

SL121_LOGA

Programmation en C

Théorie

Chapitre 7

Les fonctions

Christian HUBER (CHR)
Version 1.0 Octobre 2004
Version 1.1 Novembre 2007
Version 1.2 Novembre 2009
Version 1.3 Octobre 2010
Version 1.4 Octobre 2014

CONTENU DU CHAPITRE 7

7. Les fonctions	7-1
7.1. Objectifs	7-1
7.2. Les fonctions utilisateur	7-1
7.2.1. Définition d'une fonction	7-1
7.2.1.1. L'entête de la fonction	7-1
7.2.1.2. Le corps de la fonction	7-2
7.2.2. Appel d'une fonction	7-2
7.2.2.1. Appel d'une fonction sans valeur retournée	7-2
7.2.2.2. Appel d'une fonction avec valeur retournée	7-2
7.2.3. Passage de paramètre par valeur ou par référence	7-3
7.2.3.1. Passage de paramètre par valeur	7-3
7.2.3.2. Passage de paramètre par référence	7-3
7.2.4. Visibilité des variables	7-4
7.2.4.1. Cas de la variable rayon et du paramètre r	7-4
7.2.4.2. Cas des variables surface et aire	7-5
7.2.4.3. Cas de la variable ValPI	7-5
7.2.5. Prototype d'une fonction	7-5
7.3. Les fonctions des librairies	7-6
7.3.1. Les fonctions de stdlib	7-6
7.3.2. Les fonctions de stdio	7-8
7.3.3. Les fonctions de string	7-9
7.3.4. Les fonctions de math	7-11
7.3.5. Les fonctions de time	7-12
7.4. La fonction printf	7-13
7.4.1. Les spécificateur du format d'affichage	7-13
7.4.1.1. L'indicateur de type	7-13
7.4.1.2. L'indicateur flag	7-14
7.4.1.3. Width et precision	7-14
7.4.1.4. Indicateur de taille	7-15
7.4.2. Printf, exemples	7-15
7.5. La fonction scanf	7-16
7.5.1. Particularité avec le Visual Studio 2010	7-16
7.5.2. Scanf, exemple 1	7-16
7.5.3. Les spécificateurs du format de saisie	7-17
7.5.3.1. L'indicateur de type	7-17
7.5.3.2. L'indicateur Width	7-17
7.5.3.3. L'indicateur *	7-17
7.5.3.4. Indicateur de taille	7-18
7.5.4. Scanf, exemples	7-18
7.5.4.1. Scanf, résultats de l'exemple	7-19
7.5.4.2. Scanf, remarques à propos de l'exemple	7-19
7.5.5. Scanf, exemples de situation à problème	7-19
7.6. La fonction scanf_s	7-21

7.6.1.	Scanf_s, saisie d'un caractère	7-21
7.6.2.	Scanf_s, Saisie d'une chaîne	7-22
7.6.3.	Scanf_s, problème de taille	7-23
7.7.	Conclusion	7-24
7.8.	Historique des versions	7-24
7.8.1.	Version 1.0 Octobre 2004	7-24
7.8.2.	Version 1.1 Novembre 2007	7-24
7.8.3.	Version 1.2 Novembre 2009	7-24
7.8.4.	Version 1.3 Octobre 2010	7-24
7.8.5.	Version 1.4 Octobre 2014	7-24

7. LES FONCTIONS

Ce chapitre a pour but de présenter comment réaliser des fonctions et les utiliser pour la découpe et l'organisation du programme. En plus, comme le langage C fournit de nombreuses fonctions regroupées en librairie par catégorie, ce chapitre en dressera la liste dans le but de permettre à l'utilisateur de trouver les détails d'utilisations dans l'aide des compilateurs.

7.1. OBJECTIFS

A la fin de ce chapitre, l'étudiant sera capable de :

- De réaliser des fonctions
- De maîtriser le passage des paramètres et le retour de valeur
- De maîtriser la notion de vision des variables d'une fonction
- D'utiliser les fonctions des librairies ANSI C

7.2. LES FONCTIONS UTILISATEUR

Le langage C ne propose que la fonction contrairement à certains langages qui offrent la sous-routine et la fonction.

La fonction n'est pas une tranche de programme à laquelle on a attribué un nom. Il s'agit d'un élément de programme qui effectue un traitement de ou des données reçues et qui retourne un résultat.

Il faut tout d'abord bien séparer la définition de la fonction (la réalisation) avec l'appel de la fonction (l'utilisation de la fonction).

7.2.1. DEFINITION D'UNE FONCTION

La définition se compose de deux parties :

- L'entête de la fonction.
- Le corps de la fonction.

7.2.1.1. L'ENTETE DE LA FONCTION

`[scope] [type] nom ([t1 n1, t2 n2, ...])`

Les éléments entre [] sont optionnels.

scope : appelé souvent "classe de mémoire" est utilisé ici pour spécifier la notion de local ou global.

static = local, **extern** = global, si omis, la fonction est par défaut globale.

type : spécifie le type de la valeur retournée par la fonction, si omis, cela signifie que la fonction ne retourne pas de valeur. Il est possible d'utiliser le type void pour indiquer que la fonction retourne rien.

Le **nom** de la fonction est indispensable.

t1 n1, t2 n2, ... : représente la liste des arguments ou paramètres (0 à n). On indique pour chaque argument son type et son nom. Le nom de l'argument permet de l'utiliser directement dans le corps de la fonction comme une variable locale de la fonction. Dans le cas d'un argument de type void, son nom est omis car cela correspond à une fonction sans argument.

7.2.1.2. LE CORPS DE LA FONCTION

Structure du corps de la fonction:

```
{
    [Définitions des variables locales]
    [Définitions de variables "globales"]

    [Instructions]

    [return (expression)]
}
```

Les variables locales sont définies par un type et un nom.

Les variables globales sont précédées du mot clé *static*, par exemple :

```
static uint16 count = 0;
```

Cette déclaration rend count global, count sera initialisé à 0 lors du lancement du programme, lors des appels successifs de la fonction la valeur de count est conservée.

Remarque : la syntaxe du langage C accepte la définition d'une fonction dans le corps d'une autre fonction.

7.2.2. APPEL D'UNE FONCTION

Il faut considérer deux cas :

- La fonction ne retourne pas de valeur.
- La fonction retourne une valeur.

7.2.2.1. APPEL D'UNE FONCTION SANS VALEUR RETOURNEE

Dans ce cas on appelle la fonction de la manière suivante :

```
NomFonc1 () ;          pour une fonction sans arguments.
```

Ou

```
NomFonc2 (23) ;       pour une fonction avec 1 seul argument. (23 est un exemple
numérique, cela peut aussi être une variable ou une expression).
```

7.2.2.2. APPEL D'UNE FONCTION AVEC VALEUR RETOURNEE

Dans ce cas on appelle la fonction de la manière suivante :

```
Result = NomFonc3 () ;      pour une fonction sans arguments.
```

Ou

```
Result = NomFonc4 (23) ;    pour une fonction avec 1 seul argument.
```

Il est aussi possible d'utiliser la fonction dans une expression et de passer une expression en paramètre :

```
Result = NomFonc5( temp * 25) / 15;
```

7.2.3. PASSAGE DE PARAMETRE PAR VALEUR OU PAR REFERENCE

Il est important d'introduire encore cette notion.

7.2.3.1. PASSAGE DE PARAMETRE PAR VALEUR

Prenons par exemple la fonction test1 déclarée comme suit :

```
static sint16 test1 (sint16 value)
```

Avec par exemple l'appel suivant :

```
sint16 temp;  
Temp = 25;  
Result = test1(temp);
```

La valeur de la variable temp est passée à l'argument value. Il y a copie de temp dans value. Si dans le corps de la fonction on modifie value, cela n'a aucune influence sur la variable temp.

7.2.3.2. PASSAGE DE PARAMETRE PAR REFERENCE

Prenons par exemple la fonction test2 déclarée comme suit :

```
static sint16 test2 (sint16 *pValue)
```

L'argument pValue est déclaré comme un pointeur sur un élément de type sint16.

Avec par exemple l'appel suivant :

```
sint16 temp2;  
Temp2 = 25;  
Result = test2(&temp2);
```

L'adresse de la variable temp2 est passée à l'argument pValue. pValue pointe sur la variable temp2.

Si dans le corps de la fonction on écrit :

```
*pValue = 50;
```

Cela va modifier la valeur de la variable temp2, on obtiendra temp2 = 50.

Remarque : on utilise le passage par référence lorsque l'on a besoin de réaliser une fonction qui retourne plusieurs valeurs. Par exemple si on a besoin d'un résultat et d'une indication sur la situation du résultat.

7.2.4. VISIBILITE DES VARIABLES

Voici une version simplifiée du programme fourni en exemple dans le chapitre 3.

```
// Version simplifiée de l'exemple chapitre 3, Version 2
(main + SurfaceCercle)

// DIRECTIVES DE COMPILATION
#include <stdio.h>

// Variable globale
double ValPI = 3.14159;

// Prototype de la fonction surfaceCercle
double surfaceCercle( double r);

// SUITE DE FONCTION
int main(void)
{
    // Variable locales
    double rayon = 3.5;
    double surface;

    rayon = atof(argv[1]);
    // calcul de la surface (appel fonction)
    surface = surfaceCercle(rayon);
    printf ("surface = %f \n", surface);
    return 0;
}

// Définition de la fonction surfaceCercle
double surfaceCercle( double r)
{
    double aire;           // décl. variable locale

    aire = ValPI*r*r;
    return (aire);         // retour du résultat
}
```

7.2.4.1. CAS DE LA VARIABLE RAYON ET DU PARAMETRE R

La variable `double rayon` est déclarée localement dans le programme principal. Lors de l'appel de la fonction `surfaceCercle`, la valeur de la variable `rayon` est copiée dans le paramètre `r`. Le paramètre `r` est utilisable comme une variable à l'intérieure de la fonction.

Si la fonction `surfaceCercle`, n'avait pas le paramètre `r`, et que l'on cherche à utiliser directement la variable `rayon`, cette variable ne serait pas connue dans la fonction `surfaceCercle`.

7.2.4.2. CAS DES VARIABLES SURFACE ET AIRE

La variable double `surface` est déclarée dans le programme principal.

La variable double `aire` est déclarée dans la fonction `surfaceCercle`.

Lors de l'appel de la fonction `surfaceCercle`, la valeur de la variable `aire` (la valeur retournée) et affectée à la variable `surface`.

Si la fonction `surfaceCercle`, ne retourne pas de valeur, et que l'on cherche à utiliser directement la variable `surface` à la place de la variable `aire` dans la fonction, cette variable ne serait pas connue dans la fonction `surfaceCercle`.

7.2.4.3. CAS DE LA VARIABLE VALPI

La variable double `ValPI` est déclarée à l'extérieur des fonctions `main` et `surfaceCercle`. Cette déclaration extérieure ou déclaration globale rend cette variable visible depuis l'ensemble du programme, c'est ce qui permet de l'utiliser directement dans la fonction `surfaceCercle`.


7.2.5. PROTOTYPE D'UNE FONCTION

Si on regarde à nouveau l'exemple tiré du chapitre 3, on constate la présence d'un prototype de fonction.

```
// Prototype de la fonction surfaceCercle
double surfaceCercle( double r );
```

Le prototype de la fonction correspond à une copie de l'entête de la fonction à laquelle on ajoute un point virgule (;).

```
// Définition de la fonction surfaceCercle
double surfaceCercle( double r)
{
    double aire;           // décl. variable locale
    ...
}
```

 Le prototype de la fonction permet d'utiliser la fonction, en amont de sa définition.

Lorsque la fonction est définie à l'extérieur du module (dans un autre fichier), on utilise un fichier `.h`, contenant les prototypes des fonctions définies dans le fichier `.c` correspondant. (Voir la version 3 de l'exemple du chapitre 3).

7.3. LES FONCTIONS DES LIBRAIRIES

Les prototypes des fonctions d'une librairie sont placés dans des fichiers `.h` fournis par le compilateur. Par exemple le fichier **math.h** supporte un certain nombre de fonctions mathématiques.

Donc pour utiliser une fonction il faut savoir de quelle librairie elle fait partie et inclure le fichier `.h` correspondant.

Voici les différents groupes de fonctions de la *ANSI Runtime library*, que l'on peut traduire par librairie des fonctions d'exécution du standard ANSI. Cela correspond à un minimum, la plus part des compilateurs en offre plus, en particulier le Visual C++ qui en plus fourni toutes les librairies propre à Windows.

Voici un tableau avec le nom des fichiers d' "include" et le thème des fonctions associées.

Nom du fichier	Thème des fonctions associées
<assert.h>	Fonctions de diagnostics
<ctype.h>	Fonctions de test des caractères
<errno.h>	Définitions de macro, liée à la signalisation d'erreur
<float.h>	Définitions de macro, spécifiant les caractéristiques des éléments en virgule flottante pour l'environnement
<limits.h>	Définitions des valeurs limites de l'arithmétique entière
<locale.h>	Fonctions pour la gestion de paramètres locaux
<math.h>	Fonctions mathématiques en double précision
<setjmp.h>	Fonctions de saut pour réaliser des mécanisme système
<signals.h>	Fonction pour la gestion des signaux (gestion d'événements)
<stdarg.h>	Fonctions et macros pour réaliser des fonctions avec un nombre de paramètres variable.
<stddef.h>	Définitions standard
<stdio.h>	Fonctions d'entrée-sortie
<stdlib.h>	Fonctions d'utilité générale
<string.h>	Fonctions de manipulation des chaînes de caractères
<time.h>	Fonctions liées à l'heure et à la date

Nous ne traiterons dans ce chapitre que les groupes de fonctions les plus couramment utilisés.

7.3.1. LES FONCTIONS DE STDLIB

L'ensemble des fonctions d'utilités générales, fournie par **stdlib.h**, sont réparties dans les catégories suivantes:

- Manipulation de chaînes de caractères
- Génération de nombre pseudo aléatoire
- Gestion de la mémoire (allocation dynamique)
- Fonctions systèmes
- Recherche et tri
- Arithmétique entière

Nom	Prototype	Description
Manipulation de chaînes de caractères		
atof	double atof (const char *str);	Conversion string en double
atoi	int atoi (const char *str);	Conversion string en int
atol	long atol (const char *str);	Conversion string en long
strtod	double strtod (const char *str, char **endptr);	Conversion string en double, avec info sur dernier caractère
strtol	long strtol (const char *str, char **endptr, int base);	Conversion string en long, avec info sur dernier caractère et choix de la base
strtoul	unsigned long strtol (const char *str, char **endptr, int base);	Conversion string en unsigned long, avec info sur dernier caractère et choix de la base
Génération de nombre pseudo aléatoire		
rand	int rand (void);	Fourni un nombre aléatoire
srand	void srand (unsigned int seed);	Initialisation du générateur de nombre aléatoire
Gestion de la mémoire (allocation dynamique)		
calloc	void *calloc (size_t n, size_t size);	Réservation de n * size octets
free	void free (void *ptr);	Deallocation du bloc mémoire
malloc	void *malloc (size_t size);	Réservation de size octets
realloc	void realloc (void *ptr, size_t size);	Modification de la taille du bloc déjà alloué
Fonctions systèmes		
abort	void abort (void);	Termine abruptement l'exécution d'un programme
atexit	int atexit (void (*func) (void));	Mise en ordre de l'environnement avant la fin d'un programme
exit	void exit (int status);	Termine normalement l'exécution d'un programme
getenv	char *getenv (const char *name);	Recherche variable d'environnement
system	int system (const char *str);	Commande à faire exécuter par le système d'exploitation
Recherche et tri		
bsearch	void bsearch (const void *key, const void *base, size_t nel, size_t *keysize, int (*compar) (const void *, const void *));	Recherche binaire
qsort	void qsort (void *base, size_t nel, size_t keysize, int (*compar) (const void *, const void *));	Tri rapide (quick sort)
Arithmétique entière		
abs	int abs (int i);	Valeur absolue de i (int)
div	div_t div (int numer, int denom);	Division entière, résultat (quotient et reste) dans une structure
labs	long abs (long j);	Valeur absolue de j (long)
ldiv	ldiv_t ldiv (long numer, long denom);	Division entière, résultat (quotient et reste) dans une structure

7.3.2. LES FONCTIONS DE STDIO

Le fichier `stdio.h` regroupe les fonctions d'entrée-sortie standard. Il s'agit de la lecture et de l'écriture dans les fichiers, ainsi que de l'affichage à l'écran et de la saisie au clavier.

La plus part des fonctions décrites dans le tableau ci-dessous ont un paramètre : `FILE *stream`, qui correspond au descripteur de fichier.

Nom	Prototype	Description
<code>clearerr</code>	<code>void clearerr (FILE *stream);</code>	Efface les indicateurs d'erreurs
<code>fclose</code>	<code>int fclose (FILE *stream);</code>	Fermeture du fichier
<code>feof</code>	<code>int feof (FILE *stream);</code>	Test de la fin du fichier
<code>ferror</code>	<code>int ferror (FILE *stream);</code>	Test des erreurs du fichier
<code>fflush</code>	<code>int fflush (FILE *stream);</code>	Vide le tampon de fichier
<code>fgetc</code>	<code>int fgetc (FILE *stream);</code>	Lecture d'un caractère dans le fichier
<code>fgetpos</code>	<code>int fgetpos (FILE *stream, fpos_t *pos);</code>	Fourni la position dans le fichier
<code>fgets</code>	<code>char *fgets (char *str, int n, FILE *stream);</code>	Lecture de n caractères dans le fichier, résultat dans un string
<code>fopen</code>	<code>FILE *fopen (const char *filename, const char *mode);</code>	Ouverture du fichier
<code>fprintf</code>	<code>int fprintf (FILE *stream, const char *format, ...);</code>	Ecriture formatée dans le fichier
<code>fputc</code>	<code>int fputc (int c, FILE *stream);</code>	Ecriture d'un caractère dans le fichier
<code>fputs</code>	<code>int fputs (const char *str, FILE *stream);</code>	Ecriture d'une chaîne de caractères dans le fichier
<code>fread</code>	<code>int fread (void *buf, size_t size, int nbelem, FILE *stream);</code>	Lecture d'un bloc de données, résultat dans un tableau
<code>freopen</code>	<code>FILE *freopen (const char *filename, const char *mode, FILE *stream);</code>	Redirection du fichier
<code>fscanf</code>	<code>Int fscanf (FILE *stream, const char *format, ...);</code>	Lecture formatée dans le fichier
<code>fsetpos</code>	<code>int fsetpos (FILE *stream, fpos_t *pos);</code>	Positionnement dans le fichier
<code>fseek</code>	<code>int fseek (FILE *stream, long offset, int offsetMode);</code>	Déplacement dans le fichier
<code>ftell</code>	<code>int ftell (FILE *stream);</code>	Donne position courante dans le fichier
<code>fwrite</code>	<code>int fwrite (void *buf, size_t size, int nbelem, FILE *stream);</code>	Ecriture d'un bloc de données, dans le fichier
<code>getc</code>	<code>int getc (FILE *stream);</code>	Lecture d'un caractère dans le fichier
<code>getchar</code>	<code>int getchar (void);</code>	Lecture d'un caractère dans tampon du clavier
<code>gets</code>	<code>char *gets (char *str);</code>	Lecture d'une chaîne de caractères dans tampon du clavier
<code>perror</code>	<code>char *perror (const char *mess);</code>	Affichage d'un message d'erreur
<code>printf</code>	<code>int printf (const char *format, ...);</code>	Affichage formaté

putc	int putc (int c, FILE *stream);	Ecriture d'un caractère dans le fichier
putchar	int putchar (int c);	Affichage d'un caractère
puts	int puts (const char *str);	Affichage d'une chaîne de caractères
remove	int remove (const char *filename);	Suppression fichier
rename	int rename (const char *oldFilename, const char *newFilename);	Renomme le fichier
rewind	void rewind (FILE *stream);	Positionnement au début du fichier
scanf, scanf_s	int scanf (const char *format, ...);	Lecture formatée depuis le lecteur standard (clavier)
setbuf	void setbuf (FILE *stream, char *buf);	Crée un tampon pour le fichier
setvbuf	int setvbuf (FILE *stream, char *buf, int mode, size_t size);	Modification des propriétés du tampon pour le fichier
sprintf	int sprintf (char *str, const char *format, ...);	Ecriture formatée dans la chaîne de caractère
sscanf	int sscanf (char *str, const char *format, ...);	Lecture formatée à partir du clavier dans la chaîne de caractère
tmpfile	FILE *tmpfile (void);	Création d'un fichier temporaire
tmpnam	char *tmpnam (char *name);	Création d'un nom pour un fichier temporaire
vfprintf	int vfprintf (FILE *stream, const char *format, va_list arg);	Comme fprintf mais avec un pointeur sur une liste d'arguments
vprintf	int vprintf (const char *format, va_list arg);	Comme printf mais avec un pointeur sur une liste d'arguments
vsprintf	int vsprintf (char *str, const char *format, va_list arg);	Comme sprintf mais avec un pointeur sur une liste d'arguments
ungetc	int ungetc (int c, FILE *stream);	Reécriture du caractère lu

7.3.3. LES FONCTIONS DE STRING

Le fichier **string.h** regroupe les fonctions de manipulation de chaîne de caractères.

- Les fonctions dont le nom commence par **str** agissent sur des chaînes de caractères terminées par un 0 (null).
- Les fonctions dont le nom commence par **strn** agissent sur des chaînes de caractères dont la longueur est spécifiée.
- Les fonctions dont le nom commence par **mem** agissent sur des tableaux de données dont la longueur est spécifiée.

Nom	Prototype	Description
memchr	void *memchr (const void *buf, int c, size_t n);	recherche de la 1 ^{ère} occurrence de c dans les n premiers octets de buf
memcmp	int memcmp (const void *buf1, const void *buf2, size_t n);	compare les n premiers octet de buf1 avec les n premier octets de buf2
memcpy	void *memcpy (void *buf1, const void *buf2, size_t n);	copie n octet de buf2 dans buf1

memcpy	void *memcpy (void *buf1, const void *buf2, size_t n);	copie n octet de buf2 dans buf1
memset	void *memset (void *buf, int c, size_t n);	remplit les n premier octets de buf avec c
strcpy	char *strcpy (char *s1, const char *s2);	copie le contenu de s2 dans s1
strncpy	char *strncpy (char *s1, const char *s2, size_t n);	copie n caractères de s2 dans s1
strcat	char *strcat (char *s1, const char *s2);	ajoute le contenu de s2 à s1
strncat	char *strncat (char *s1, const char *s2, size_t n);	ajoute n caractères de s2 dans s1
strcmp	int strcmp (const char *s1, const char *s2);	compare s1 avec s2
strncmp	int strncmp (const char *s1, const char *s2, size_t n);	compare les n caractères de s1 avec s2
strerror	char *strerror (int errnum);	pointe sur le message d'erreur correspondant à errnum
strlen	size_t strlen (const char *s);	fourni la longueur de la chaîne s
strchr	char *strchr (const char *s, int c);	pointe sur la 1ere occurrence de c (converti en char) dans s
strcspn	size_t strcspn (const char *s1, const char *s2);	retourne le nombre de caractères de s1 qui ne correspondent pas à ceux de s2
strpbrk	char *strpbrk (const char *s1, const char *s2);	retourne un pointeur sur le 1 ^{er} caractère de s1 qui correspond à celui de s2
strrchr	char *strrchr (const char *s, int c);	retourne un pointeur sur le 1 ^{er} caractère de s qui correspond à c (converti en char)
strspn	size_t strspn (const char *s1, const char *s2);	compte le nombre de caractères de s1 qui correspondent à ceux de s2
strstr	char *strstr (const char *s1, const char *s2);	retourne un pointeur à l'endroit où se situe la chaîne s2 dans s1
strtok	char *strtok (char *s1, const char *s2);	découpe la chaîne s1 en symbole élémentaire (token), séparé par les caractères de s2
strtok	char *strtok (char *s1, const char *s2);	découpe la chaîne s1 en symbole élémentaire (token), séparé par les caractères de s2

7.3.4. LES FONCTIONS DE MATH

Le fichier math.h fourni les fonctions trigonométriques et hyperboliques, les fonctions exponentiel et logarithmique, ainsi que diverses fonctions mathématiques.

Les fonctions trigonométriques utilisent les radians et non pas les degrés.

Nom	Prototype	Description
Fonctions trigonométriques		
acos	double acos (double x);	Calcule l'arc cosinus de x
asin	double asin (double x);	Calcule l'arc sinus de x
atan	double atan (double x);	Calcule l'arc tangente de x
atan2	double atan2 (double x, double y);	Calcule l'arc tangente de x/y
cos	double cos (double x);	Calcule le cosinus de l'angle x
cosh	double cosh (double x);	Calcule le cosinus hyperbolique de l'angle x
sin	double sin (double x);	Calcule le sinus de l'angle x
sinh	double sinh (double x);	Calcule le sinus hyperbolique de l'angle x
tan	double tan (double x);	Calcule la tangente de l'angle x
tanh	double tanh (double x);	Calcule la tangente hyperbolique de l'angle x
Fonctions logarithmiques et exponentielles		
exp	double exp (double x);	Calcule l'exponentiel de x
frexp	double frexp (double x, int *exp);	Sépare la mantisse et l'exposant de x
ldexp	double ldexp (double m, int exp);	Calcul de $m * 2^{\text{exp}}$
log	double log (double x);	Calcule le logarithme naturel de x
log10	double log10 (double x);	Calcule le logarithme en base 10 de x
modf	double modf (double x, double *intpart);	Retourne la partie fractionnaire de x et fourni la partie entière par *intpart
pow	double pow (double x, double y);	Calcul x^y
sqrt	double sqrt (double x);	Calcule la racine carrée de x
Fonctions diverses		
ceil	double ceil (double x);	Calcule le plus petit entier \geq à x
fabs	double fabs (double x);	Calcule la valeur absolue de x
floor	double floor (double x);	Calcule le plus grand entier $<$ x
fmod	double fmod (double x, double y);	Calcule le reste de la division de x par y (modulo)

7.3.5. LES FONCTIONS DE TIME

Le fichier time.h fournis des fonctions donnant l'accès à l'horloge système et au calendrier.

Le fichier time.h contient les définitions de trois types:

- clock_t type arithmétique capable de représenter le temps
- time_t type arithmétique capable de représenter le temps
- tm structure comprenant les éléments du calendrier (heure et date)

La structure tm contient les éléments suivants :

```
int tm_sec; /* secondes [0, 59] */
int tm_min; /* minutes [0, 59] */
int tm_hour; /* heures [0, 23] */
int tm_mday; /* jour du mois [1, 31] */
int tm_mon; /* mois depuis janvier [0, 11] */
int tm_year; /* année depuis 1900 */
int tm_wday; /* jours depuis dimanche [0, 6] */
int tm_yday; /* jours depuis 1er janvier [0, 365] */
int tm_isdst; /* indicateur heur d'été */
```

Nom	Prototype	Description
clock	clock_t clock (void);	Valeur de l'horloge processeur au moment de l'appel de la fonction
time	time_t time (time_t *timer);	Nombres de secondes écoulées depuis minuit
mktime	time_t mktime (struct tm *timeptr);	conversion de l'heure de la structure tm en une valeur normalisée
asctime	char *asctime (const struct tm *timeptr);	conversion de l'heure de la structure tm en une chaîne de caractère affichable
ctime	char *ctime (const time_t *timer);	conversion de l'heure en une chaîne de caractère affichable
difftime	double difftime (time_t time1, time_t time0);	retourne la différence time1 – time0 exprimée en secondes
gmtime	struct tm *gmtime (const time_t *timer);	conversion de l'heure en une structure tm (heure GMT)
localtime	struct tm *localtime (const time_t *timer);	conversion de l'heure en une structure tm (heure locale)
strtime	size_t strtime (char *s, size_t maxsize, const char *format, const struct tm *timeptr);	création d'une chaîne de caractère, avec formatage, à partir de la structure tm

7.4. LA FONCTION PRINTF

Cette fonction permet d'écrire sur le périphérique de sortie standard (stdout), ce qui correspond à l'affichage à l'écran. Elle nécessite le fichier d'inclure **stdio.h**.

Prototype de la fonction :

```
int printf (const char *format, ... );
```

La fonction printf retourne un code d'erreur, cependant dans la plus part des cas on utilise la fonction printf en ignorant la valeur de retour.

Les paramètres passé à la fonction printf sont assez particulier, le `const char *format` représente une chaîne de caractère contenant les indications du format d'affichage. Les ... représentent une liste d'argument.

Pour mieux comprendre voici un exemple:

```
#include <stdio.h>

int main (void)
{
    int min = -100;
    int max = 500;

    printf ("min = %d max = %d \n", min, max);
    return(0);
}
```

On obtient l'affichage suivant :

```
min = -100 max = 500
```

On constate que dans le format qui est donné entre " " il est possible d'y placer du texte à afficher mais aussi des spécificateur de type comme le %d. Le \n qui est appelé une séquence d'échappement (*escape sequence*), à pour effet de forcer un retour de chariot (*carriage return*), ce qui termine l'affichage sur cette ligne.

La liste d'argument contient deux variables entières, leur valeur est affichée à la place des %d. Il faut le même nombre de % que de variables fournies.

7.4.1. LES SPECIFICATEUR DU FORMAT D'AFFICHAGE

La syntaxe d'un spécificateur d'affichage est la suivante :

```
%[flags] [width] [.precision] [{h | l | L}]type
```

Tous les éléments sont optionnels excepté le type. Nous allons décrire le rôle de chacun de ces indicateurs.

7.4.1.1. L'INDICATEUR DE TYPE

Cet indicateur est obligatoire, il détermine le type de donnée et la conversion nécessaire pour l'affichage.

Voici la table des caractères utilisés pour indiquer le type et la conversion d'affichage:

Caractère	Type	Conversion
d, i	Entier signé	Affichage en décimal
u	Entier non signé	Affichage en décimal (valeur non signée)
o	Entier signé	Affichage en octal
X, x	Entier signé	Affichage en hexadécimal, avec x utilise des minuscules, avec X des majuscules.
f	réels	Virgule flottante
E, e	réels	Virgule flottante, notation avec exposant
G, g	réels	Virgule flottante, utilise f ou e en fonction de la taille du nombre.
c	caractère	Affiche le caractère ASCII
s	Chaîne de caractères	Affichage de type texte
n	void*	Affichage selon implémentation
p	int*	Fourni le nombre d'éléments affichés

7.4.1.2. L'INDICATEUR FLAG

Optionnel, agit sur le format d'affichage (justification, espaces, etc.)

Le tableau ci-dessous résume les différentes possibilités

Caractère Flag	Effet sur l'affichage
-	Justification à gauche
+	Force l'affichage d'un signe + ou - devant le nombre
espace	Les nombres positifs sont affichés avec un espace devant, les négatifs avec le signe -.
#	Sont effet dépend du spécificateur de type: Pour c, d, i, s, u est sans effet Pour o , préfixe avec un 0 Pour x, X , préfixe avec 0x Pour e, E, f, F, g, G , force l'ajout d'un point décimal.

7.4.1.3. WIDTH ET PRECISION

Optionnel, permet de définir le nombre de caractères à afficher. Pour les nombres réels, il est possible d'indiquer le nombre de caractères après le point décimal.

Par exemple 3.4 indique que l'on veut 3 caractères au moins avant le point décimal et 4 au maximum après.

Si le nombre de caractère est précédé d'un 0, le nombre 24, en spécifiant %04d est affiché 0024

Pour les chaînes de caractères on utilise .n pour limiter le nombre de caractères affichés.

7.4.1.4. INDICATEUR DE TAILLE

Permet d'indiquer la taille de l'argument. Il se place avant l'indicateur de type.

Indicateur de taille	Type correspondant
h	Indique que l'entier utilisé est un short int ou unsigned short int
l	Indique que l'entier utilisé est un long int ou unsigned long int Indique que le réel utilisé est un double
L	Agit comme l pour les entiers. Pour les réels indique l'utilisation d'un long double. Agit idem l pour un double

7.4.2. PRINTF, EXEMPLES

Voici un petit programme utilisant quelques situations de printf.

```
#include <stdio.h>

int main (void)
{
    int min = -100;
    int max = 500;
    float coeff = 123456.78912;
    unsigned short int val = -45;

    printf ("min = %d max = %d \n", min, max);
    printf ("max = %06d min(hex) = %#x \n", max, min);
    printf ("val = %hu val(hex)= %06hX \n", val, val);
    printf ("coeff %f %e %g \n", coeff, coeff, coeff);
    printf ("coeff %1.3f \n", coeff);

    return(0);
}
```

Voici les résultats au niveau de l'affichage

```
min = -100 max = 500
max = 000500 min(hex) = 0xffffffff9c
val = 65491 val(hex)= 00FFD3
coeff 123456.789063 1.234568e+005 123457
coeff 123456.789
```

Il faut bien observer la combinaison des indicateurs et leurs effets sur l'affichage.

7.5. LA FONCTION SCANF

La fonction **scanf** permet la saisie formatée de donnée depuis le clavier (*standard input stream **stdin***). Les données lues sont attribuées aux arguments qui doivent être des pointeurs sur des variables d'un type correspondant au spécificateur de type du format.

Syntaxe de la fonction scanf :

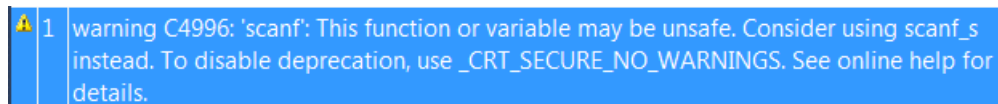
```
int scanf( const char *format , . . . );
```

La fonction scanf retourne un code d'erreur, cependant dans la plus part des cas on utilise la fonction scanf en ignorant la valeur de retour.

Les paramètres passé à la fonction scanf sont assez particulier, le `const char *format` représente une chaîne de caractère contenant les indication du format de saisie. Les `. . .` représentent une liste d'argument. La liste d'argument contient l'adresse des variables dans lesquelles on veut obtenir le résultat de la saisie

7.5.1. PARTICULARITE AVEC LE VISUAL STUDIO 2010

La fonction scanf produit le warning suivant :



Pour éviter les warning, il faut placer la directive ci-après avant les `#include`.

```
#define _CRT_SECURE_NO_WARNINGS
```

7.5.2. SCANF, EXEMPLE 1

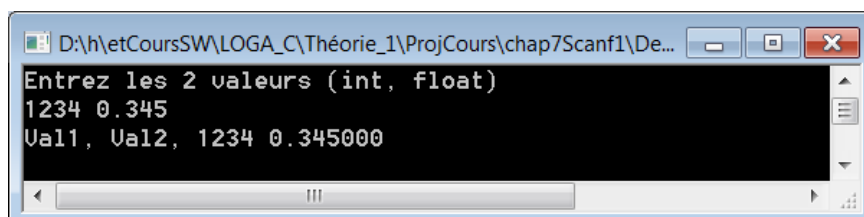
Pour mieux comprendre voici un exemple:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int Val1;
    float Val2;

    // Demande des 2 valeurs à l'utilisateur
    printf ("Entrez les 2 valeurs (int, float) \n");
    scanf ("%d%f", &Val1, &Val2);
    // Affichage du résultat
    printf ("Val1, Val2, %d %f \n", Val1, Val2);
    return 0;
}
```

On obtient la situation suivante (cas d'une application console):



On constate que dans le format qui est donné entre " " on y place des spécificateur de type comme le **%d** et le **%f**.

La liste d'argument contient l'adresses d'une variable entière (&Val1) et d'une variable float (&Val2), pour que la saisie aie lieu correctement, il faut une correspondance exact du type de la variable résultat et du spécificateur du type de saisie.

7.5.3. LES SPECIFICATEURS DU FORMAT DE SAISIE

La syntaxe d'un spécificateur de saisie est la suivante :

%[*] [width] [{h | l | L}]type

Tous les éléments sont optionnels (ceux entre []) excepté le type. Nous allons décrire le rôle de chacun de ces indicateurs.

7.5.3.1. L'INDICATEUR DE TYPE

Cet indicateur est obligatoire, il détermine le type de donnée et la conversion nécessaire pour la saisie.

Voici la table des caractères utilisés pour indiquer le type et la conversion lors de la saisie:

Caractère	Description de la saisie	Type de l'argument
d	Saisie d'un entier en décimal	Pointeur sur int
d, i	Saisie d'un entier en décimal, supporte les préfixes + ou -	Pointeur sur int
u	Saisie d'un entier en décimal (valeur non signée)	Pointeur sur unsigned int
o	Saisie d'un entier en octal	Pointeur sur int
X, x	Saisie d'une valeur hexadécimale, avec x utilise des minuscules, avec X des majuscules.	Pointeur sur int
f, E,e, G,g	Saisie d'un nombre en virgule flottante, s'adapte au format.	Pointeur sur float
c	Saisie d'un caractère ASCII	Pointeur sur char
s	Saisie d'une chaîne de caractères	Pointeur sur char
n	saisie d'un pointeur selon implémentation	void*
p	Fourni le nombre d'éléments saisis	Pointeur sur int

7.5.3.2. L'INDICATEUR WIDTH

Optionnel, permet d'indiquer le nombre de caractères à saisir. La saisie se termine lorsque le nombre de caractères spécifiés a été lu.

7.5.3.3. L'INDICATEUR *

Un astérisque (*) placé après le signe %, indique à scanf qu'il faut lire la donnée, mais qu'il ne faut pas la convertir.

7.5.3.4. INDICATEUR DE TAILLE

Permet de modifier la taille du type de l'argument. Il se place avant l'indicateur de type.

Indicateur de taille	Indicateur de type	Type de l'argument
h	d, i, o, x, X	Pointeur sur short int
h	u	Pointeur sur unsigned short int
l	d, i, o, x, X	Pointeur sur long int
l	u	Pointeur sur unsigned long int
l	e, E, f, g, or G	Pointeur sur double
L	e, E, f, g, or G	Pointeur sur long double

7.5.4. SCANF, EXEMPLES

Voici un petit programme montrant quelques cas d'utilisations de la fonction scanf.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main (void)
{
    short int  ValInt1;
    int        ValInt2;
    long int   ValInt3;
    float      ValReal1;
    double     ValReal2;
    double     ValReal3;
    char chaine1[20];

    // Demande de 3 valeurs entière à l'utilisateur
    printf ("Entrez les 3 valeurs (short ,int, long) \n");
    scanf  (" %hd %i %li", &ValInt1, &ValInt2, &ValInt3);
    // Affichage du résultat
    printf ("ValInt1 ValInt2 ValInt3, %hd %d %ld \n",
           ValInt1, ValInt2, ValInt3);
    // Demande 3 valeurs réelles à l'utilisateur
    printf ("Entrez les 3 valeurs (float ,double,
           double(exp) \n");
    scanf  (" %f %lf %Lf", &ValReal1, &ValReal2,
           &ValReal3);
    // Affichage du résultat
    printf ("ValReal1 ValReal2 ValReal3, %f %lf %lf \n",
           ValReal1, ValReal2, ValReal3);
    // Demande 2 mots à l'utilisateur, ignore le premier
    printf ("Entrez les 2 mots \n");
    scanf  ("%*s %s", chaine1);
    // Affichage du résultat
    printf ("chaine1 %s \n", chaine1);
    printf ("Pause, press enter \n");
    scanf ("%*c%c", &Rep);
    return 0;
}
```

7.5.4.1. SCANF, RESULTATS DE L'EXEMPLE

```

D:\h\etCoursSW\LOGA_C\Théorie_1\ProjCours\Chap7Scanf2\Debug\Chap7Scanf2.e...
Entrez les 3 valeurs (short ,int, long)
-123 -12345 -12345678
ValInt1 ValInt2 ValInt3, -123 -12345 -12345678
Entrez les 3 valeurs (float ,double, double(exp)
1.25 1.3333333333333333 1.25e10
ValReal1 ValReal2 ValReal3, 1.250000 1.333333 12500000000.000000
Entrez les 2 mots
trop froid
chaine1 froid
Pause, press enter
  
```

7.5.4.2. SCANF, REMARQUES A PROPOS DE L'EXEMPLE

Pour la gestion des nombres entiers, scanf est assez tolérant avec le d et le i, puisque avec %d il accepte le nombre négatif.

Pour la gestion des nombres flottants, en spécifiant f, scanf accepte aussi les formats avec exposant. Notez que l'usage de %Lf pour un double n'a pas posé de problème.

Pour la gestion des chaînes de caractères, il faut noter la façon de déclarer la chaîne et la façon de la référer dans la fonction scanf (pas de &).

```
char chaine1[20];
```

```
scanf ("%*s%s", chaine1);
```

On notera encore l'usage du %*s qui permet de ne saisir que le 2^{ème} mot entré par l'utilisateur.

Pour réaliser une pause, il faut consommer le caractère enter qui traîne, sinon le scanf passe tout droit d'où:

```
printf ("Pause, press enter \n");
scanf ("%*c%c", &Rep);
```

7.5.5. SCANF, EXEMPLES DE SITUATION A PROBLEME

Voici un petit programme montrant deux cas conduisant à des problèmes.

```

#define _CRT_SECURE_NO_WARNINGS

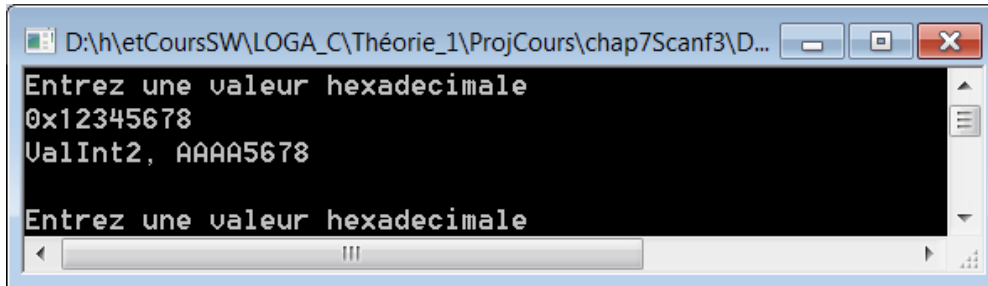
#include <stdio.h>

int main (void)
{
    short int          ValInt1;
    long int           ValInt2;

    ValInt2 = 0xAAAAAAAA;
  
```

```
// Saisie avec perte d'info
printf ("Entrez une valeur hexadecimale \n");
scanf ("%hX" , &ValInt2) ;
// Affichage du résultat
printf ("ValInt2, %lX \n\n", ValInt2);
```

A l'exécution de la 1^{ère} partie on obtient :

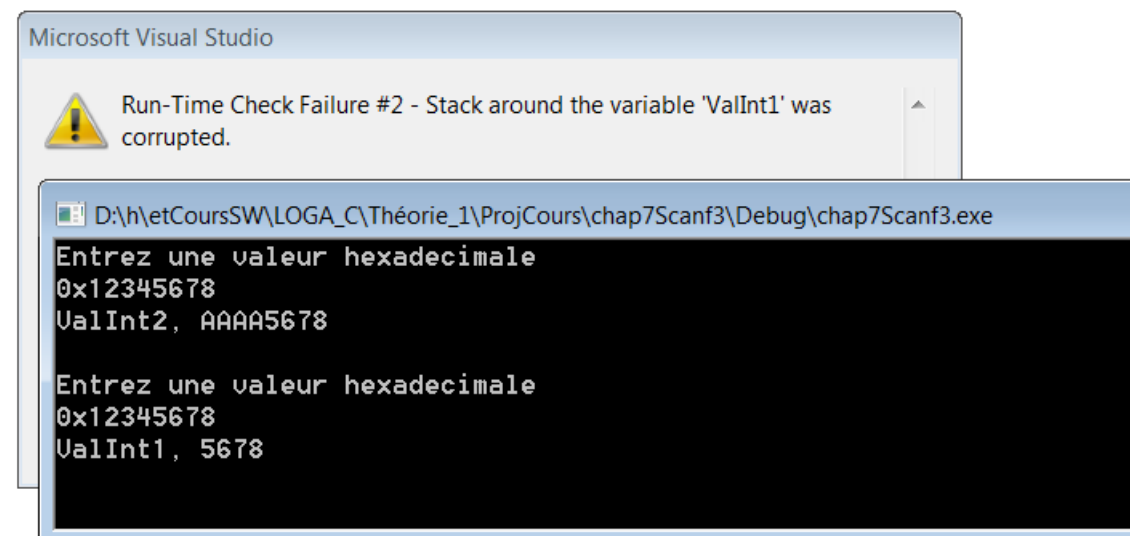


Dans ce cas, avec **h** on indique que la taille de la variable destination est 16 bits (short int), et on fournit l'adresse d'une variable 32 bits (ValInt2). Pour montrer que l'on écrit que 16 bits dans ValInt2, ValInt2 a été initialisée au préalable avec 0xAAAAAAAA. Le résultat AAAA5678 nous montre que seul les 16 bits de poids faible de la valeur saisie ont été transférés.

Voici la 2^{ème} partie du programme:

```
printf ("Entrez une valeur hexadecimale \n");
scanf ("%X" , &ValInt1) ;
// Affichage du résultat
printf ("ValInt1, %X \n\n", ValInt1);
return 0;
}
```

A l'exécution de la 2^{ème} partie on obtient :



On constate que l'on obtient bien le poids faible dans ValInt1, mais le système écrit une valeur 32 bits, donc il y a accès à la mémoire en dehors de ValInt1 (short = 16 bits), ce qui est détecté comme une corruption.

7.6. LA FONCTION SCANF_S

La fonction **scanf_f** est une version sécurisée du **scanf**. Cette fonction n'est pas standard, c'est une spécialité Microsoft introduite depuis le Visual studio 2005.

Son utilisation est la même que **scanf**, cependant il y a quelques différences dans le cas du traitement d'un caractère (**%c**) et dans le cas d'une chaîne de caractère (**%s**).

7.6.1. SCANF_S, SAISIE D'UN CARACTERE

Lors de la saisie d'un caractère il faut indiquer le nombre de caractère à saisir.

Par exemple :

```
char UserAnswer;
```

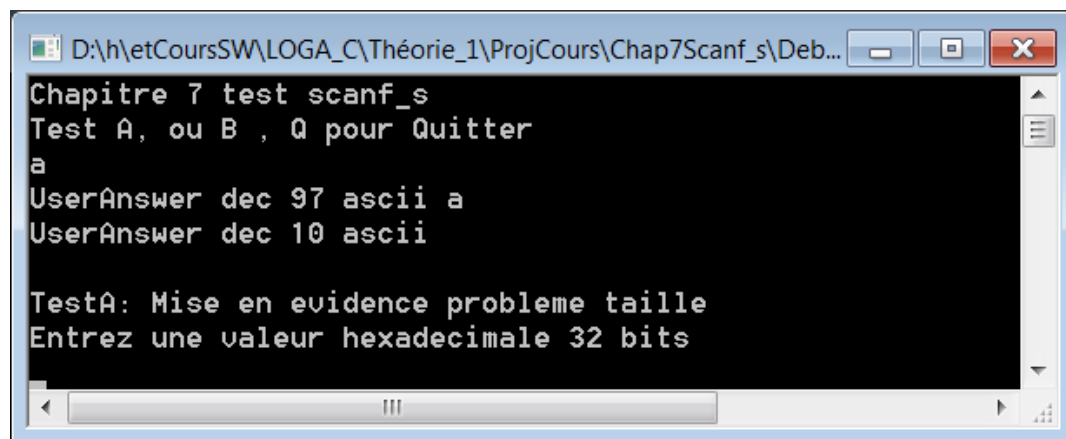
```
scanf_s("%c", &UserAnswer, 1);
```

Cette solution permet d'obtenir le caractère, mais comme dans le tampon on trouve le caractère équivalent à l'action sur la touche enter, un deuxième appel (situation en boucle) va fournir le résultat sans intervention de l'utilisateur.

Le programme ci-dessous met en évidence le problème :

```
scanf_s("%c", &UserAnswer, 1);  
printf ("UserAnswer dec %d ascii %c \n",  
        UserAnswer, UserAnswer);  
scanf_s("%c", &UserAnswer2, 1);  
printf ("UserAnswer dec %d ascii %c \n",  
        UserAnswer2, UserAnswer2);
```

Lors de l'exécution on obtient :



Pour éviter cela il faut utiliser la solution suivante :

```
scanf_s("%c%c", &UserAnswer, 2);
```

A noter l'indication de 2, pour saisie de 2 caractères.

7.6.2. SCANF_S, SAISIE D'UNE CHAÎNE

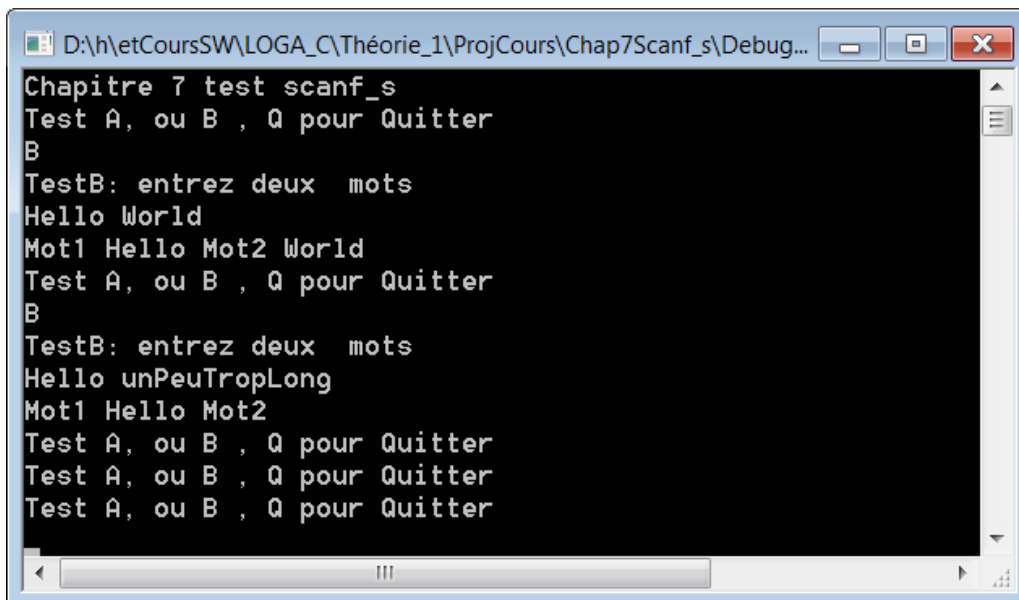
Avec **scanf_s**, lors de la saisie d'une chaîne, il faut indiquer le nombre de caractère max que l'on veut saisir.

Exemple :

```
char Mot1[10];  
char Mot2[10];  
  
printf("TestB: entrez deux mots \n");  
scanf_s("%s%s", Mot1, 9, Mot2, 9);  
scanf_s("%*c"); // pour éliminer CR  
printf ("Mot1 %s Mot2 %s \n", Mot1, Mot2);
```

Pour chaque **%s** il faut indiquer la taille maximum, on prend la taille du tableau -1 pour éviter des accès hors de la mémoire prévue.

Exemple résultat :



```
D:\h\etCoursSW\LOGA_C\Théorie_1\ProjCours\Chap7Scanf_s\Debug...  
Chapitre 7 test scanf_s  
Test A, ou B , Q pour Quitter  
B  
TestB: entrez deux mots  
Hello World  
Mot1 Hello Mot2 World  
Test A, ou B , Q pour Quitter  
B  
TestB: entrez deux mots  
Hello unPeuTropLong  
Mot1 Hello Mot2  
Test A, ou B , Q pour Quitter  
Test A, ou B , Q pour Quitter  
Test A, ou B , Q pour Quitter
```

On constate que le **scanf_s** protège la mémoire, mais que dans le cas du mot trop long, on n'obtient pas l'affichage de sa valeur et qu'en plus il y a des caractères qui restent dans le tampon.

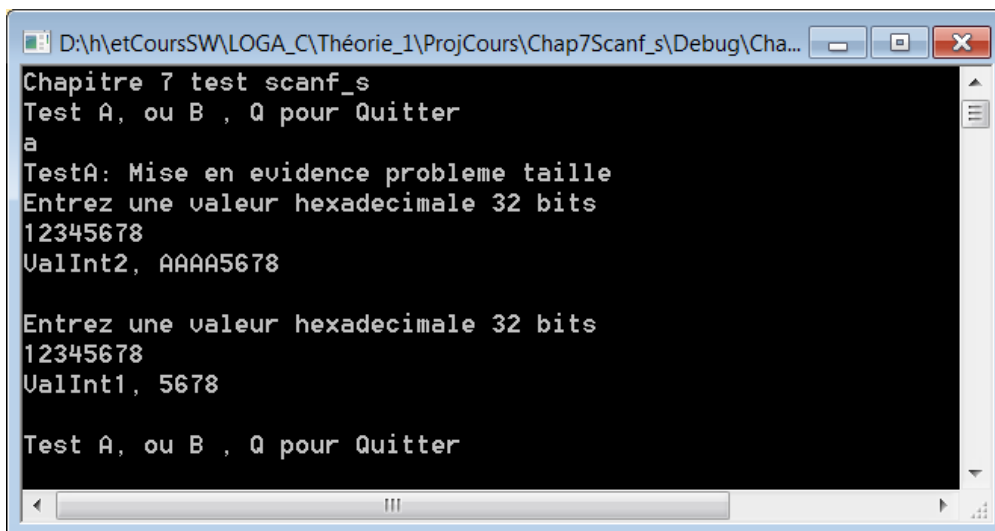
7.6.3. SCANF_S, PROBLEME DE TAILLE

Les deux éléments de programme du testA ci dessous :

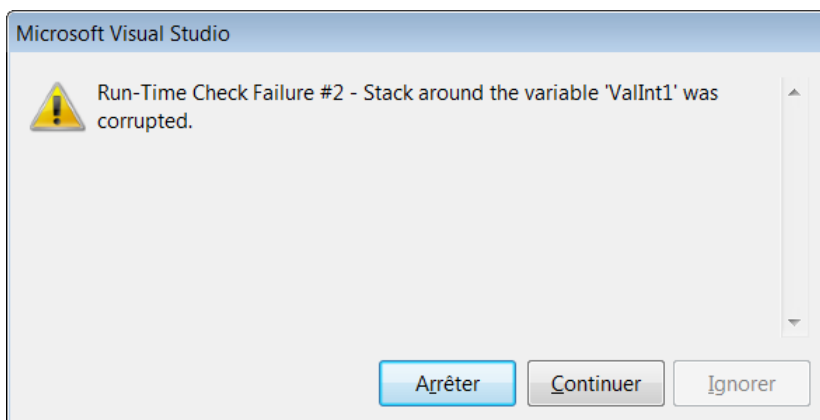
```
printf("TestA: Mise en evidence probleme taille \n");
ValInt2 = 0xAAAAAAAA;
// Saisie avec perte d'info
printf ("Entrez une valeur hexadecimale 32 bits\n");
scanf_s ("%hX%c" , &ValInt2) ;
// Affichage du résultat
printf ("ValInt2, %lX \n\n", ValInt2);

// Saisie avec problème taille destination
printf ("Entrez une valeur hexadecimale 32 bits \n");
scanf_s ("%X%c" , &ValInt1) ;
// Affichage du résultat
printf ("ValInt1, %X \n\n", ValInt1);
```

Produisent le résultat suivant:



Suivit de l'erreur suivante lorsque l'on quitte:



Donc le scanf_s dans cette situation n'apporte pas la sécurité supposée.

7.7. CONCLUSION

Ce chapitre, bien qu'incomplet, devrait permettre aux étudiants de créer leurs propres fonctions, et de par la liste des fonctions les plus courantes des bibliothèques, d'effectuer un choix et d'avoir une référence pour accéder à l'aide fournie par le compilateur.

La description des fonctions printf, scanf et scanf_s devrait permettre la réalisation de programmes comportant des affichages et des saisies.

7.8. HISTORIQUE DES VERSIONS

7.8.1. VERSION 1.0 OCTOBRE 2004

Création du document.

7.8.2. VERSION 1.1 NOVEMBRE 2007

Ajout scanf_s à cause du Visual Studio 2005.

7.8.3. VERSION 1.2 NOVEMBRE 2009

Adaptation à office 2007 et retouches orthographiques.

7.8.4. VERSION 1.3 OCTOBRE 2010

Changement de Logos. Test scanf et scanf_s avec le Visual studio 2010.

7.8.5. VERSION 1.4 OCTOBRE 2014

Ajout du numéro de module et suppression du I a théorie.