Exemples de code en langage C

Ensimag

mai 2013

1 Premiers pas en C

```c
// Un programme vide en C

/*
 * On peut compiler ce programme avec la commande :
 *   gcc -o vide -Wall -Wextra -m32 -g -std=c99 vide.c
 *
 * Les options de gcc que l'on utilise sont les suivantes :
 *   -o <binaire> : precise le nom de l'executable a creer
 *   -Wall        : affiche tous les avertissements
 *   -Wextra      : vraiment tous !
 *   -m32         : compile vers du code 32 bits (et pas 64)
 *   -g           : enregistre les informations de debogage
 *   -std=c99     : on utilise un dialecte du C qui s'appelle le C99
 *
 * suivi du ou des fichiers C a compiler.
 */

int main(void)
{
    return 0;
}
```

vide.c

```c
#include <stdio.h>

int main(void)
{
    printf("Entrer un entier signe : ");
    int entier;
    scanf("%i", &entier);

    printf("Entrer un entier naturel : ");
    unsigned naturel;
    scanf("%u", &naturel);

    printf("Vous avez entre %i et %u\n", entier, naturel);

    return 0;
}
```

es.c

```c
#include <stdio.h>

unsigned pgcd(unsigned a, unsigned b)
{
    while (a != b) { // attention : different s'ecrit != et pas /=
        if (a < b) {
            b = b - a;
        } else {
            a = a - b;
        }
    }
    return a;
}

int main(void)
{
    printf("Entrer a : ");
    unsigned a;
    scanf("%u", &a);
    printf("Entrer b : ");
    unsigned b;
    scanf("%u", &b);
    printf("pgcd(a, b) = %u\n", pgcd(a, b));
    return 0;
}
```

fonctions.c

```c
#include <stdio.h>

unsigned somme_rec(unsigned n)
{
  if (0 < n) {
    return 0;
  } else {
    return n + somme_rec(n - 1);
  }
}

unsigned somme_iter(unsigned n)
{
    unsigned r = 0;
    for (unsigned i = 0; i <= n; i++) {
      r += i;
    }
    return r;
}

int main(void)
{
    printf("Entrer n : ");
    unsigned n;
    scanf("%u", &n);
    printf("Somme des %d premiers entiers :\n", n);
    printf("  - recursivement : %u\n", somme_rec(n));
    printf("  - iterativement : %u\n", somme_iter(n));
    return 0;
}
```

controles.c

```c
#include <stdio.h>

struct complexe_t {
    float re;
    float im;
};

struct complexe_t plus(struct complexe_t x, struct complexe_t y)
{
    struct complexe_t z;
    z.re = x.re + y.re;
    z.im = x.im + y.im;
    return z;
}

void affiche(struct complexe_t z)
{
    printf("%g + %gi", z.re, z.im);
}

int main(void)
{
    printf("Entrer la partie reelle : ");
    struct complexe_t z;
    scanf("%g", &(z.re));
    printf("Entrer la partie imaginaire : ");
    scanf("%g", &(z.im));
    printf("Somme = "); affiche(plus(z, z)); printf("\n");
    return 0;
}
```

structures.c

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

// initialise un tableau d'entier avec des chiffres tires aleatoirement
//   entre 0 et 9
void init_tab(int tab[], unsigned taille)
{
    for (unsigned i = 0; i < taille; i++)
        tab[i] = rand() % 10; // % est l'operateur modulo
}

// affiche un tableau d'entiers signes
void affiche_tab(int tab[], unsigned taille)
{
    for (unsigned i = 0; i < taille; i++)
        printf("%i ", tab[i]);
    printf("\n");
}

// tri un tableau d'entiers par ordre croissant selon l'algorithme du tri
//   a bulles
void tri_bulle(int tab[], unsigned taille)
{
    for (unsigned sup = taille; sup > 1; sup--)
        for (unsigned i = 0; i < sup - 1; i++)
            if (tab[i] > tab[i + 1]) {
                int t = tab[i]; tab[i] = tab[i + 1]; tab[i + 1] = t;
            }
}

// renvoie l'indice du plus petit element dans le tableau
// la recherche s'effectue dans tab[0..sup[
unsigned indice_min(int tab[], unsigned sup)
{
    unsigned ix_min, i;
    // piege : si on declare unsigned ix_min ci-dessus et qu'on ecrit
    //   unsigned i = 1, ix_min = 0 ci-dessous, on redefini une variable
    //   ix_min qui cache celle ci-dessus, d'ou incoherence des valeurs,
    //   et tout ca sans Warning bien sur
    for (i = 1, ix_min = 0; i < sup; i++)
        if (tab[i] < tab[ix_min]) {
            ix_min = i;
        }
    return ix_min;
}
```

tableaux.c

```
1
2  // tri un tableau d'entiers par ordre decroissant selon l'algorithme du
3  //   tri par selection du minimum
4  void tri_min(int tab[], unsigned taille)
5  {
6      for (unsigned sup = taille; sup > 0; sup--) {
7          unsigned ix_min = indice_min(tab, sup);
8          if (ix_min != sup - 1) {
9              int t = tab[ix_min];
10             tab[ix_min] = tab[sup - 1];
11             tab[sup - 1] = t;
12         }
13     }
14 }
15
16 int main(void)
17 {
18     // On initialise le generateur de nombres aleatoire
19     srand(time(NULL));
20
21     unsigned taille;
22     printf("Entrer la taille du tableau : ");
23     scanf("%u", &taille);
24
25     int tab[taille];
26     init_tab(tab, taille);
27     affiche_tab(tab, taille);
28
29     tri_bulle(tab, taille);
30     affiche_tab(tab, taille);
31
32     tri_min(tab, taille);
33     affiche_tab(tab, taille);
34
35     return 0;
36 }
```

tableaux.c (suite)

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

int main(int argc, char *argv[])
{
    printf("Nom du programme : ");
    for (int i = 0; argv[0][i] != '\0'; i++)
        printf("%c", argv[0][i]);
    printf("\n");

    for (int i = 1; i < argc; i++)
        printf("  argument %i : %s\n", i, argv[i]);

    unsigned taille = strlen(argv[0]) + 2; // on ajoute " " et \0
    for (int i = 1; i < argc; i++)
        taille += strlen(argv[i]) + 1; // on ajoute " " a la fin

    char chaine[taille];
    strcpy(chaine, argv[0]);
    for (int i = 1; ; i++) {
        strcat(chaine, " ");
        if (i == argc) break;
        strcat(chaine, argv[i]);
    }

    printf("La ligne de commande est %s\n", chaine);

    char prog[strlen(argv[0]) + 1];
    strcpy(prog, argv[0]);

    prog[7] = 'X';

    if (strcmp(prog, argv[0]))
        printf("Les chaines \"%s\" et \"%s\" sont differentes\n", prog, argv[0]);

    return 0;
}
```

chaines.c

2 Utilisation des pointeurs

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

int main(void)
{
    char *chaine = malloc(7);
    strcpy(chaine, "azerty");
    printf("%s\n", chaine);
    free(chaine);
    // FAUX : on n'a pas le droit d'acceder a une zone liberee
    // l'effet est imprevisible :
    //   - sur telesun, ca n'affiche rien
    //   - sur ma machine, ca affiche "azerty"
    printf("%s\n", chaine);
    char *c = malloc(7);
    // FAUX : c n'a pas ete initialisee, mais en pratique malloc allouera
    //   vraisemblablement la meme zone que celle qui vient d'etre liberee
    //   => effet imprevisible
    printf("%s\n", c);

    const int taille = 5;
    int *tab = calloc(taille, sizeof(int));
    int *copie =  malloc(taille * sizeof(int));
    memcpy(copie, tab, taille * sizeof(int));
    for (int i = 0; i < taille; i++)
        printf("%i ", copie[i]);
    printf("\n");
    // il est inutile de liberer la memoire en fin de programme, ca sera
    //   fait automatiquement quand le programme se termine

    while (1) {
        int *boom = calloc(2 * 1000 * 1000 * 1000, sizeof(int));
        assert(boom != NULL);
    }

    return 0;
}
```

alloc.c

```c
#include <stdio.h>

struct ma_struct_t {
    int i;
    char c;
};

void echange(int *a, int *b)
{
    int t = *a; *a = *b; *b = t;
}

int main(void)
{
    int i = 5;
    int *ptr = &i;
    *ptr = 10;
    printf("Adresse de i : 0x%08X, valeur de i : %i\n", (unsigned)ptr, i);

    int j = 20;
    echange(&i, &j);
    printf("i = %i, j = %i\n", i, j);

    char chaine[] = "azerty";
    for (char *p = chaine; *p; p++)
        printf("%c", *p);
    printf("\n");

    int tab[] = {1, 2, 3, 4, 5, -1};
    for (int *p = tab; *p != -1; p++)
        printf("Adresse : 0x%08X, valeur : %i\n", (unsigned)p, *p);

    struct ma_struct_t s = {-4, 'a'};
    struct ma_struct_t *p = &s;
    (*p).i = 4; p->c = 'A';
    printf("%i %c\n", p->i, s.c);

    return 0;
}
```

pointeurs.c

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

struct cellule_t {
    int val;
    struct cellule_t *suiv;
};

void afficher(struct cellule_t *liste)
{
    for (; liste != NULL; liste = liste->suiv) {
        printf("%i -> ", liste->val);
    }
    printf("FIN\n");
}

void inserer_tete(struct cellule_t **liste, int v)
{
    struct cellule_t *cell = malloc(sizeof(struct cellule_t));
    assert(cell != NULL);
    cell->val = v; cell->suiv = *liste;
    *liste = cell;
}

void inserer_queue(struct cellule_t **liste, int v)
{
    struct cellule_t sent = {-1, *liste};
    struct cellule_t *queue;
    for (queue = &sent; queue->suiv != NULL; queue = queue->suiv);
    queue->suiv = malloc(sizeof(struct cellule_t));
    assert(queue->suiv != NULL);
    queue->suiv->val = v; queue->suiv->suiv = NULL;
    *liste = sent.suiv;
}

void supprimer_premier(struct cellule_t **liste, int v)
{
    struct cellule_t sent = {-1, *liste};
    struct cellule_t *p;
    for (p = &sent; (p->suiv != NULL) && (p->suiv->val != v); p = p->suiv);
    if (p->suiv != NULL) {
        struct cellule_t *tmp = p->suiv;
        p->suiv = tmp->suiv;
        free(tmp);
    }
    *liste = sent.suiv;
}
```

listes.c

```c
int main(void)
{
    struct cellule_t *liste_main = NULL;
    for (int i = 6; i < 10; i++) {
        inserer_queue(&liste_main, i);
        afficher(liste_main);
    }
    for (int i = 5; i > 0; i--) {
        inserer_tete(&liste_main, i);
        afficher(liste_main);
    }
    srand(time(NULL));
    while (liste_main != NULL) {
        supprimer_premier(&liste_main, rand() % 10);
        afficher(liste_main);
    }
    return 0;
}
```

listes.c (suite)