Carte de référence ANSI C

Valvassori Moïse – Revision : 1.17

Structure des programmes / Fonctions

type fnct(*type*$_1$,...)	déclaration de fonction
type nom	déclaration externe de variables
main(){	routine principale
declarations	déclaration de variables locales
actions	
}	
type fnct(*type*$_1$,...)	définition de fonction
declarations	déclaration de variables locales
actions	
return *valeur* ;	
}	
/* */	commentaires
main(int argc, char *argv[])	fonctions principales avec arguments
exit(*valeur*)	sortir du programme

Préprocesseur C

inclure un fichier de bibliothèque	#include <*fichier*>
inclure un fichier utilisateur	#include "*fichier*"
remplacer un texte	#define *nom texte*
remplacer une macro	#define *nom*(*var*) *texte*
Exemple. #define max(A,B) ((A)>(B)? (A) : (B))	
supprimer une définition	#undef *nom*
empêcher un remplacement	#
concaténer les arguments et rescanner	##
exécution conditionnelle	#if #else #elif #endif
si un *nom* est définis ou pas	#ifdef #ifndef
nom est définis	defined(*nom*)
continuation de la ligne	\

Types de donnée / Déclarations

caractère (1 octet)	char
entier	int
flottant (simple précision	float
flottant (double précision	double
mot court (entier 16 bits)	short
mot long (entier 32 bits)	long
positif et négatif	signed
seulement positif	unsigned
pointeur sur un int, float	*int *float
énumération constante	enum
valeur constante	const
déclare une variable externe	extern
variable dans un registre	register
local à ce fichier source	static
pas de valeur	void
structure	struct
créer un type de donnée	typedef *nomdutype*
taille d'un objet (renvoie un size_t)	sizeof *objet*
taille d'un type (renvoie un size_t)	sizeof(*type*)

Initialisation

initialise une variable	*type nom = valeur*
initialise un tableau	*type nom*[]={*valeur*$_1$,...}
initialise une chaîne de caractères	char *nom*[]="*chaîne*"

Constantes

long (suffixe)	L ou l
float (suffixe)	F ou f
forme exponentielle	e
octal (zéro préfixe)	0
hexadécimal (zéro x préfixe)	0x ou 0X
caractère constant (char, octal, hex)	'a', '\ooo', '\xhh'
nouvelle ligne, cr, tab, bs	\\, \?, \', \"
constante chaîne (se finit par '\0')	"abc...de"

Pointeurs, Tableaux et Structures

déclare un pointeur de type *type*	*type *nom*
déclare une fonction qui retourne le pointeur *type*	*type *f*()
déclare un pointeur de fonction	*type* (*pf)()
pointeur générique	void *
pointeur nul	NULL
objet pointé par le pointeur	**pointeur*
adresse de l'objet *nom*	&*nom*
tableau	*nom*[*dim*]
tableau multi-dimensionnel	*nom*[*dim*$_1$][*dim*$_2$]...

Structures

struct *tag* {	modèle de la structure
déclarations	déclaration des membres
};	
créer une structure	struct *tag nom*
membre d'une structure	*nom.membre*
membre d'une structure pointée	*pointeur->membre*
Exemple. (*p).x et p->x sont le même	
simple valeur, structure de type multiple	union
champs de bit avec *b* bits	*membre* :*b*

Opérateurs

membre d'une structure	*nom.membre*
membre d'une structure pointée	*pointeur->membre*
incrémentation, décrémentation	++, --
plus, moins, non logique, non bits à bits	+, -, !, ~
indirection via pointeur, adresse d'un objet	**pointeur*, &*nom*
conversion explicite	(*type*) *expr*
taille d'un objet	sizeof
multiplication, division, reste	*, /, %
addition, soustraction	+, -
décalage à gauche, à droite	<<, >>
comparaisons	>, >=, <, <=
comparaisons	==, !=
et bits à bits	&
ou exclusif bits à bits	^
ou bits à bits	|
et logique	&&
ou logique	||
expression conditionnelle	*expr*$_1$? *expr*$_2$: *expr*$_3$
assignements	+=, -+, *=, ...
séparateur d'évaluation d'expression	,

Les opérateurs unaire, les expressions conditionnelles et les opérateurs d'assignements se groupent de droite à gauche ; tous les autres se groupent de gauche à droite.

Flot de contrôle

fin d'instruction	;
délimiteur de blocs	{ }
sortie d'un switch, do, for	break
prochaine itération d'un switch, do, for	continue
aller à	goto *étiquette*
étiquette	*étiquette* :
renvoie la valeur d'une fonction	return *expr*

Constructions

instruction "*si*"	if (*expr*) *instruction* else if (*expr*) *instruction* else *instruction*
instructions "*tant que*"	while (*expr*) *instruction*
instructions "*pour*"	for (*expr*$_1$; *expr*$_2$; *expr*$_3$) *instruction*
instructions "*jusqu'à*"	do *instruction* while(*expr*) ;

instructions "*choix*"	switch (*expr*) {
	case *const*$_1$: *instruction*$_1$ break ;
	case *const*$_2$: *instruction*$_2$ break ;
	default : *instruction*
	}

Bibliothèques Standards ANSI

<assert.h>	<ctype.h>	<errno.h>	<float.h>	<limits.h>
<locale.h>	<math.h>	<setjmp.h>	<signal.h>	<stdarg.h>
<stddef.h>	<stdio.h>	<stdlib.h>	<string.h>	<time.h>

Tests de classe de caractères <ctype.h>

alphanumérique ?	isalnum(c)
alphabétique ?	isalpha(c)
caractère de contrôle ?	iscntrl(c)
chiffre décimal ?	isdigit(c)
caractère imprimable (pas les espaces) ?	isgraph(c)
lettre minuscule ?	islower(c)
caractère imprimable (avec les espaces) ?	isprint(c)
caract. imprimable sauf les espaces, les lettres et les chiffres ?	ispunct(c)
espace, retour à la ligne, tab,... ?	isspace(c)
lettre majuscule ?	isupper(c)
chiffre hexadécimal	isxdigit(c)
convertir en minuscule	tolower(c)
convertir en majuscule	toupper(c)

Opération sur les Chaînes de caractères <string.h>

s, t sont des chaîne. cs, ct sont des chaînes constantes.

longueur de s	strlen(s)
copie ct dans s	strcpy(s,ct)
jusqu'à n caractères	strncpy(s,ct,n)
concatène ct après s	strcat(s,ct)
jusqu'à n caractères	strncat(s,ct,n)
compare cs à ct	strcmp(cs,ct)
seulement les n premiers caractères	strncmp(cs,ct,n)
pointeur sur le premier c dans cs	strchr(cs,c)
pointeur sur le dernier c dans cs	strrchr(cs,c)
copie n caractères de ct dans s	memcpy(s,ct,n)
copie n caractères de ct dans s (peut se chevaucher)	memmove(s,ct,n)
compare n caractères de cs avec ct	memcmp(cs,ct,n)
pointeur sur premier c dans n premiers caractères de cs	memchr(cs,c,n)
met c dans les n premiers caractères de cs	memset(s,c,n)

Entrée/Sortie <stdio.h>

Standard I/O

entrée standard	`stdin`
sortie standard	`stdout`
sortie erreur	`stderr`
fin de fichier	`EOF`
lire un caractère	`getchar()`
écrire un caractère	`putchar(`*chr*`)`
écriture formatée	`printf("`*format*`",`arg_1`, ...)`
écrire dans la chaîne `s`	`sprintf(s,"`*format*`",`arg_1`, ...)`
lecture formatée	`scanf("`*format*`",&`nom_1`, ...)`
lecture dans la chaîne `s`	`sscanf("`*format*`",&`nom_1`, ...)`
lire une lire dans `s` (< `max` car.)	`gets(s,max)`
affiche la chaîne `s`	`puts(s)`

Fichiers

déclare un pointeur de fichier	`FILE * `*fp*
ouvre un fichier	`fopen("`*fichier*`" "`*mode*`")`
modes : `r` : lecture ; `w` : écriture ; `a` : ajout	
lit un caractère	`getc(`*fp*`)`
écrit un caractère	`putc(`*chr*`, `*fp*`)`
écrit dans le fichier	`fprintf(`*fp*`,"`*format*`",`arg_1`, ...)`
lit dans le fichier	`fscanf(`*fp*`,"`*format*`",`arg_1`, ...)`
ferme le fichier	`fclose(`*fp*`)`
$\neq 0$ si erreur	`ferror(`*fp*`)`
$\neq 0$ si fin du fichier	`feof(`*fp*`)`
lit une ligne `s` du fichier (long < `max`)	`fgets(`*s*`,`*max*`,`*fp*`)`
écrit la chaîne `s`	`fputs(`*s*`,`*fp*`)`

Code pour les I/O formattée

codes de la forme : "`%-+ 0`*w*`.`*pmc*"

`-`	justification à gauche
`+`	afiche le signe
espace	affiche un espace à la place du signe
`0`	complète avec des zéros
`w`	largeur minimum
`p`	précision
`m`	caractère de conversion `h` short ; `l` long ; `L` long double
`c`	caractère de conversion :

`d,i`	entier	`unsigned`	non signé
`c`	caractère seul	`s`	chaîne de caractère
`f`	double	`e,E`	exponentiel
`o`	octal	`x,X`	hexadécimal
`p`	pointeur	`n`	nombre de caratère écrit

`g,G` comme `f` ou `e,E` suivant le contexte.

Liste d'arguments variables <stdargs.h>

déclaration de pointeur vers les arguments	`va_list `*nom* `;`
initialisation de la liste d'argument	`va_start(`*nom*`, `*darg*`)`
darg est le nom du dernier argument de la fonction	
accede au prochain arg et met le pointeur	`va_arg(`*nom*`, `*type*`)`
appel avant de sortir de la fonction	`va_end(`*nom*`)`

Utilitaires Standard <stdlib.h>

valeur absolue entière	`abs(`*n*`)`
valeur absolue entier long	`labs(`*n*`)`
nombre pseudo aléatoire `[0,RAND_MAX]`	`rand()`
initialise la graine aléatoire	`srand(`*n*`)`
termine l'execution du programme	`exit(`*status*`)`
execute la chaîne *c*	`system(c)`

Conversion

convertit la chaîne vers un double	`atof(c)`
convertit la chaîne vers un entier	`atoi(c)`
convertit la chaîne vers un long	`atol(c)`
convertit le prefix de *c* vers un double	`strtod(c,l)`
convertit le prefix de *c* en base *b* vers un long	`strtol(c,l,b)`
convertit le prefix de *c* en base *b* vers un ulong	`strtoul(c,l,b)`

Allocation de mémoire

allocation de mémoire	`malloc(`*taille*`), calloc(`*nobj*`,`*taille*`)`
change la taille d'un bloc	`realloc(`*ptr*`,`*taille*`)`
libère la mémoire	`free(`*ptr*`)`

Manipulation de tableau

cherche *clé* dans *tab*	`bsearch(`*clé*`,`*tab*`,`*n*`,`*taille*`,`*cmp*`())`
tri tableau dans ordre croissant	`qsort(`*tab*`,`*n*`,`*taille*`,`*cmp*`())`

Gestion du temps <time.h>

temps processeur est utilisé par	`clock()`
Exemple. `clock()\CLOCKS_PER_SEC`	heure en seconde
heure courrante	`time()`
$\texttt{temps}_2\texttt{-temps}_1$ en seconde (*double*)	`difftime(`$temps_2$`,`$temps_1$`)`
type arithmétique représentant le temps	`clock_t,time_t`
structure pour des calcule de temps	`tm`

`tm_sec`	secondes après la minute
`tm_min`	minutes après heure
`tm_hour`	heures depuis minuit
`tm_mday`	jour du mois
`tm_mon`	mois depuis janvier
`tm_years`	années depuis 1900
`tm_wday`	jours depuis dimanche
`tm_yday`	jours depuis 1 janvier

convertit l'heure locale en calendrier	`mktime(`*tp*`)`
convertit un temps en `tp` en chaîne	`asctime(`*tp*`)`
convertit le calendrier en heure locale	`ctime(`*tp*`)`
convertit le calendrier en heure GMT	`gmtime(`*tp*`)`
convertit le calendrier en heure locale	`localtime(`*tp*`)`
formate les info de date et heure	`strftime(`*s*`,`*max*`,"`*format*`",`*tp*`)`

tp est un pointeur sur une structure de type `tm`

Fonctions Mathématique <math.h>

Les arguments et les valeurs de retour sont des `double`.

fonctions trigo	`cos(`*x*`), sin(`*x*`), tan(`*x*`)`
fonctions trigo inverse	`acos(`*x*`), asin(`*x*`), atan(`*x*`)`
arctan$\frac{y}{x}$	`atan2(y,x)`
fonctions trigo hyperbolique	`cosh(`*x*`), sinh(`*x*`), tanh(`*x*`)`
exponentielle et log	`exp(`*x*`), log(`*x*`), log10(`*x*`)`
x^n et $e = \log_2 x$	`ldexp(`*x*`,`*n*`), frexp(`*x*`,*`*n*`)`
division et reste	`modf(`*x*`,*`*ip*`), fmod(`*x*`,`*y*`)`
puissance	`pow(`*x*`,`*y*`), sqrt(`*x*`)`
arrondis	`ceil(`*x*`),floor(`*x*`),fabs(`*x*`)`

Limites des Entiers <limits.h>

`CHAR_BIT`	nb de bits de `char`	(8)
`CHAR_MAX`	valeur max de `char`	(127 ou 255)
`CHAR_MIN`	valeur min de `char`	(-128 ou 0)
`INT_MAX`	valeur max de `int`	(+32.767)
`INT_MIN`	valeur min de `int`	(-32.768)
`LONG_MAX`	valeur max de `long`	(+2.147.482.647)
`LONG_MIN`	valeur min de `long`	(-2.147.482.648)
`SCHAR_MAX`	valeur max de `signed char`	(127)
`SCHAR_MIN`	valeur min de `signed char`	(-128)
`SHRT_MAX`	valeur max de `short`	(+32.767)
`SHRT_MIN`	valeur min de `short`	(-32.768)
`UCHAR_MAX`	valeur max de `unsigned char`	(255)
`UINT_MAX`	valeur max de `unsigned int`	(65.535)
`ULONG_MAX`	valeur max de `unsigned long`	(4.294.967.295)
`USHRT_MAX`	valeur max de `unsigned short`	(65.535)

Limites des Flottants <float.h>

`FLT_RADIX`	base de représentation de l'exposant	(2)
`FLT_ROUND`	mode d'arrondis	()
`FLT_DIG`	nb de décimale de précision	(6)
`FLT_EPSILON`	ppt x tq $1.0 + x \neq 1.0$	(10^{-5})
`FLT_MANT_DIG`	nb de décimale de mantisse	(6)
`FLT_MAX`	nb flottant max	(10^{37})
`FLT_MAX_EXP`	exposant max	()
`FLT_MIN`	nb flottant min	(10^{-37})
`FLT_MIN_EXP`	exposant min	()
`DBL_DIG`	nb de décimale de précision	(10)
`DBL_EPSILON`	ppt x tq $1.0 + x \neq 1.0$	(10^{-9})
`DBL_MANT_DIG`	nb de décimale de mantisse	(6)
`DBL_MAX`	nb double max	(10^{37})
`DBL_MAX_EXP`	exposant max	()
`DBL_MIN`	nb double min	(10^{-37})
`DBL_MIN_EXP`	exposant min	()

C++

Classes, Héritage

```
class maclasse : superclasse1,superclasse2{
  public:
  protected:
  private:
};
```

creation	*Object o* `= new` *Object()*
destruction	`delete` *o*

Porté

accessible partout	`public`
accessible classe et sous classes	`protected`
accessible dans la classe	`private`

Flux

affiche sortie standard	`cout <<` val_1 `<<` val_2 `...`
lit entréé standard	`cin >>` val_1 `>>` val_2 `...`