Exemples de code en langage C

Ensimag

mai 2012

1 Premiers pas en C

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

vide.c

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

es.c

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

fonctions.c

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

controles.c

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

structures.c

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

tab[sup - 1] = t;
10
11
12
             }
        }
13
14
15
   int main(void)
16
17
18
        // On initialise le generateur de nombres aleatoire
        srand(time(NULL));
19
20
21
        unsigned taille;
22
        printf("Entrer la taille du tableau : ");
23
        scanf("%u", &taille);
24
25
        int tab[taille];
26
        \verb|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
        return 0;
35
36
```

tableaux.c (suite)

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

chaines.c

2 Utilisation des pointeurs

```
#include <stdio.h>
    #include <assert.h>
    #include <stdlib.h>
 3
    #include <string.h>
 6
    int main(void)
 7
         char *chaine = malloc(7);
 8
         strcpy(chaine, "azerty");
9
         printf("%s\n", chaine);
10
11
         free (chaine);
         // FAUX : on n'a pas le droit d'acceder a une zone liberee
12
         // l'effet est imprevisible :
13
         // - sur\ telesun, ca\ n'affiche rien
14
         // - sur ma machine, ca affiche "azerty"
15
         printf("%s\n", chaine);
16
         \mathbf{char} *c = \mathrm{malloc}(7);
17
         /\!/ FAUX : c n'a pas ete initialisee , mais en pratique malloc allouera
18
19
         // vraisemblablement la meme zone que celle qui vient d'etre liberee
20
         // \Rightarrow effet imprevisible
21
         printf("%s\n", c);
22
23
         const int taille = 5;
24
         int *tab = calloc(taille, sizeof(int));
         \begin{array}{lll} & \textbf{int} * \texttt{copie} = & \texttt{malloc}(\texttt{taille} * \textbf{sizeof}(\textbf{int})); \\ & \texttt{memcpy}(\texttt{copie}, \texttt{tab}, \texttt{taille} * \textbf{sizeof}(\textbf{int})); \end{array}
25
26
         for (int i = 0; i < taille; i++)
27
             printf("%i ", copie[i]);
28
         printf("\n");
29
30
         // il est inutile de liberer la memoire en fin de programme, ca sera
31
              fait automatiquement quand le programme se termine
32
33
         while (1) {
              int *boom = calloc(2 * 1000 * 1000 * 1000, sizeof(int));
34
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
    void echange(int *a, int *b)
8
9
10
        int t = *a; *a = *b; *b = t;
11
12
   int main(void)
13
14
        int i = 5;
15
16
        int *ptr = \&i;
17
        *ptr = 10;
        printf("Adresse de i : 0x\%08X, valeur de i : \%i\n", (unsigned)ptr, i);
18
19
20
        int j = 20;
21
        echange(&i , &j);
22
        printf("i = \%i, j = \%i \setminus n", i, j);
23
24
        char chaine[] = "azerty";
25
        for (char *p = chaine; *p; p++)
            printf("%c", *p);
26
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;
        (*p).i = 4; p -> c = \dot{A};
35
        printf("%i %c\n", p->i, s.c);
36
37
38
        return 0;
39
```

pointeurs.c

```
#include <time.h>
2 #include <stdio.h>
3 #include <assert.h>
4
   #include <stdlib.h>
5
6
   struct cellule_t {
7
        int val;
        struct cellule_t *suiv;
8
9
   };
10
   void afficher(struct cellule_t *liste)
11
12
        for (; liste != NULL; liste = liste ->suiv) {
13
            printf("%i -> ", liste -> val);
14
15
        printf("FIN\n");
16
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 \rightarrow val = v; cell \rightarrow 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
   void supprimer_premier(struct cellule_t **liste, int v)
38
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
        *liste = sent.suiv;
48
49
   }
```

listes.c

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

listes.c (suite)