Bases du langage C

Shockwave

Structure d'un programme élémentaire

Il faut toujours inclure les bibliothèques stdlib et stdio :

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

int main(){

  /*suite d'instructions*/
  return 0; // aussi possible return EXIT_SUCCESS;
}
```

Types prédéfinis du langage C

Avec le préfixe unsigned devant le type, on ne garde que les valeurs positives ou nulles.

entiers	short (2 octets), int (4o), long (8o), long long
réels	float (4o), double (8o)
caractère	char (1o)

Variables et affectation

On peut définir une variable avec : a-z, A-Z, 0-9 et _ .
Une variable commence obligatoirement par une lettre ou _. Une variable est visible dans le bloc où elle est définie (un bloc commence par { et se termine par }). On définit les variables en haut des blocs.

```c
int i;              // definition de variables
double x, y;

int n=3;             // definition et initialisation de variables
double v=5, w=7.9;

i = n-2;             // affectation
x = (v+w)/3;
```

Opérateurs et expressions

arith. ordinaires	+, -, *, \, % (modulo)
arith. combinés avec =	+=, -=, *=, \=
incrémentation\décrémentation	++, −−
comparaison	<, <=, >, >=, == (égale), != (différent)
logique	! (non), && (et), || (ou)
autre	sizeof (donne le nombre d'octets utilisés pour stocker une variable

```
int i, j, k;
i++;  // ceci est equivalent a faire i = i+1;
i+=5; // ceci est equivalent a faire i = i+5;
i*=5  // ceci est equivalent a faire i = i*5;

j = i++;  // ATTENTION affectation du contenu de i a j PUIS incrementation de i, c'est donc
    different de j = i +1;
k = ++i;  // incrementation de i PUIS affectation du contenu de i a k
```

Schéma itératif : instruction if

```
if(/*condition*/){
  // instructions a executer si condition est vrai
}


if(/*condition*/){
  // instructions a executer si condition est vrai
}
else{
  // instructions a executer si condition est faux
}


if(/*condition1*/){
  // instructions a executer si condition1 est vrai
}
else if (/*condition2*/){
  // instructions a executer si condition2 est vrai
}
```

Schéma itératif : instruction while, do/while, for

```
while(/*condition*/){
  // instructions a executer tant que condition est vrai
}
```

```c
do{
  // instructions a executer au moins 1 fois puis autant de fois que necessaire avant que
     condition soit faux
} while(/*condition*/)

#define N 10  // On d fini la  constante N egale    10, pas de point-virgule
int i;
for(i=1; i<=N; i++){
  // instructions a executer N fois
}
```

Tableau à une dimension

```c
#define NB_ELEMENTS 5
int tab[NB_ELEMENTS]; // on a defini un tableau a une dimension qui s'appelle tab et qui
    contient 5 entiers


int tab[NB_ELEMENTS] = {3,8,-9,0,5};  // on a defini et initialise les 5 valeurs du tableau

int tab[] = {3,8,-9,0,5};  // une facon equivalente de definir et initialiser un tableau de 5
    valeurs

char message[] = "Bonjour";  //definition et initialisation du tableau nomme message et qui
    contient les caracteres B,o,n,j,o,u,r ET \0

char message[] ={'B', 'o', 'n', 'j', 'o', 'u', 'r', '\0'};  // facon equivalente

// Attention, les indices d'un tableau commencent a 0
tab[3] = 2; // on accede a la 4eme case du tableau tab et on l'affecte avec la valeur 2
```

Tableau à deux dimensions

```c
#define NB_LIGNES 3
#define NB_COLONNES 4

double matrice[NB_LIGNES][NB_COLONNES]; // on a defini un tableau 2D 3*4 de reels qui s'
    appelle matrice

double matrice[NB_LIGNES][NB_COLONNES] = {
  {3.2, 4.2, 8.4, 7.3}
  {6.3, 2.1, 5.9, 9.3}                  // on initialise en ecrivant ligne par ligne
  {8.9, 7.3, 1.5, 6.4}
};

matrice[ligne][colonne] = 10.; // on accede a la ligne ligne et la colonne colonne pour
    affecter la valeur 10
```

Types scalaires et tableau

```c
typedef unsigned int Naturel;  // definition du type naturel en utilisant le mot-cle typedef, en precisant un type predefini et en donnant un nom

typedef enum{FAUX, VRAI} Booleen;   // respecter l'ordre dans le enum, d'abord FAUX puis VRAI


#define N 10
#define M 20
typedef int Vecteur[N]; //definition d'un type tableau appele Vecteur
typedef int Matrice[N][M]  // definition d'un type tableau 2D appele Matrice
```

Types structure

```c
// un exemple
#define LONGUEUR (80 + 1)
typedef struct { int reference;
                 char nom[LONGUEUR];
                 double prix;
               } Produit;            // on defini le type Produit qui est une structure comprenant 3 champs
Produit monProduit; // monProduit est de type Produit

monProduit.prix = 9.99; // je fixe le prix de mon produit ainsi en ecrivant son nom, puis point puis le champs correspondant dans Produit
```

Fonctions et procédures

```c
// un exemple de fonction

int minimum (int x, int y) {     // on met dans l'ordre ce que renvoie la fonction, son nom et les parametres types
  if (x <y )                     // bloc d'instructions
    return x; // on n'est pas oblige de mettre des accolades quand il n'y a qu'une seule instruction
  else
    return y;
}


// un exemple de procedure

void afficherNombresParfaits (int n) {    // une procedure ne renvoie rien, donc mettre void au debut
  int i;                                  // bloc d'instructions
  for (i=1;i<=n;i++)
    if( estParfait(i) == VRAI)
      printf("%d est parfait \n", i);         // pas de return
}
```

Affichage : printf

```c
// Un exemple d'affichage de texte, d'entier et de reel

#include <stdio.h>

int n = 3, m = -5;
double x = 1.5, y = 3.8;

printf("n + m = %d \n", n+m);

printf("x - y = %f \n", x - y);

// %c pour un char, %d pour un int, %f pour un float ou double, %s pour un tableau de
    caracteres
// caracteres speciaux \n: retour a la ligne, \t: tabulation
```

Chaîne de production

D'abord écrire l'algorithme.

Puis codage et saisie :
$ emacs monProg.c &

Compilation
$ gcc -Wall monProg.c -o monProg
-Wall affiche tous les warnings, -o monProg génère l'exécutable monProg

Lancement de l'exécution :
$ ./monProg