }
    return success;
}
```

Cadre



```
BOOL SDL_Initialisation_Media(SDL_Window*fenetre, SDL_Surface* surfaceEcran) {
    BOOL success = TRUE;
    return success;
}
```

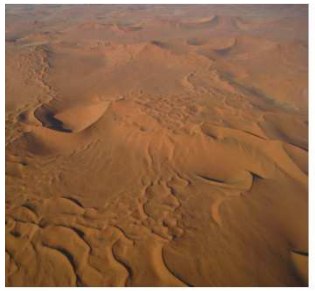
```
void SDL_libererMemoire(SDL_Window** fenetre) {
    SDL_DestroyWindow(*fenetre);
    *fenetre = NULL;
}

BOOL SDL_evenenement(SDL_Event* evt) {
    if (evt->type == SDL_QUIT)
        return TRUE;
    return FALSE;
}
```

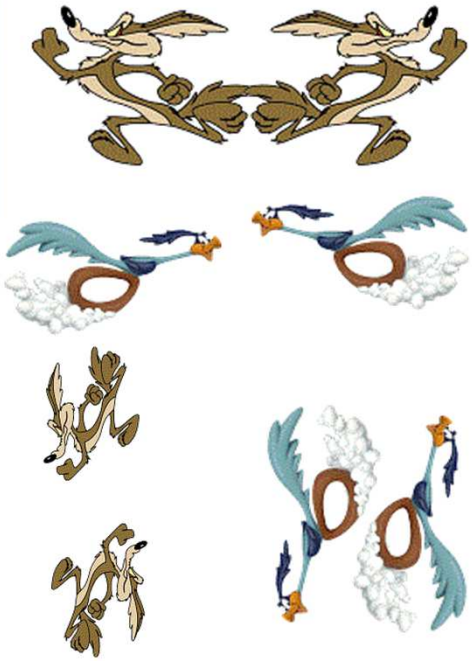
Définition des images



```
BOOL SDL_Initialisation_Media(SDL_Window*fenetre, SDL_Surface* surfaceEcran) {
    BOOL success = TRUE;
    SDL_Rect dstrect;
    fdesert = IMG_Load("C:\\STL_img\\desert900.png");
    fcoyote_droite = IMG_Load("C:\\STL_img\\coyote_min_droite.png");
    fbipbip_droite = IMG_Load("C:\\STL_img\\bipbip_min_droite.png");
    fcoyote_gauche = IMG_Load("C:\\STL_img\\coyote_min_gauche.png");
    fbipbip_gauche = IMG_Load("C:\\STL_img\\bipbip_min_gauche.png");
    fcoyote_haut = IMG_Load("C:\\STL_img\\coyote_min_haut.png");
    fbipbip_haut = IMG_Load("C:\\STL_img\\bipbip_min_haut.png");
    fcoyote_bas = IMG_Load("C:\\STL_img\\coyote_min_bas.png");
    fbipbip_bas = IMG_Load("C:\\STL_img\\bipbip_min_bas.png");
    fbgauche = IMG_Load("C:\\STL_img\\gauche.png");
    fbdroite = IMG_Load("C:\\STL_img\\droite.png");
    fbhaut = IMG_Load("C:\\STL_img\\haut.png");
    fbbas = IMG_Load("C:\\STL_img\\bas.png");
    if (fdesert == NULL || fcoyote_droite == NULL || fbipbip_droite == NULL
            || fcoyote_gauche == NULL || fbipbip_gauche == NULL
            || fcoyote_haut == NULL || fbipbip_haut == NULL
            || fcoyote_bas == NULL || fbipbip_bas == NULL || fbgauche == NULL
            || fbdroite == NULL || fbhaut == NULL || fbbas == NULL) {
        ERREUR_STL()
    }
    SDL_BlitSurface(fdesert, NULL, surfaceEcran, NULL);
    dstrect.x = xc;
    dstrect.y = yc;
    dstrect.w = 150;
    dstrect.h = 120;
    SDL_BlitSurface(fcoyote_droite, NULL, surfaceEcran,&dstrect);

    dstrect.x = xb;
    dstrect.y = yb;
    dstrect.w = 150;
    dstrect.h = 120;
    SDL_BlitSurface(fbipbip_droite, NULL, surfaceEcran,&dstrect);
    return success;
}
```

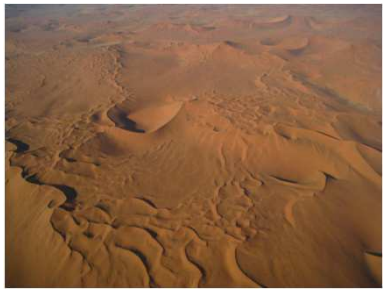
```
#include <SDL_image.h>
```

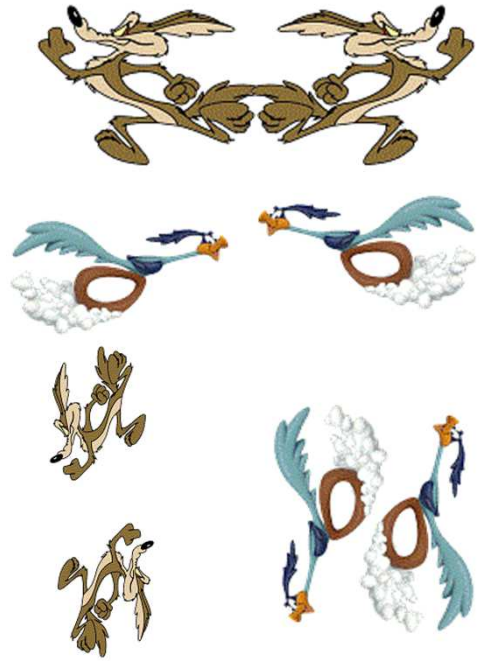




Définition des images


```
SDL_Surface* fdesert = NULL;
SDL_Surface* fcoyote_droite = NULL;
SDL_Surface* fbipbip_droite = NULL;
SDL_Surface* fcoyote_gauche = NULL;
SDL_Surface* fbipbip_gauche = NULL;
SDL_Surface* fcoyote_haut = NULL;
SDL_Surface* fbipbip_haut = NULL;
SDL_Surface* fcoyote_bas = NULL;
SDL_Surface* fbipbip_bas = NULL;
SDL_Surface* fbgauche = NULL;
SDL_Surface* fbdroite = NULL;
SDL_Surface* fbhaut = NULL;
SDL_Surface* fbbas = NULL;
```





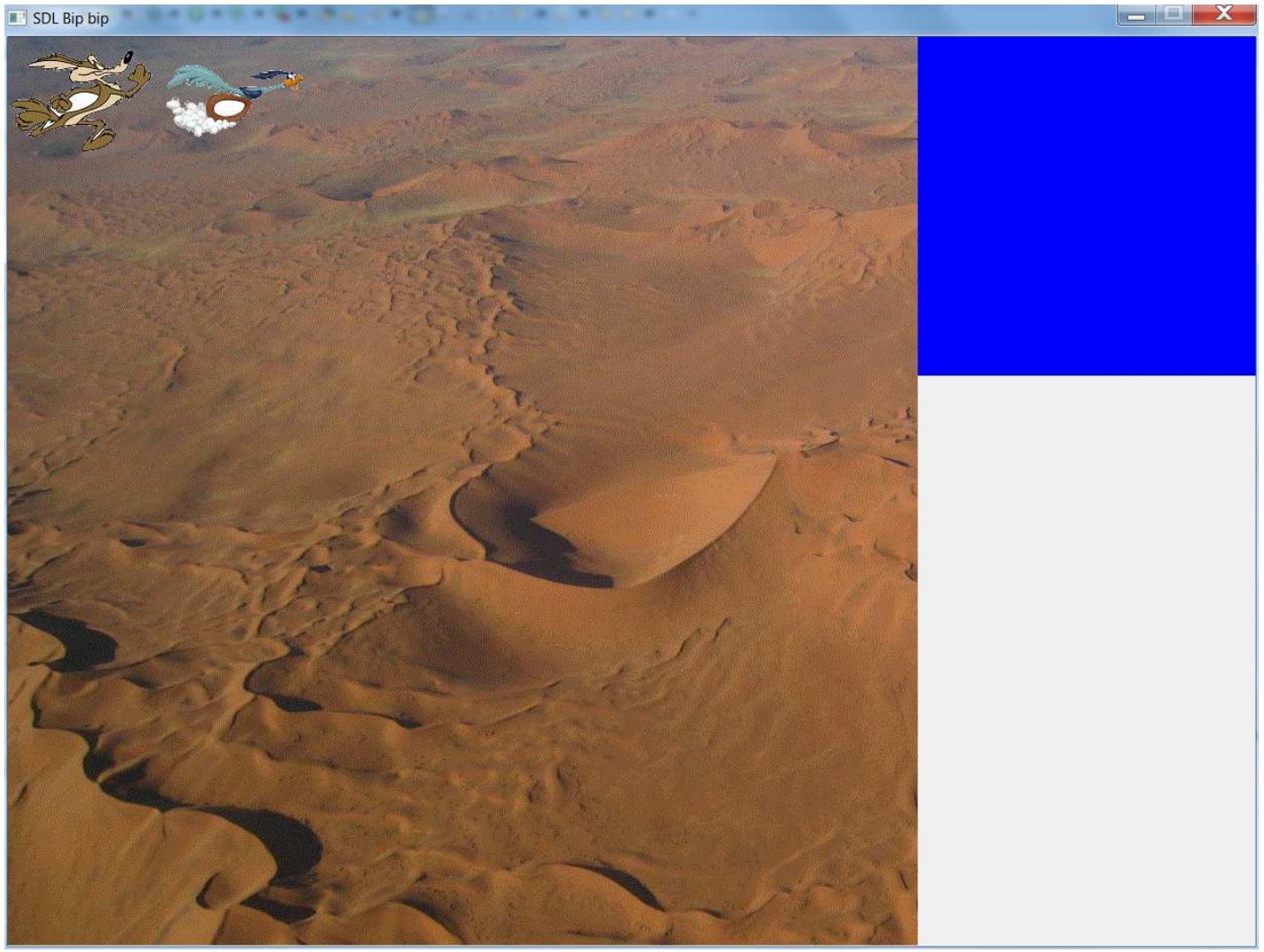
Définition des images



```
void SDL_libererIMG() {
    SDL_FreeSurface(fdesert);
    SDL_FreeSurface(fcoyote_droite);
    SDL_FreeSurface(fbipbip_droite);
    SDL_FreeSurface(fcoyote_gauche);
    SDL_FreeSurface(fbipbip_gauche);
    SDL_FreeSurface(fcoyote_haut);
    SDL_FreeSurface(fbipbip_haut);
    SDL_FreeSurface(fcoyote_bas);
    SDL_FreeSurface(fbipbip_bas);
    SDL_FreeSurface(fbgauche);
    SDL_FreeSurface(fbdroite);
    SDL_FreeSurface(fbhaut);
    SDL_FreeSurface(fbbas);

    fdesert = NULL;
    fcoyote_droite = NULL;
    fbipbip_droite = NULL;
    fcoyote_gauche = NULL;
    fbipbip_gauche = NULL;
    fcoyote_haut = NULL;
    fbipbip_haut = NULL;
    fcoyote_bas = NULL;
    fbipbip_bas = NULL;
    fbgauche = NULL;
    fbdroite = NULL;
    fbhaut = NULL;
    fbbas = NULL;

}

void SDL_libererMemoire(SDL_Window** fenetre) {

    SDL_libererIMG();

    SDL_DestroyWindow(*fenetre);
    *fenetre = NULL;

    IMG_Quit();

}
```

Définition des images

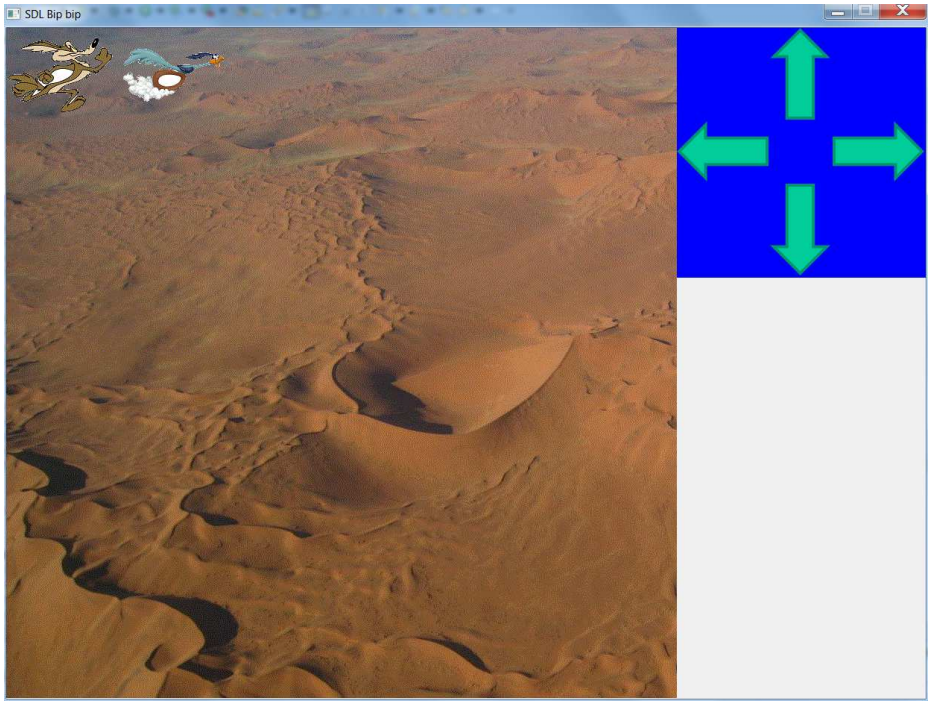




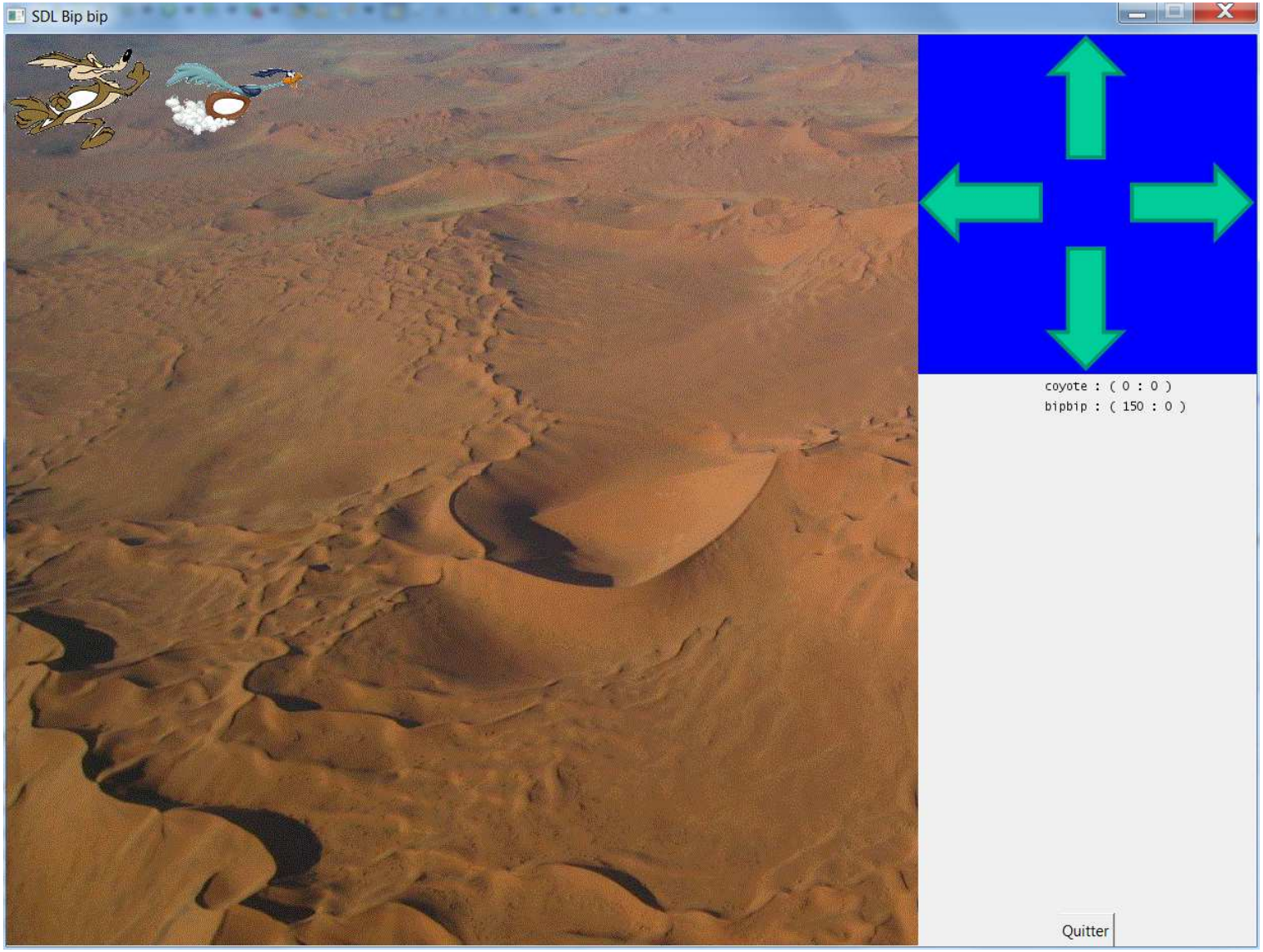
Insérer définition des boutons



```
dstrect.x = xc;
dstrect.y = yc;
dstrect.w = 150;
dstrect.h = 120;
SDL_BlitSurface(fcoyote_droite, NULL, surfaceEcran,&dstrect);

dstrect.x = xb;
dstrect.y = yb;
dstrect.w = 150;
dstrect.h = 120;
SDL_BlitSurface(fbipbip_droite, NULL, surfaceEcran,&dstrect);

dstrect.x = TMAX + 125;
dstrect.y = 0;
dstrect.w = 83;
dstrect.h = 125;
SDL_BlitSurface(fbhaut, NULL, surfaceEcran, &dstrect);

dstrect.x = TMAX + 125;
dstrect.y = 125 + 84;
dstrect.w = 83;
dstrect.h = 125;
SDL_BlitSurface(fbbas, NULL, surfaceEcran, &dstrect);


dstrect.x = TMAX;
dstrect.y = 125;
dstrect.w = 125;
dstrect.h = 83;
SDL_BlitSurface(fbgauche, NULL, surfaceEcran, &dstrect);

dstrect.x = TMAX + 125 + 84;
dstrect.y = 125;
dstrect.w = 125;
dstrect.h = 83;
SDL_BlitSurface(fbdroite, NULL, surfaceEcran, &dstrect);
```



Insérer définition des boutons



Insérer définition des boutons



```
#include <SDL_ttf.h>
BOOL SDL_creerWindow(SDL_Window**fenetre, SDL_Surface** surfaceEcran) {
        if(TTF_Init()  < 0 ) { ERREUR_STL(); }
 ..................
TTF_Font *police = NULL ;

BOOL SDL_Initialisation_Media(SDL_Window*fenetre, SDL_Surface* surfaceEcran) {
    BOOL success = TRUE;
    SDL_Rect dstrect;
    SDL_Surface *texte=NULL;
    SDL_Color couleurNoire = {0, 0, 0};
    static char buffer [50];
 ..................
    sprintf(buffer,"coyote : ( %d : %d )",xc,yc);
    texte = TTF_RenderText_Blended(police,buffer, couleurNoire);
    dstrect.x =  TMAX + 125;
    dstrect.y =  125 * 2 + 84 + 5;
    SDL_BlitSurface(texte, NULL, surfaceEcran, &dstrect);
    sprintf(buffer,"bipbip : ( %d : %d )",xb,yb);
    dstrect.x =  TMAX + 125;
    dstrect.y =  125 * 2 + 84 + 5+20;
    texte = TTF_RenderText_Blended(police,buffer, couleurNoire);
    SDL_BlitSurface(texte, NULL, surfaceEcran, &dstrect);
 ..................
    SDL_Surface* fbquit = IMG_Load("C:\\STL_img\\bouton_quitter.png");
    dstrect.x = TMAX + 125 + 84 / 2 - 28;
    dstrect.y = TMAX - 33;
    SDL_BlitSurface(fbquit, NULL, surfaceEcran, &dstrect);
```

Insérer définition des boutons



```
void SDL_libererMemoire(SDL_Window** fenetre) {

    SDL_libererIMG();

    SDL_DestroyWindow(*fenetre);
    *fenetre = NULL;

    TTF_CloseFont(police);
    TTF_Quit();
    IMG_Quit();
    SDL_Quit();
}
```

Insérer définition des boutons



```
int main(int argc, char* args[]) {
    BOOL quit = FALSE;
    SDL_Event evt;
    SDL_Window* fenetre = NULL;
    SDL_Surface* surfaceEcran = NULL;

    if (!SDL_creerWindow(&fenetre, &surfaceEcran)) {
        printf("Erreur d'initialisation!\n");
    } else {
        //Load media
        if (!SDL_Initialisation_Media(fenetre, surfaceEcran)) {
            printf("Erreur de lecture des media !\n");
        } else {
            SDL_UpdateWindowSurface(fenetre);
            while (!quit) {
                //Handle events on queue
                while (SDL_PollEvent(&evt) != 0) {
                    quit = SDL_evenenement(&evt,fenetre,surfaceEcran);
                }
            }
        }
    }
    SDL_libererMemoire(&fenetre);
    return 0;
}
```

Gestion évènement : souris, clavier



```
bool testRect(int x, int y, SDL_Rect* dstrect) {
    return (x > dstrect->x) && (x < dstrect->x + dstrect->w) && (y > dstrect->y)
            && (y < dstrect->y + dstrect->h);
}
```

```
BOOL SDL_evenenement(SDL_Event* evt, SDL_Window*fenetre,SDL_Surface* surfaceEcran) {
    SDL_Rect dstrecthaut;
    SDL_Rect dstrectbas;
    SDL_Rect dstrectgauche;
    SDL_Rect dstrectdroite;
    SDL_Rect dstrectquit;
    dstrecthaut.x = TMAX + 125;dstrecthaut.y = 0;
    dstrecthaut.w = 83;dstrecthaut.h = 125;
    dstrectbas.x = TMAX + 125;dstrectbas.y = 125 + 84;
    dstrectbas.w = 83;dstrectbas.h = 125;
    dstrectgauche.x = TMAX;dstrectgauche.y = 125;
    dstrectgauche.w = 125;dstrectgauche.h = 83;
    dstrectdroite.x = TMAX + 125 + 84;dstrectdroite.y = 125;
    dstrectdroite.w = 125;dstrectdroite.h = 83;
    dstrectquit.x = TMAX + 125 + 84 / 2 - 28;dstrectquit.y = TMAX - 33;
    dstrectquit.h = 33; dstrectquit.w = 56;
```

Gestion évènement : souris, clavier



```
if (evt->type == SDL_QUIT) return TRUE;
else if (evt->type == SDL_MOUSEBUTTONDOWN) {
    do {
        if (testRect(evt->motion.x, evt->motion.y, &dstrecthaut)) {
            event_haut(fenetre, surfaceEcran);
        } else if (testRect(evt->motion.x, evt->motion.y, &dstrectbas)) {
            event_bas(fenetre, surfaceEcran);
        } else if (testRect(evt->motion.x, evt->motion.y, &dstrectgauche)) {
            event_gauche(fenetre, surfaceEcran);
        } else if (testRect(evt->motion.x, evt->motion.y, &dstrectdroite)) {
            event_droite(fenetre, surfaceEcran);
        } else if (testRect(evt->motion.x, evt->motion.y, &dstrectquit)) {
            return TRUE;
            break;
        } else
            break;
        SDL_PollEvent (evt);
        if (evt->type == SDL_MOUSEBUTTONUP) break;
    } while (true);

} else if (evt->type == SDL_KEYDOWN) {
```

Gestion évènement : souris, clavier



```
    } else if (evt->type == SDL_KEYDOWN) {
        do {
            switch (evt->key.keysym.sym) {
            case SDLK_UP:
                event_haut(fenetre, surfaceEcran);
                break;
            case SDLK_DOWN:
                event_bas(fenetre, surfaceEcran);
                break;
            case SDLK_LEFT:
                event_gauche(fenetre, surfaceEcran);
                break;
            case SDLK_RIGHT:
                event_droite(fenetre, surfaceEcran);
                break;
            default:
                break;
            }
            SDL_PollEvent(evt);
            if (evt->type == SDL_KEYUP) break;
        } while (true);
    }
    return FALSE;
}
```

Gestion évènement : souris, clavier



```
void event_droite(SDL_Window* screen, SDL_Surface* screenSurface) {
    SDL_Color couleurNoire = { 0, 0, 0 };
    SDL_Surface *texte = NULL;
    SDL_Rect dstrect;
    static char buffer[50];
    droite();

    SDL_BlitSurface(fdesert, NULL, screenSurface, NULL);
    dstrect.x = xc;dstrect.y = yc;
    dstrect.w = 150;dstrect.h = 120;

    SDL_BlitSurface(fcoyote_droite, NULL, screenSurface, &dstrect);
    dstrect.x = xb;dstrect.y = yb;
    dstrect.w = 150;dstrect.h = 120;

    SDL_BlitSurface(fbipbip_droite, NULL, screenSurface, &dstrect);
    dstrect.x = TMAX;dstrect.y = 125 * 2 + 85;
    dstrect.w = 125 * 2 + 85;dstrect.h = 50;

    SDL_FillRect(screenSurface, &dstrect, bleue(screenSurface));
    sprintf(buffer,"coyote : ( %d : %d )",xc,yc);
    texte = TTF_RenderText_Blended(police,buffer, couleurNoire);
    dstrect.x =  TMAX + 125;dstrect.y =  125 * 2 + 84 + 5;
    SDL_BlitSurface(texte, NULL, screenSurface, &dstrect);
    sprintf(buffer,"bipbip : ( %d : %d )",xb,yb);
    dstrect.x =  TMAX + 125;dstrect.y =  125 * 2 + 84 + 5+20;
    texte = TTF_RenderText_Blended(police,buffer, couleurNoire);
    SDL_BlitSurface(texte, NULL, screenSurface, &dstrect);
    SDL_UpdateWindowSurface(screen);
}
```

Gestion évènement : souris, clavier

```
void droite() {
    xb = xb + TAILLE * 2;
    xc = xc + TAILLE;
    if (xb == xc and yb == yc)
        xb = xb + TAILLE;
    if (xb >= TMAX)
        xb = 0;
    if (xc >= TMAX)
        xc = 0;
    if (xb == xc and yb == yc)
        xb = xb + TAILLE;
}
```

Son



```
#include <SDL_mixer.h>
```

```
Mix_Chunk *son = NULL;

BOOL SDL_Initialisation_Media(SDL_Window*fenetre, SDL_Surface* surfaceEcran) {
    BOOL success = TRUE;
    SDL_Rect dstrect;
    SDL_Surface *texte = NULL;
    SDL_Color couleurNoire = { 0, 0, 0 };
    SDL_Surface* fbquit = NULL;
    static char buffer[50];

    if (Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 2048) < 0) {
            fprintf(stderr,"SDL_mixer Error: %s\n", Mix_GetError());
            return FALSE;
        }
        son = Mix_LoadWAV("C:\\STL_img\\4074.wav");
        if (son == NULL) {
            fprintf(stderr,"Failed to load beat music! SDL_mixer Error: %s\n",Mix_GetError());
            return FALSE;
        }
```

Son



```
void event_droite(SDL_Window* screen, SDL_Surface* screenSurface) {
    SDL_Color couleurNoire = { 0, 0, 0 };
    SDL_Surface *texte = NULL;
    SDL_Rect dstrect;
    static char buffer[50];
    Mix_PlayChannel(-1, son, 0);
    droite();
     ..................

 void SDL_libererMemoire(SDL_Window** fenetre) {

     SDL_libererIMG();

     SDL_DestroyWindow(*fenetre);
     *fenetre = NULL;

     Mix_FreeChunk(son);
     Mix_Quit();

     TTF_CloseFont(police);
     TTF_Quit();
     IMG_Quit();
     SDL_Quit();
 }
```

SDL : site

https://www.libsdl.org/

Get the current **stable** SDL version 2.0.3

SDL
Simple Directmedia Layer

Main
- About
- Bugs
- Licensing
- Credits
- Feedback

Documentation
- Wiki
- Forums
- Mailing Lists

Download
- SDL 2.0
- SDL 1.2
- SDL Mercurial
- Bindings

About SDL

Simple DirectMedia Layer is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D. It is used by video playback software, emulators, and popular games including Valve's award winning catalog and many Humble Bundle games.

SDL officially supports Windows, Mac OS X, Linux, iOS, and Android. Support for other platforms may be found in the source code.

SDL is written in C, works natively with C++, and there are bindings available for several other languages, including C# and Python.

SDL 2.0 is distributed under the zlib license. This license allows you to use SDL freely in any software.

Made with SDL: Musaic Box

SDL : Lien

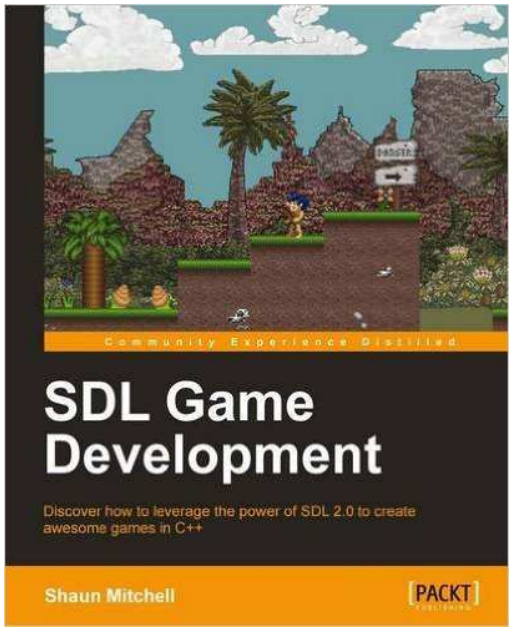
http://lazyfoo.net/tutorials/SDL

http://prod.openclassrooms.com/forum/sujet/creation-de-projet-sdl-en-c

http://loka.developpez.com/tutoriel/sdl/installation/eclipse/

http://jeux.developpez.com/tutoriels/sdl-2/guide-migration/

SDL : Bibliographie



www.yantra-technologies.com

Le Langage C modulaire

carole.grondein@yantra-technologies.com
david.palermo@yantra-technologies.com

Sommaire

1 - Définition

La programmation modulaire sert à décomposer une grosse application en modules, groupes de fonctions, de méthodes et de traitement, pour pouvoir les développer indépendamment, et les réutiliser dans d'autres applications.

Le principe est tout bête : plutôt que de placer tout le code de notre programme dans un seul fichier (main.c), nous le « séparons » en plusieurs petits fichiers.

http://fr.wikipedia.org/wiki/Programmation_modulaire
http://www.siteduzero.com/informatique/tutoriels/apprenez-a-programmer-en-c/la-programmation-modulaire

2 - Pourquoi ?

- La programmation est modulaire, donc plus compréhensible
- La séparation en plusieurs fichiers produit des listings plus lisibles
- La maintenance est plus facile car seule une partie du code est recompilée

=> La programmation modulaire simplifie l'ensemble d'un programme, facilite les futurs modifications, évolutions et réutilisations.

3- Le prototype

Le prototype d'une fonction (d'une procédure ou d'une méthode) désigne la syntaxe d'appel de cette fonction. Il spécifie ce que la fonction fait mais ne donne pas de détails sur la façon dont elle le fait.

```
// structure nombre complexe
typedef struct complexe {
    double a; // partie reel
    double b; // partie imaginaire
} Complexe;

Complexe* complexe_creer  (const double a ,const double b );
Complexe* complexe_copier (const Complexe* const Y);
Complexe* complexe_default();
void complexe_liberer     (Complexe** X);

void complexe_afficher(const Complexe* const X);

Complexe* complexe_multiplier (Complexe* X, const Complexe* const Y );
Complexe* complexe_diviser    (Complexe* X, const Complexe* const Y );
Complexe* complexe_additionner(Complexe* X, const Complexe* const Y );
Complexe* complexe_soustraire (Complexe* X, const Complexe* const Y );

int complexe_isegale    (const Complexe* const X, const Complexe* const Y );
int complexe_isdifferent(const Complexe* const X, const Complexe* const Y );
```

4 - Le header

Les fichiers header contiennent les prototypes de fonction.

complexe.h

```
#ifndef COMPLEXE_H
#define COMPLEXE_H

// structure nombre complexe
typedef struct complexe {
    double a; // partie reel
    double b; // partie imaginaire
} Complexe;

Complexe* complexe_creer  (const double a ,const double b );
Complexe* complexe_copier (const Complexe* const Y);
Complexe* complexe_default();
void complexe_liberer     (Complexe** X);

void complexe_afficher(const Complexe* const X);

Complexe* complexe_multiplier (Complexe* X, const Complexe* const Y );
Complexe* complexe_diviser    (Complexe* X, const Complexe* const Y );
Complexe* complexe_additionner(Complexe* X, const Complexe* const Y );
Complexe* complexe_soustraire (Complexe* X, const Complexe* const Y );

int complexe_isegale    (const Complexe* const X, const Complexe* const Y );
int complexe_isdifferent(const Complexe* const X, const Complexe* const Y );

#endif // COMPLEXE_H
```

5 - Le corps

Le corps de la fonction contient le code de la fonction

complexe.c

```
#include "complexe.h"
#include <malloc.h>

Complexe* complexe_creer  (const double a ,const double b ) {
    Complexe *res = (Complexe*) malloc(sizeof(Complexe));
    if ( res == NULL ) exit(10);
    res->a=a;
    res->b=b;
    return res;
}

Complexe* complexe_copier (const Complexe* Y) {
    if ( Y == NULL ) return NULL;
    return complexe_creer(Y->a,Y->b);
}

Complexe* complexe_default() {
    return complexe_creer(0,0);
}
```

6 - L'utilisation

On peut écrire dès la partie création des headers, le programme utilisateur et compiler, mais on ne peut faire un exécutable que si le corps des fonctions a été écrit.

main,c

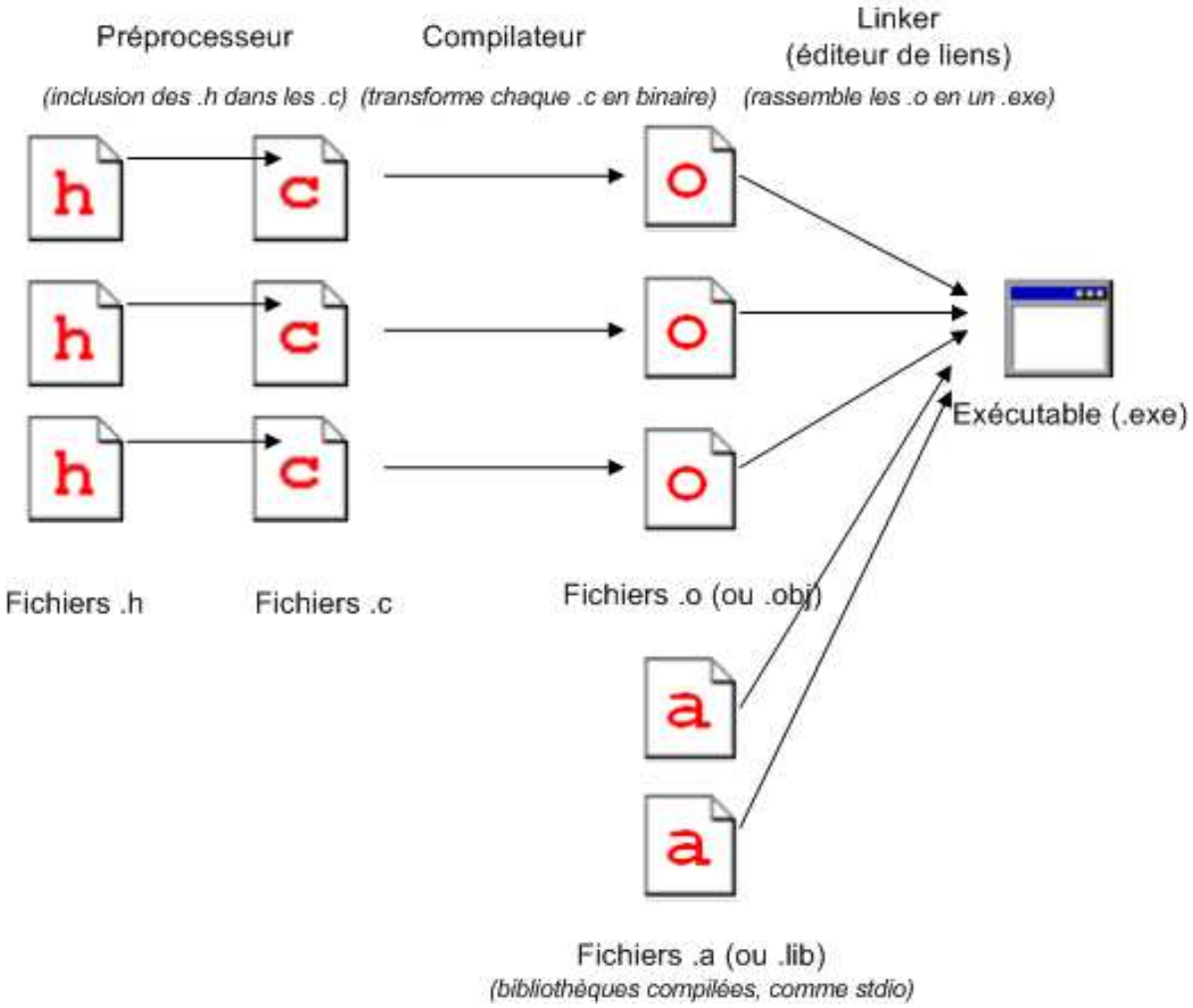
```c
#include <stdio.h>
#include <stdlib.h>
#include "complexe.h"

int main()
{
    Complexe* C1 = complexe_creer(1,1);
    Complexe* C2 = complexe_creer(2,2);
    printf("C1=");complexe_afficher(C1);
    printf("C2=");complexe_afficher(C2);
    complexe_additionner(C1,C2);
    printf("C1=");complexe_afficher(C1);
    printf("C2=");complexe_afficher(C2);
    return 0;
}
```

7 - La compilation séparée

La programmation modulaire entraîne la compilation séparée.

- Étape 1 : Le programmeur sépare ses fonctions dans des fichiers distincts regroupés par thème
- Exemple : complexe.h et complexe.c
- Étape 2 : création d'un librairie
- Étape 3 : Edition de liens (linker) qui a pour rôle de regrouper toutes les fonctions utilisées dans un même exécutable.

7 - La compilation séparée



7 - La compilation séparée

#création des fichiers binaires à mettre dans la libraire

- gcc –fpic –o complexe.o –Wall complexe.c

#création de la libraire statique

- ar –cr libcomplexe.a complexe.o
- ranlib libcomplexe.a

#création de la libraire dynamique

- gcc -o libcomplexe.so –shared complexe.o

#création du fichier contenant le programme principal

- gcc –fpic –o main.o –Wall main.c

#création exécutable dynamique

- gcc –o testcomplexe.dyn main.o –L. –lcomplexe

#création exécutable statique

- gcc -static –o testcomplexe.sta main.o –L. –lcomplexe

7 - La compilation séparée

- make
- automake
- cmake : http://www.cmake.org/
- qmake

8 - Répertoire projet

Projet :

- inc
- src
- obj
- lib
- exe
- test
- test_unitaire
- doc
- outils

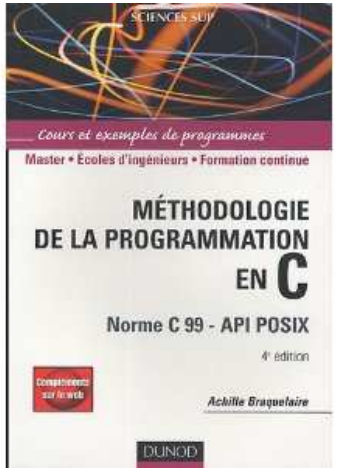
8 - Répertoire projet

Projet :

- inc
- src
- release
  - obj
  - lib
  - exe
- debug
  - obj
  - lib
  - exe
- test
- test_unitaire
- doc
- outils

5 - Bibliographie

- **Le langage C : Norme ANSI**
  - **Brian-W Kernighan et Dennis-M Ritchie**
- **Langage C**
  - **Claude Delannoy**
- **Méthodologie de la programmation en C : Norme C 99 - API POSIX**
  - **Achille Braquelaire**



5 - Bibliographie

Quelques liens

- http://gcc.gnu.org/
- http://c.developpez.com/
- http://www.cplusplus.com/reference/clibrary/
- http://www-roc.inria.fr/secret/Anne.Canteaut/COURS_C/
- http://www.inf.enst.fr/~charon/CFacile/
- http://www.ltam.lu/Tutoriel_Ansi_C/
- http://www.siteduzero.com/tutoriel-3-14189-apprenez-a-programmer-en-c.html