Résumé de langage C

/*ceci met en commentaire un bloc jusqu'aux caractères*/ //ceci met en commentaire jusqu'à la fin de la ligne

TYPES DE DONNÉES

bool	booléen	1 bit	0 ou 1
char	caractère	1 octet	de –128 à +127
short	entier	2 octets	de –32768 à +32767
long **int**	entier	4 octets	de –2147483648 à +2147483647
float	réel	4 octets	de $-3,4.10^{+38}$ à $+3,4.10^{+38}$ précision 7 chiffres
double	réel	8 octets	de $-1,7.10^{+308}$ à $+1,7.10^{+308}$ précision 15 chiffres

unsigned short : non signé (0 à 65535), unsigned char (0 à 255)... **0x**50 : hexadécimal.

CARACTÈRES

\n	interligne	**\'**	cote
****	anti-slash	**\"**	guillemets

printf("message à l'écran") ;
%c indique un caractère, **%s** indique une chaîne de caractères
%d indique un entier, **%f** indique un réel

DÉCLARATION DES VARIABLES

int origine, i, j, alpha ; float x, y, z ; //variables déclarées **printf("i vaut %d et x vaut %f",i,x) ;**

i =3/(int) x; //convertit le réel « x » en entier lors de l'opération, = est l'affectation

TABLEAUX

coord[0]=1 ;//1er élément

int coord[3]={4,2,13} ; // tableau de 3 **int** initialisé **printf ("adresse du tableau %d, 3ème élément %d", coord, coord[2]) ;**

float matrice[5][3]; // 5x3 éléments réels 5 lignes et 3 colonnes

char nom[20]="albator"; // Déclare et initialise une chaîne de 19 caractères (plus le marqueur de fin de chaîne : \0)

POINTEURS ET ADRESSES

int *iptr; // pointeur sur un entier « iptr »

iptr = (int*)malloc(10*sizeof(int)); // réserve la place mémoire pour 10 entiers soit 40 octets
printf ("adresse du pointeur %d, 1er élément %d", iptr, *iptr) ; printf ("3ème élément %d",*(iptr+2)) ;
free(iptr); // libère la place précédemment réservée

adresse=&entier ; //« &entier » est l'adresse de la variable « entier »

float a;printf("A="); scanf("%f",&a);char Nom[10];printf("Nom="); scanf("%s",Nom);//scanf demande 1 adresse en argument

OPÉRATIONS ÉLÉMENTAIRES

Arithmétiques

multiplication :	**a*b**	addition soustraction :	**a+b a-b**
division :	**a/b**	incrémentation :	**a++**
modulo :	**a%b**	décrémentation :	**a--**

opérateurs relationnels : le résultat est 1 ou 0 (true ou false)

supérieur à :	**b>a**	< ou égal à :	**c<=d**
> ou égal à :	**b>=a**	égal à :	**a==b**
inférieur à :	**c<d**	différent de :	**a!=d**

logiques portant sur des mots binaires (opérations bit à bit)

décalage à gauche :	**a<<n**	//décale a de n bits à gauche	
décalage à droite :	**a>>n**	OU exclusif :	**a^b**
ET logique :	**a&b**	NON logique :	**~a**
OU logique :	**a|b**		

logiques portant sur des booléens : combinaison d'opérateurs relationnels

ET :	**a&&b**	//vaut 1 si a et b sont vrais
OU :	**a||b**	//vaut 1 si a ou b est vrai
NON :	**!a**	//vaut 1 si a est faux

a=c==d ; //affecte à « a » la valeur 1 si c est égal à d et la valeur 0 dans le cas contraire.
condition=(x > a)&&(x < b) ; //vaut 1 si x est compris entre a et b.
a +=b ; //a=a+b valable pour la plupart des opérateurs

INSTRUCTIONS CONDITIONNELLES

if (expression) {instruction1 ; instruction2 ;}
else {instruction1 ; instruction1 ;}
//**else** est facultatif

switch (expression){
case valeur1 : instruction1 ; instruction2 ; **break ;**
case valeur2 : instruction1 ; instruction2 ; **break ;**
case valeur3 : instruction1 ; instruction2 ; **break ;**
default : instruction1 ; instruction2 ; /*facultatif*/ }

if((a==2)||(b==2)) printf("l'un vaut 2") ; else printf("aucun ne vaut 2") ; // les {} sont facultatives en cas d'instruction unique

INSTRUCTIONS ITERATIVES

while (expression) {instruction1 ; instruction2 ;}

for (expression1; expression2; expression3)
{instruction1 ; instruction2 ;}

do {instruction1 ; instruction2 ;}
while (expression);

int j=0; while(j<89) { printf("%d",j); j++;} /* ou */ **for(j=0 ;j<89 ;j++) printf("%d",j);** //89 itérations

FONCTIONS

float sin(int); //déclaration d'une fonction qui retourne un **float**, et prend un **int** en argument

x = a*b*sin(alpha) ;/*la fonction **sin** retourne un nombre utilisable dans une expression, elle prend en argument la valeur de **alpha***/

```
void main()
{/*fonction
particulière :
programme principal*/
int i ,a=5;
i=3+add(a,4) ; //i=12
}
```

```
void affiche(char c)
{printf("%c",c) ;}
//aucun retour
void bonjour()
{printf("bonjour") ;}
//aucun argument
//aucun retour
```

```
int add (int a, int b)
//définit 1 fonction
{     int c ;
      c = a + b ;
      return c ;
//valeur retournée
}
```

```
void passvaleur(int d) {d++ ;} //prend 1 valeur
void passadresse(int *d) {(*d)++ ;}//prend 1 adresse
void passreference(int &d) {d++ ;} /*prend 1 variable
en argument et se réfère à sont adresse : référence*/
void main()
{ int a=2,b=2,c=2 ;
  passvaleur(a) ; //a=2 l'adresse n'étant pas connue
  passadresse(&b) ; //b=3 passage de l'adresse de b
  passreference(c) ; /*c=3 l'adresse est connue*/}
```

PREPROCESSEUR

#define pi 3.14159 // 3.14159 remplacera les occurrences de « pi ».

#include<math.h> /*Le préprocesseur réécrira le fichier math.h dans le fichier source avant la compilation.*/

d_iervese@yahoo.com