

# Exemples de code en langage C

Ensimag

mai 2012

## 1 Premiers pas en C

```
1 // Un programme vide en C
2
3 /*
4  * On peut compiler ce programme avec la commande :
5  *   gcc -o vide -Wall -Wextra -m32 -g -std=c99 vide.c
6  *
7  * Les options de gcc que l'on utilise sont les suivantes :
8  *   -o <binaire> : precise le nom de l'executable a creer
9  *   -Wall       : affiche tous les avertissements
10 *   -Wextra      : vraiment tous !
11 *   -m32         : compile vers du code 32 bits (et pas 64)
12 *   -g           : enregistre les informations de debogage
13 *   -std=c99     : on utilise un dialecte du C qui s'appelle le C99
14 *
15 * suivi du ou des fichiers C a compiler.
16 */
17
18 int main(void)
19 {
20     return 0;
21 }
```

vide.c

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("Entrer un entier signe : ");
6     int entier;
7     scanf("%i", &entier);
8
9     printf("Entrer un entier naturel : ");
10    unsigned naturel;
11    scanf("%u", &naturel);
12
13    printf("Vous avez entre %i et %u\n", entier, naturel);
14
15    return 0;
16 }
```

es.c

```

1 #include <stdio.h>
2
3 unsigned pgcd(unsigned a, unsigned b)
4 {
5     while (a != b) { // attention : different s'ecrit != et pas /=
6         if (a < b) {
7             b = b - a;
8         } else {
9             a = a - b;
10        }
11    }
12    return a;
13 }
14
15 int main(void)
16 {
17     printf("Entrer a : ");
18     unsigned a;
19     scanf("%u", &a);
20     printf("Entrer b : ");
21     unsigned b;
22     scanf("%u", &b);
23     printf("pgcd(a, b) = %u\n", pgcd(a, b));
24     return 0;
25 }

```

fonctions.c

```

1 #include <stdio.h>
2
3 unsigned somme_rec(unsigned n)
4 {
5     if (0 < n) {
6         return 0;
7     } else {
8         return n + somme_rec(n - 1);
9     }
10 }
11
12 unsigned somme_iter(unsigned n)
13 {
14     unsigned r = 0;
15     for (unsigned i = 0; i <= n; i++) {
16         r += i;
17     }
18     return r;
19 }
20
21 int main(void)
22 {
23     printf("Entrer n : ");
24     unsigned n;
25     scanf("%u", &n);
26     printf("Somme des %d premiers entiers :\n", n);
27     printf("  - recursivement : %u\n", somme_rec(n));
28     printf("  - iterativement : %u\n", somme_iter(n));
29     return 0;
30 }

```

contrôles.c

```

1 #include <stdio.h>
2
3 struct complexe_t {
4     float re;
5     float im;
6 };
7
8 struct complexe_t plus(struct complexe_t x, struct complexe_t y)
9 {
10     struct complexe_t z;
11     z.re = x.re + y.re;
12     z.im = x.im + y.im;
13     return z;
14 }
15
16 void affiche(struct complexe_t z)
17 {
18     printf("%g + %gi", z.re, z.im);
19 }
20
21 int main(void)
22 {
23     printf("Entrer la partie reelle : ");
24     struct complexe_t z;
25     scanf("%g", &(z.re));
26     printf("Entrer la partie imaginaire : ");
27     scanf("%g", &(z.im));
28     printf("Somme = "); affiche(plus(z, z)); printf("\n");
29     return 0;
30 }

```

structures.c

```

1  #include <time.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  // initialise un tableau d'entier avec des chiffres tires aleatoirement
6  // entre 0 et 9
7  void init_tab(int tab[], unsigned taille)
8  {
9      for (unsigned i = 0; i < taille; i++)
10         tab[i] = rand() % 10; // % est l'operateur modulo
11 }
12
13 // affiche un tableau d'entiers signes
14 void affiche_tab(int tab[], unsigned taille)
15 {
16     for (unsigned i = 0; i < taille; i++)
17         printf("%i ", tab[i]);
18     printf("\n");
19 }
20
21 // tri un tableau d'entiers par ordre croissant selon l'algorithme du tri
22 // a bulles
23 void tri_bulle(int tab[], unsigned taille)
24 {
25     for (unsigned sup = taille; sup > 1; sup--)
26         for (unsigned i = 0; i < sup - 1; i++)
27             if (tab[i] > tab[i + 1]) {
28                 int t = tab[i]; tab[i] = tab[i + 1]; tab[i + 1] = t;
29             }
30 }
31
32 // renvoie l'indice du plus petit element dans le tableau
33 // la recherche s'effectue dans tab[0..sup[
34 unsigned indice_min(int tab[], unsigned sup)
35 {
36     unsigned ix_min, i;
37     // piege : si on declare unsigned ix_min ci-dessus et qu'on ecrit
38     // unsigned i = 1, ix_min = 0 ci-dessous, on redefini une variable
39     // ix_min qui cache celle ci-dessus, d'ou incoherence des valeurs,
40     // et tout ca sans Warning bien sur
41     for (i = 1, ix_min = 0; i < sup; i++)
42         if (tab[i] < tab[ix_min]) {
43             ix_min = i;
44         }
45     return ix_min;
46 }

```

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)

```

1  #include <stdio.h>
2  #include <string.h>
3
4  int main(int argc, char *argv[])
5  {
6      printf("Nom du programme : ");
7      for (int i = 0; argv[0][i] != '\0'; i++)
8          printf("%c", argv[0][i]);
9      printf("\n");
10
11     for (int i = 1; i < argc; i++)
12         printf("    argument %i : %s\n", i, argv[i]);
13
14     unsigned taille = strlen(argv[0]) + 2; // on ajoute " " et \0
15     for (int i = 1; i < argc; i++)
16         taille += strlen(argv[i]) + 1; // on ajoute " " a la fin
17
18     char chaine[taille];
19     strcpy(chaine, argv[0]);
20     for (int i = 1; ; i++) {
21         strcat(chaine, " ");
22         if (i == argc) break;
23         strcat(chaine, argv[i]);
24     }
25
26     printf("La ligne de commande est %s\n", chaine);
27
28     char prog[strlen(argv[0]) + 1];
29     strcpy(prog, argv[0]);
30
31     prog[7] = 'X';
32
33     if (strcmp(prog, argv[0]))
34         printf("Les chaines \"%s\" et \"%s\" sont differentes\n", prog, argv[0]);
35
36     return 0;
37 }

```

chaines.c



## 2 Utilisation des pointeurs

```
1 #include <stdio.h>
2 #include <assert.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 int main(void)
7 {
8     char *chaine = malloc(7);
9     strcpy(chaine, "azerty");
10    printf("%s\n", chaine);
11    free(chaine);
12    // FAUX : on n'a pas le droit d'accéder à une zone libérée
13    // l'effet est imprévisible :
14    // - sur telesun, ça n'affiche rien
15    // - sur ma machine, ça affiche "azerty"
16    printf("%s\n", chaine);
17    char *c = malloc(7);
18    // FAUX : c n'a pas été initialisée, mais en pratique malloc allouera
19    // vraisemblablement la même zone que celle qui vient d'être libérée
20    // => effet imprévisible
21    printf("%s\n", c);
22
23    const int taille = 5;
24    int *tab = calloc(taille, sizeof(int));
25    int *copie = malloc(taille * sizeof(int));
26    memcpy(copie, tab, taille * sizeof(int));
27    for (int i = 0; i < taille; i++)
28        printf("%i ", copie[i]);
29    printf("\n");
30    // il est inutile de libérer la mémoire en fin de programme, ça sera
31    // fait automatiquement quand le programme se termine
32
33    while (1) {
34        int *boom = calloc(2 * 1000 * 1000 * 1000, sizeof(int));
35        assert(boom != NULL);
36    }
37
38    return 0;
39 }
```

alloc.c

```

1  #include <stdio.h>
2
3  struct ma_struct_t {
4      int i;
5      char c;
6  };
7
8  void echange(int *a, int *b)
9  {
10     int t = *a; *a = *b; *b = t;
11 }
12
13 int main(void)
14 {
15     int i = 5;
16     int *ptr = &i;
17     *ptr = 10;
18     printf("Adresse de i : 0x%08X, valeur de i : %i\n", (unsigned)ptr, i);
19
20     int j = 20;
21     echange(&i, &j);
22     printf("i = %i, j = %i\n", i, j);
23
24     char chaine[] = "azerty";
25     for (char *p = chaine; *p; p++)
26         printf("%c", *p);
27     printf("\n");
28
29     int tab[] = {1, 2, 3, 4, 5, -1};
30     for (int *p = tab; *p != -1; p++)
31         printf("Adresse : 0x%08X, valeur : %i\n", (unsigned)p, *p);
32
33     struct ma_struct_t s = {-4, 'a'};
34     struct ma_struct_t *p = &s;
35     (*p).i = 4; p->c = 'A';
36     printf("%i %c\n", p->i, s.c);
37
38     return 0;
39 }

```

pointeurs.c

```

1  #include <time.h>
2  #include <stdio.h>
3  #include <assert.h>
4  #include <stdlib.h>
5
6  struct cellule_t {
7      int val;
8      struct cellule_t *suiv;
9  };
10
11 void afficher(struct cellule_t *liste)
12 {
13     for (; liste != NULL; liste = liste->suiv) {
14         printf("%i -> ", liste->val);
15     }
16     printf("FIN\n");
17 }
18
19 void inserer_tete(struct cellule_t **liste, int v)
20 {
21     struct cellule_t *cell = malloc(sizeof(struct cellule_t));
22     assert(cell != NULL);
23     cell->val = v; cell->suiv = *liste;
24     *liste = cell;
25 }
26
27 void inserer_queue(struct cellule_t **liste, int v)
28 {
29     struct cellule_t sent = {-1, *liste};
30     struct cellule_t *queue;
31     for (queue = &sent; queue->suiv != NULL; queue = queue->suiv);
32     queue->suiv = malloc(sizeof(struct cellule_t));
33     assert(queue->suiv != NULL);
34     queue->suiv->val = v; queue->suiv->suiv = NULL;
35     *liste = sent.suiv;
36 }
37
38 void supprimer_premier(struct cellule_t **liste, int v)
39 {
40     struct cellule_t sent = {-1, *liste};
41     struct cellule_t *p;
42     for (p = &sent; (p->suiv != NULL) && (p->suiv->val != v); p = p->suiv);
43     if (p->suiv != NULL) {
44         struct cellule_t *tmp = p->suiv;
45         p->suiv = tmp->suiv;
46         free(tmp);
47     }
48     *liste = sent.suiv;
49 }

```

listes.c

```

1 int main(void)
2 {
3     struct cellule_t *liste = NULL;
4     for (int i = 6; i < 10; i++) {
5         inserer_queue(&liste , i);
6         afficher(liste);
7     }
8     for (int i = 5; i > 0; i--) {
9         inserer_tete(&liste , i);
10        afficher(liste);
11    }
12    srand(time(NULL));
13    while (liste != NULL) {
14        supprimer_premier(&liste , rand() % 10);
15        afficher(liste);
16    }
17    return 0;
18 }

```

listes.c (suite)