C Cheat Sheet Ensimag ©2017

Auteurs :
F. Broquedis, M. Chabanas,
T. Diasalves, P. Kocelniak,
O. Muller, F. Wagner

Options de compilation de base

`clang prog.c <options>`	Compile le code source prog.c
`-o prog`	Fichier executable : prog
`-c -o prog.o`	Fichier objet : prog.o sans édition de liens
`-Wall -Wextra`	Tous les warnings
`-std=c99`	Standard C99
`-g`	Debug
`-fsanitize=address -fsanitize=undefined -fno-omit-frame-pointer -O0`	Fsanitize
`-l<nom librairie>`	Lib statique standard
-lm	Ex. : lib. math. `libm`
`clang -o prog prog1.o prog2.o`	Edition de liens

Options de compilation avancées

`-S -o prog.s`	Fichier assembleur (prog.s)
`-E -o prog.pre`	Sortie préprocesseur
`-DVAL`	Variable macro VAL
# lib.o : code objet à convertir en libLS.a	Lib. statique personnelle:
`ar -scr libLS.a lib.o`	Archivage
`clang test.o libLS.a -o prog`	Edition de liens
# lib.o : code objet à convertir en libLD.so	Lib dynamique pers. :
`clang -shared lib.o -o libLD.so`	Génération
`clang test.o -L. -lLD -o prog`	Edition de liens
`-L<path> -l<nom librairie>`	Librairie via un path

Débogage : méthode

Warnings de compilation et **mode debug** :

```
clang prog.c -Wall -Wextra -std=c99 -g -o prog
```

Sortie **préprocesseur** (macros uniquement) :

```
clang prog.c -E -o prog.pre
```

Débogueur **fsanitize** (code débogage inclus dans le code exécutable) :

```
clang prog.c -Wall -Wextra -std=c99 -g
-o prog_fsanitize -fsanitize=address
-fsanitize=undefined -fno-omit-frame-pointer -O0
```

Débogueur **valgrind** (mémoire) :

```
valgrind ./prog
```

Débogueur **gdb/ddd** :

```
ddd ./prog
```

Portée des variables et fonctions

`extern <variable>`	Variable définie dans un autre module
`static <fonction>`	Fonction locale au fichier source
`void fct() { static <variable>`	<variable> est locale à la fonction fct, avec valeur persistante

Types de données de base

`short int, int, long int, long long int`	Entiers signés
`unsigned short int, ..., unsigned long long int`	Entiers non signés
`float, double, long double`	Flottants
`char`	Caractère
`bool`	Booléen (<stdbool.h>)
`<type> <nomtab>[<taille>]`	Tableau statique de <taille> cases
`char chaine[50]`	Ex. : tableau de 50 caractères
`int32_t tab[10][20]`	Ex. : tableau 10 lignes x 20 colonnes

Types de données entiers de taille explicite (stdint.h)

`uint8_t, uint16_t, uint32_t, uint64_t`	Entiers non signés (8, 16, 32, 64 bits)
`int8_t, int16_t, int32_t, int64_t`	Entiers signés (8, 16, 32, 64 bits)

Types de données : structure

`struct matrix {`	Déclaration de structure :
`  uint32_t vertex_x;`	2 champs valeurs : vertex_x et _y
`  uint32_t vertex_y;`	1 champ pointeur sur suivant :
`  struct matrix *next;`	next
`};`	
`struct matrix neo;`	Déclaration de variable (neo)

Types de données : énumération

`enum type_piece { PION, ROI, FOU };`	Déclaration
`enum type_piece piece = FOU;`	Exemple

Types de données : transtypage (cast)

`(<type de donnée>)<variable>;`	Forçage du type
`int32_t e = 20;`	Exemple
`double res = 1 / (double) e;`	

Types de données : constantes

`const <type> <var> = <value>;`	Définition
`const float pi = 3.14159;`	Ex : flottant
`const char chaine = "Bonjour";`	Ex : chaîne de caractères
`0xF0`	Ex : hexadécimal
`0b11110000`	Ex : binaire

Types de données avancés : typedef

`typedef struct matrix matrice;`	Définition de type
`matrice neo;`	Déclaration de variable

Types de données avancés : union

`union valeur {`	Définition
`  int32_t entier;`	
`  float flottant;`	
`};`	
`union valeur *val;`	Déclaration de variable

Structures de contrôle : test

```
if (<condition1>) {
   <instructions1>
} else if (<condition2>) {
   <instructions2>
} else {
   <instructions3>
}
```

Structures de contrôle : boucles

```
// for (<initialiser>; <tester>; <post traiter>) {
for (int i = 0; i < 10; i++) {
   <instructions>
}
while (<condition>) {
   <instructions>
}
do {
   <instructions>
} while (<condition>);
```

Structures de contrôle : switch

```
switch (<variable>) {
case <constante1>:
   <instructions1>
   break;
case <constante2>:
   <instructions2>
   break;
default :
   <instructions>
   break;
}
```

Structures de contrôle : break, continue

`break;`	Arrêt de la première instruction for, while, do englobante
`continue;`	Arrêt de l'itération courante (dans un for, while, do) et passage à l'itération suivante.

Commentaires

`/* Commentaire sur plusieurs lignes possible */`	Commentaire de bloc
`// Commentaire sur une ligne`	Commentaire de ligne

Entrées / Sorties standards (<stdio.h>)

`int getchar(void);`	Lecture d'un caractère
`car = getchar();`	Exemple
`int putchar(int);`	Affichage d'un caractère
`putchar('a');`	Exemple
`char *fgets(char *str, int num, stdin)`	Lecture d'une ligne
`fgets(chaine, 100, stdin);`	Exemple
`int puts(const char *str);`	Affichage d'une ligne
`puts(string);`	Exemple

Entrées / Sorties standards, suite (<stdio.h>)

`int printf(const char *format, ...);`	Affichage formatté
`printf("i=%d x=%f\n", i, x);`	Exemple
`int scanf(const char *format, ...);`	Entrée formattée depuis le clavier
`scanf("%d %f", &i, &x);`	Exemple
`int sscanf(const char *s, const char *format, ...);`	Entrée formattée depuis une chaîne de caractères
`sscanf(chaine,"%d %f", &i, &x);`	Exemple
`perror("Fichier inconnu");`	Ecriture sortie erreur

Ouverture / Fermeture fichiers (<stdio.h>)

`FILE *fopen(const char *filename, const char *mode);`	Ouverture de fichier r : read, w : write t : text, b : binary
`FILE *fic;` `fic=fopen("file.txt", "rt");`	Exemple
`int fclose(FILE *stream);`	Fermeture de fichier
`fclose(fic);`	Exemple

Entrées / Sorties fichiers, mode texte (<stdio.h>)

`int fscanf(FILE *fic, const char *format, ...);`	Lecture formattée
`fscanf(fic, "%d %f", &i,&x);`	Exemple
`int fprintf(FILE *fic, const char *format, ...);`	Ecriture formattée
`fprintf(fic, "i=%d x=%f\n", i, x);`	Exemple
`char *fgets(char *str, int num, FILE *fic);`	Lecture d'une ligne
`fgets(chaine, 100, fic);`	Exemple
`int fputs(const char *str, FILE *fic);`	Ecriture d'une ligne
`fputs(chaine, fic);`	Exemple
`car = fgetc(fic);`	Lecture d'un caractère
`fputc('A', fic);`	Ecriture d'un caractère

Entrées / Sorties fichiers, mode binaire (<stdio.h>)

`size_t fread(void *ptr, size_t size, size_t count, FILE *fic);`	Lecture binaire
`result=fread(&x, sizeof(float), 1, fic);`	Exemple
`size_t fwrite (const void *ptr, size_t size, size_t count, FILE *fic);`	Ecriture binaire
`float ftab[5]={1.,2.,3.,4.,5.};` `result = fwrite(ftab, sizeof(float), 5, fic);`	Exemple
`int fseek (FILE *fic, long int offset, int origin);`	Positionnement binaire
`SEEK_SET (début) SEEK_END (fin) SEEK_CUR (courant)`	origin
`fseek(fic, 9 * sizeof(double), SEEK_SET);`	Exemple

Pointeurs et adresses de base <stdlib.h>

`<type> *<variable>;`	Déclaration de pointeur
`&var`	Adresse de var
`*pvar`	Valeur pointée par pvar
`sizeof(<type>)`	Retourne la taille mémoire d'un type en octets
`void *malloc(size_t size);`	Allocation de size octets
`void *calloc(size_t num, size_t size);`	Allocation avec mise à 0

`float *f1, *f2;`
`f1 = calloc(5, sizeof(float));`
`f2 = malloc(5 * sizeof(float));`
	Ex. : Allocation mémoire de 5 * taille d'un float

`void free(void *ptr);`	Libération de mémoire
`free(f1);`	Exemple

Pointeurs et adresses avancés <string.h>

`void *memcpy(void *destination, const void *source, size_t num);`	Copie de zone mémoire
`memcpy(f1, f2, 5 * sizeof(float));`	Exemple

Pointeurs et adresses avancés

`int32_t *tab[10];`	Tableau de 10 pointeurs vers des entiers
`int32_t **tab;`	Pointeurs de pointeurs

`int (*pf1) (void);`
`double (*pf2) (double, double);`
`void* (*pf2) (void*, void*);`
	Pointeurs de fonctions

Opérateur d'affectation

`=`	Affectation

Opérateurs arithmétiques

`+ , - , * , /`	Add, Soust, Mult, Div
`%`	Modulo

Opérateurs de comparaison

`> , >= , < , <=`	
`==`	Egal
`!=`	Différent

Opérateurs d'incrémentation

`++`	Incrémentation
`--`	Décrémentation

Opérateurs logiques booléens

`&&`	et logique
`||`	ou logique
`!`	non logique

Opérateurs logiques bit à bit

`&`	et
`|`	ou inclusif
`^`	ou exclusif
`~`	complément à 1
`<<`	décalage à gauche
`>>`	décalage à droite

Opérateurs structures et unions

`.`	`<variable>.<champ>` `neo.vertex_x`	<variable> != adresse Exemple
`->`	`<pointeur>-><champ>` `next->vertex_y`	<pointeur> = adresse Exemple

Opérateurs d'affectation composée

`+= -= *= /= %= &= ^= |= <<= >>=`	Opérateurs
`i += 1;`	Exemple : i=i+1

Opérateur conditionnel

`m = (a > b) ? a : b;`	Affecte a à m si a > b, sinon b

Macros #include #define

`#include <stdio.h>`	Inclusion fichier entête système
`#include "fonctions.h"`	Inclusion fichier entête personnel
`#define PI 3.14159`	Constante (préprocesseur)

Chaînes de caractères <string.h>

`char *strncpy(char *dest, char *source, size_t num)`	Copie de chaînes de caractères
`char *strncat(char *dest, char *source, size_t num)`	concaténation
`int strncmp(const char *str1, const char *str2, size_t num)`	Comparaison. 0 : égalité >0, <0 : str1 est > ou < alphabétiquement
`size_t strlen(const char *str);`	Longueur d'une chaine
`int atoi(const char *str);`	Conversion chaîne vers entier
`long int atol(const char *str);`	Conversion chaîne vers entier long
`double atof(const char* str);`	Conversion chaîne vers flottant

Fichier entête : exemple standard "fonctions.h"

```
#ifndef _FONCTIONS_H_
#define _FONCTIONS_H_
struct produit {
   uint16_t quantite;
   double prix;
};
double calculer_total(struct produit prod);
#endif /* _FONCTIONS_H_ */
```

Makefile (exec: make, test: make -n, debug: make -d)

```
CC=clang
CFLAGS=-std=c99 -Wall -Wextra -g
LDFLAGS=-lm
EXEC=prog
all:  $(EXEC)
prog:  fonctions.o main.o
     $(CC) -o $@ $^ $(LDFLAGS)
main.o:  fonctions.h
%.o:  %.c
     $(CC) -o $@ -c $< $(CFLAGS)
clean:
     rm -f *.o $(EXEC)
```