Le Langage C

Mots clé :
Cahier des charges, algorithme, algorigramme, langage évolué

Centre d'intérêt : CI4 : Gestion de l'information / Structures matérielles et logicielles associées au traitement de l'information

Objectif du cours :
Comprendre un langage évolué

SOMMAIRE

1. INTRODUCTION

Le langage C est un langage évolué et structuré, assez proche du langage machine destiné à des applications de contrôle de processus (gestion d'entrées/sorties, applications temps réel ...). Les compilateurs C possèdent les taux d'expansion les plus faibles de tous les langages évolués (rapport entre la quantité de codes machine générée par le compilateur et la quantité de codes machine générée par l'assembleur et ce pour une même application).

Le langage C possède assez peu d'instructions, il fait par contre appel à des bibliothèques, fournies en plus ou moins grand nombre avec le compilateur.

exemples: math.h : bibliothèque de fonctions mathématiques
stdio.h : bibliothèque d'entrées / sorties standard
conio.h…

On ne saurait développer un programme en C sans se munir de la documentation concernant ces bibliothèques (voir aide …).

Les compilateurs C sont remplacés petit à petit par des compilateurs C++.
Un programme écrit en C est en principe compris par un compilateur C++.
Le cours qui suit est un cours de langage C adapté au compilateur (gcc) du logiciel DEVC++

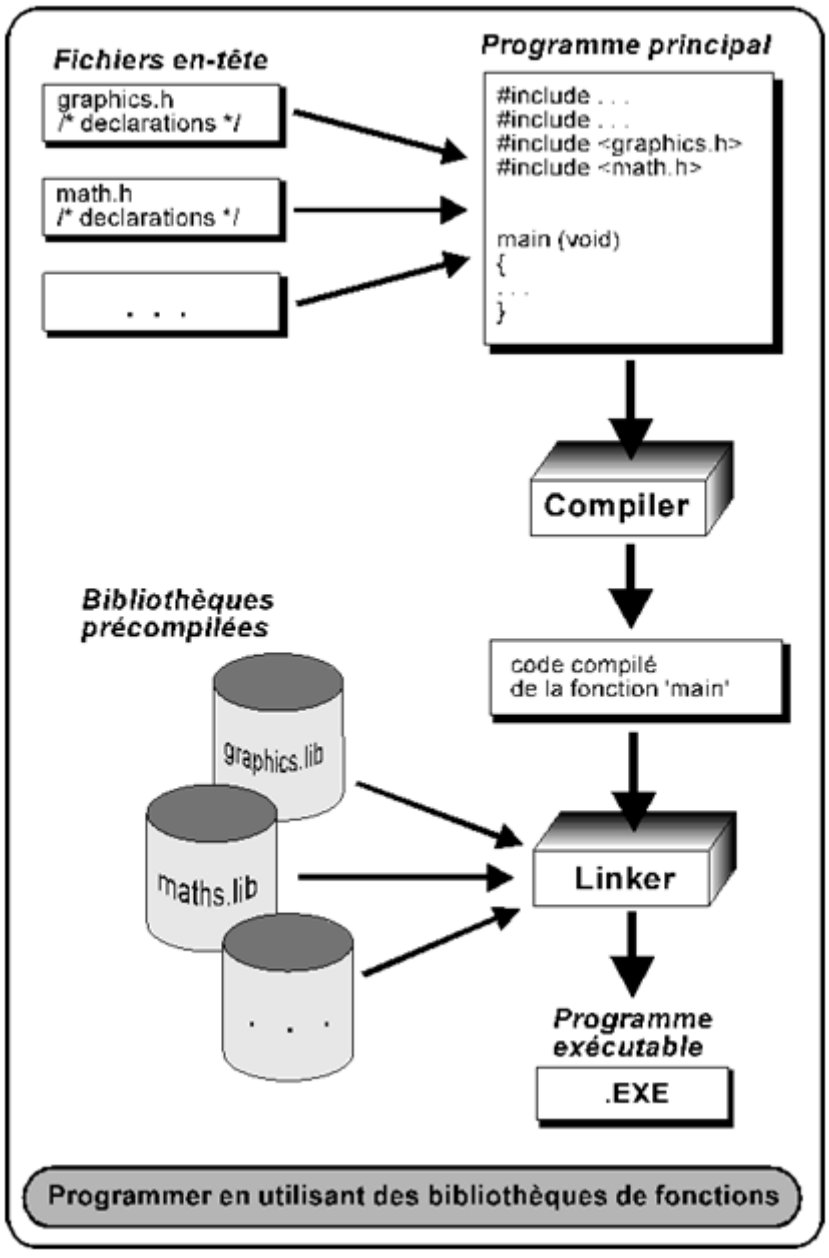
ETAPES PERMETTANT L'EDITION, LA MISE AU POINT, L'EXECUTION D'UN PROGRAMME

1- ***Création d'un projet dans DEVC++*** (en langage C et pas C++ !, en mode console application) avec un nom explicite.
Edition du programme source : à l'aide de l'éditeur de texte. Le nom du fichier contient l'extension .C, exemple: main.c ou cube.c

2- ***Compilation du programme source*** **:** c'est à dire création des codes machine destinés au microprocesseur utilisé. Le compilateur indique les erreurs de syntaxe mais ignore à ce moment là les fonctions-bibliothèque appelées par le programme.
Le compilateur génère un fichier binaire, non listable, appelé fichier objet: cube.o

3- ***Editions de liens***: Le code machine des fonctions-bibliothèque est chargé, création d'un fichier binaire, non listable, appelé fichier executable: cube.exe

4- ***Exécution du programme*** *:*Les compilateurs permettent en général de construire des programmes composés de plusieurs fichiers sources, d'ajouter à un programme des unités déjà compilées ...



Etudions l'exemple simple suivant :

Exercice 1: Analyser l'algorithme ci-dessous et tester le programme:

```
Algorithme afficher ; co nom ou en-tête de l'algorithme fco ;
Début
        Afficher (« BONJOUR ») ;co le mot BONJOUR doit apparaitre sur l'écran du PC fco ;
Fin.
```

Tracuction en langage C :

```
#include <stdio.h>                  /* bibliothèque d'entrées-sorties standard */
#include <conio.h>

int main(void)                      /*  peut remplacer     int main(int argc, char *argv[])  et   return 0; à la fin  */
{
   puts("BONJOUR");                 /* utilisation d'une fonction-bibliotheque dans laquelle la fonction puts est définie */
   system("pause") ;                /* pour faire une pause dans la fenêtre d'affichage de la console */
}
```

Le langage C distingue les minuscules, des majuscules. Les mots réservés du langage C doivent être écrits **en minuscules. Prenez l'habitude d'écrire les identificateurs en minuscule.**

On a introduit dans ce programme la notion d'interface homme/machine (IHM).
- L'utilisateur visualise une information sur l'écran,
- La fonction **main** est la fonction principale des programmes en C: Elle se trouve obligatoirement dans tous les programmes. L'exécution d'un programme commence automatiquement par l'appel de la fonction **main**().

Exercice 2: Analyser l'algorithme ci-dessous, compléter le type et tester le programme:

```
Algorithme somme ; co en-tête fco ;
Var ent a,b,somme ; co déclaration des variables fco ;
Début
        Afficher (« BONJOUR ») ;
        Dans a mettre 10 ;
        Dans b mettre 50 ;
        Dans somme mettre a+b ;
        Afficher (« voici le résultat : » somme) ;
Fin.
```

Tracuction en langage C :

```
#include <stdio.h>                      /* bibliotheque d'entrees-sorties standard */
#include <conio.h>
int main(void)
{
  int a, b, somme ;                     /* déclaration de 3 variables de type …….. */
  puts("BONJOUR");                      /* utilisation d'une fonction-bibliotheque */
  a = 10 ;                              /* affectation*/
  b = 50 ;                              /* affectation */
  somme = a + b ;                       /* affectation et opérateurs   */
  printf(" Voici le resultat : %d\n", somme) ;
  system("pause") ;
}
```

Dans ce programme, on introduit 3 nouveaux concepts :
- La notion de déclaration de variables : les **variables** sont les données que manipulera le programme lors de son exécution. Ces variables sont stockées dans la **mémoire vive de l'ordinateur**. Elles doivent être déclarées au début du programme.
- La notion d'affectation, symbolisée par le signe =
- La notion d'opération +
- l'utilisation de la fonction d'affichage printf

2. Les différents types de variables

21- Les entiers

Le langage C distingue plusieurs types d'entiers:

TYPE	DESCRIPTION	TAILLE MEMOIRE
int	entier standard signé	4 octets: $-2^{31} \leq n \leq 2^{31}-1$
unsigned int	entier positif	4 octets: $0 \leq n \leq 2^{32}$
short	entier court signé	2 octets: $-2^{15} \leq n \leq 2^{15}-1$
unsigned short	entier court non signé	2 octets: $0 \leq\leq n \leq 2^{16}$
char	caractère signé	1 octet : $-2^{7} \leq\leq n \leq 2^{7}-1$
unsigned char	caractère non signé	1 octet : $0 \leq n \leq 2^{8}$

Numération: En décimal les nombres s'écrivent tels que. Ils sont précédés de **0x** en hexadécimal.
exemple: 127 en décimal s'écrit 0x7f en hexadécimal.

Remarque: En langage C, le type **char** est un cas particulier du type entier:
un caractère est un entier de 8 bits (voir tableau des codes ASCII)

Exemples:
Les caractères alphanumériques s'écrivent entre ‘ ‘ (quotes)
Le **caractère** 'B' a pour valeur 0x42 en hexadécimal (son code ASCII).

Quelques caractères particuliers qui servent au formatage de l'affichage (saut à la ligne, tabulation …):

CARACTERE		VALEUR (code ASCII)	NOM ASCII
'\n'	**interligne**	0x0a	LF
'\t'	tabulation horizontale	0x09	HT
'\v'	tabulation verticale	0x0b	VT
'\r'	retour charriot	0x0d	CR
'\f'	saut de page	0x0c	FF
'\\'	backslash	0x5c	\
'\''	quote	0x2c	'
'\"'	guillemets	0x22	"

22- Les réels

Un réel est composé - d'un signe - d'une mantisse - d'un exposant
Un nombre de bits est réservé en mémoire pour chaque élément.

Le langage C distingue 2 types de réels:

TYPE	DESCRIPTION	TAILLE MEMOIRE	
float	réel standard	4 octets	1.E-36 to 1.E+36.
double	réel double précision	8 octets	1.E-303 to 1.E+303.

23- Les déclarations de constantes

Le langage C autorise 2 méthodes pour définir des constantes :

1ere méthode: **déclaration** d'une variable, dont la valeur sera constante pour tout le programme:

Exemple:
```
int void main()
{
        const float PI = 3.14159;
        float perimetre,rayon = 8.7;
        perimetre =  2*rayon*PI;
}
```

Dans ce cas, le compilateur réserve de la place en mémoire (ici 4 octets), pour la variable pi, mais dont on ne peut changer la valeur (programme en mémoire morte).

2eme méthode: **définition d'un symbole** à l'aide de la directive de compilation **#define.**

Exemple:
```
#define PI = 3.14159;
int void main()
{
        float perimetre,rayon = 8.7;
        perimetre =  2*rayon*PI;
}
```

Le compilateur ne réserve pas de place en mémoire. Les constantes déclarées par #define s'écrivent traditionnellement en majuscules, mais ce n'est pas une obligation (utilise la mémoire vive).

24- Opérateurs combinés

Le langage C autorise les opérations arithmétiques, logiques et de comparaison. Lorsqu'une même variable est utilisée de chaque côté du signe = d'une affectation on peut utiliser les écritures simplifiées. Ces écritures sont à éviter lorsque l'on débute l'étude du langage C car elles nuisent à la lisibilité du programme.

a = a+b;	est équivalent à	a+= b;
a = a-b;	est équivalent à	a-= b;
a = a & b;	est équivalent à	a&= b;

3. Les fonctions

En langage C les sous-programmes s'appellent des **fonctions.**

L'imbrication de fonctions n'est pas autorisée en C: une fonction ne peut pas être déclarée à l'intérieur d'une autre fonction. Par contre, une fonction peut appeler une autre fonction. Cette dernière doit être déclarée **avant** celle qui l'appelle.

Une fonction possède un et un seul point d'entrée, mais éventuellement plusieurs points de sortie (à l'aide du mot **return**).

Une variable connue uniquement d'une fonction ou de main() est une **variable locale.**
Une variable connue de tout le programme est une **variable globale.**

FONCTIONS SANS PASSAGE D'ARGUMENTS ET NE RENVOYANT RIEN AU PROGRAMME :

Une fonction ne renvoyant rien au programme est une fonction de type **void.**

Exemple:

```
#include <stdio.h>
#include <conio.h>

void bonjour()                  /* declaration de la fonction */
        {
                printf("bonjour");
        }

int  main(void)                 /* programme principal */
        {
                bonjour();      /* appel de la fonction */
                system("pause ");
        }
```

FONCTION AVEC PASSAGE DE PARAMETRE :

Exemple: **log**

fonction Fonction logarithme népérien.

prototype double log(double);

prototype dans math.h et donc bibliothèque à charger au début du programme

La fonction log, renvoie au programme un réel. On traite la fonction comme une variable de type double. Il faut lui passer un paramètre de type double, elle calcule le log et retourne le résultat sous forme d'un réel de type double

ex:

```
double x,y;
x= 2,3421;
y = log(x);
printf("log(x) = %f\n",y);
```

4. Saisie au clavier et affichage à l'écran

LA FONCTION GETCH

La fonction getch, appartenant à la bibliothèque conio.h permet la saisie au clavier d'un caractère alphanumérique, **sans écho écran**. La saisie s'arrête dès que le caractère a été frappé (getche pour avoir un écho à l'écran).

La fonction getch n'est pas définie dans la norme ANSI mais elle peut exister dans la bibliothèque d'autres compilateurs.

On peut utiliser getch de deux façons:

- sans retour de variable au programme:

Exemple:

```
printf("POUR CONTINUER FRAPPER UNE TOUCHE ");
getch();
```

- avec retour de variable au programme:

Exemple:

```
char alpha;
printf("ENTRER UN CARACTERE (ATTENTION PAS DE RETURN) ");
alpha = getch();
```

Les parenthèses vides de getch() signifient qu'aucun paramètre n'est passé à cette fonction par le programme.

LA FONCTION SCANF

La fonction scanf, appartenant à la bibliothèque stdio.h, permet la saisie clavier de n'importe quel type de variable.

Les variables à saisir sont formatées, le nom de la variable est précédé du symbole & désignant l'adresse de la variable (On reverra ce symbole dans le chapitre sur les pointeurs). La saisie s'arrête avec "RETURN" (c'est à dire LF), les éléments saisis s'affichent à l'écran (**saisie avec écho écran**).

Tous les éléments saisis après un **caractère d'espacement** (espace, tabulation) sont ignorés. Ne pas taper d'espace.

Exemples:
```
char alpha;
int i;
float r;
scanf("%c",&alpha);          /* saisie d'un caractère */
scanf("%d",&i);              /* saisie d'un nombre entier en décimal */
scanf("%x",&i);              /* saisie d'un nombre entier en hexadécimal*/
scanf("%f",&r);              /* saisie d'un nombre réel */
```

Remarque: Si l'utilisateur ne respecte pas le format indiqué dans scanf, la saisie est ignorée. Aucune erreur n'est générée.

Exemple:
```
char alpha;
scanf("%d",&alpha);
```
Si l'utilisateur saisit « 97 » tout va bien, alpha devient le caractère dont le code ASCII vaut 97.
Si l'utilisateur saisit « a », sa saisie est ignorée.

LA FONCTION GETCHAR

La fonction getchar pemet la saisie d'un caractère (char). Elle appartient à la bibliothèque stdio.h. Les 2 écritures suivantes sont équivalentes:

```
char c;
printf("ENTRER UN CARACTERE: ");
scanf("%c",&c);
```

```
char c;
printf("ENTRER UN CARACTERE: ");
c = getchar();
```

Non formatée, la fonction getchar est moins gourmande en place mémoire que scanf. Il vaut mieux l'utiliser quand cela est possible; getchar utilise le flux d'entrée exactement comme scanf.

AFFICHAGE DE NOMBRES OU DE TEXTE A L'ECRAN

LA FONCTION PRINTF

Ce n'est pas une instruction du langage C, mais une fonction de la bibliothèque stdio.h.

Exemple: affichage d'un texte:
```
printf("BONJOUR"); /* pas de retour à la ligne du curseur après l'affichage, */
printf("BONJOUR\n");        /* affichage du texte, puis retour à la ligne du curseur.  */
```

La fonction printf permet l'utilisation de formats de sortie, avec la structure suivante:
printf("texte %format",nom_de_variable);
printf("texte %format1…% format2 ....% formatn",variable1,variable2, ...,variablen);

Formats de sortie:
%d affichage en décimal (entiers de type int),
%x affichage en hexadécimal (entiers de type int),
%f affichage d'un réel %.2f pour afficher avec 2 décimales,
%u affichage en décimal non signé (entiers de type unsigned int),
D'autres formats existent (consulter une documentation constructeur) notamment :
%c affichage d'un caractère…

Exercice 3: Analyser l'algorithme ci-dessous, le traduire en langage C et tester le programme:

```
Algorithme ASCII ; co en-tête fco ;
Var car a ; co déclaration des variables fco ;
Début
        Dans a mettre 'Z' ; co mettez la lettre de votre choix fco ;
        Afficher ("Le code ASCII du caractère "___" est: "___"en hexa et"___"en decimal") ; co affichage à l'écran fco ;
Fin.
```

VARIABLES ET FONCTIONS

On a vu qu'une variable **globale** est déclarée au début du programme et qu'elle est connue de tout le programme. **Les variables globales sont initialisées à 0 au début de l'exécution du programme, sauf si on les initialise à une autre valeur.**

On a vu aussi qu'une variable **locale** (déclarée au début d'une fonction ou de main()) n'est connue que de cette fonction ou de main().
Les variables locales ne sont pas initialisées (sauf si on le fait dans le programme) et elles perdent leur valeur à chaque appel à la fonction.

Exemple :

```
#include <stdio.h>
#include <conio.h>

int n;                  /* variable globale, connue de tout le programme */

void carre()        /* déclaration de la fonction qui calcule le carré d'un nombre saisi */
{
  int n2;   /* variable locale */
  printf("ENTRER UN NOMBRE: ");
  scanf("%d",&n);
  n2 = n*n;
  printf("VOICI SON CARRE: %d\n",n2);
}

void cube()        /* declaration de la fonction qui calcule le cube d'un nombre saisi */
{
  int n3;  /* variable locale */
  printf("ENTRER UN NOMBRE: ");
  scanf("%d",&n);
  n3 = n*n*n;
  printf("VOICI SON CUBE: %d\n",n3);
}

int main(void)              /* programme principal */
{
  char choix;                   /* variable locale a main() */
  printf("CALCUL DU CARRE TAPER 2\n");
  printf("CALCUL DU CUBE TAPER 3\n");
  printf("\nVOTRE CHOIX: ");
  scanf("%c",&choix);
  switch(choix)
          {
          case '2':carre();break;
          case '3':cube();break;
          }
  system("pause");
}
```

Un programme bien construit possède peu de variables globales, on aurait pu ici déclarer 2 variables locales n indépendantes l'une de l'autre dans chacune des fonctions carre() et cube().

Exercice 4: L'algorithme ci-dessous correspond à une amélioration de l'exercice2. L'analyser, le traduire en langage C et tester le programme.

```
Algorithme somme-exr4
Var entier a, b, somme ; co déclaration variables globales fco ;

Co programme principal fco ;
Début
        Ecrire ("Bonjour") ;
        Ecrire ("Veuillez saisir un premier nombre et appuyer sur entrée") ;
        a := lire (clavier) ;
        Ecrire ("Veuillez saisir un deuxième nombre et appuyer sur entrée") ;
        b := lire (clavier) ;
        Somme := a + b ;
        Ecrire ("la somme de (a) et (b) vaut (somme)") ;
Fin.
```

Exercice 5: L'algorithme ci-dessous correspond à une amélioration de l'exercice3. L'analyser, le traduire en langage C et le tester.

```
Algorithme ASCII-exr5
Var car touche; co déclaration des variables fco ;

Co programme principal fco ;
Début
        Ecrire ("Bonjour, veuillez appuyer sur une touche");
        Touche := lire (clavier) ; co mémorisation de la touche enfoncée fco ;
        Ecrire ("Le code ASCII de (touche) est (touche) en décimal et (touche) en hexadécimal");
        co par exemple : le code ASCII de A est 65 en décimal et $41 en héxadécimal fco ;
Fin.
```

Exercice 6: L'algorithme ci-dessous permet de convertir un nombre décimal en hexadécimal. Analyser l'algorithme, le traduire en langage C et tester le programme.

```
Algorithme entier-hexadecimal
Var entier nombre; co déclaration des variables fco ;

Co programme principal fco ;
Début
        Ecrire ("Entrer un nombre entier en décimal");
        Lire (nombre) ; co mémorisation dans la variable « nombre » de la touche enfoncée fco ;
        Ecrire (" le nombre décimal (nombre) vaut (nombre) en HEXADECIMAL ") ;
Fin.
```

NOTION DE FLUX D'ENTREE

Lorsque l'on saisit au clavier une suite de caractères terminés par "RETURN" ces caractères sont rangés dans un tampon (ou buffer) de type FIFO (First In/First Out), le dernier caractère rangé dans le tampon est LF (code ASCII 0x0A).
Cette suite de caractères est appelée **flux d'entrée.**
La taille du tampon dépend de la machine et du compilateur utilisé
Une compilation du programme vide le tampon.

5. Structures de choix

L'INSTRUCTION SI ... ALORS ... SINON ...

Il s'agit de l'instruction: **si** (expression conditionnelle vraie)
alors {BLOC 1 D'INSTRUCTIONS}
sinon {BLOC 2 D'INSTRUCTIONS}

Syntaxe en C:

```
if (expression)
        {
          .....;          /* bloc 1 d'instructions */
          .....;
        }
        else
        {
          .....;          /* bloc 2 d'instructions */
          .....;
        }
```

Le bloc "sinon" est optionnel: **si** (expression vraie)
alors {BLOC D'INSTRUCTIONS}

Syntaxe en C:
```
if (expression)
{
  .....;          /* bloc d'instructions */
  .....;
}
```

Remarque: les {} ne sont pas nécessaires lorsque les blocs ne comportent qu'une seule instruction.

Les opérateurs logiques dans les structures de choix

test d'égalité: **if (a==b)** **" si a égal b "** **ATTENTION ==**

test de non égalité: **if (a!=b)** **" si a différent de b "**

tests de relation d'ordre: **if (a<b)** **if (a<=b)** **if (a>b)** **if (a>=b)**

test de ET LOGIQUE: **if ((expression1) && (expression2))**
" si l'expression1 ET l'expression2 sont vraies "

test de OU LOGIQUE **if ((expression1) || (expression2))**
" si l'expression1 OU l'expression2 est vraie "

test de NON LOGIQUE **if (!(expression1))**
" si l'expression1 est fausse "

Exercice 7: Analyser l'algorithme ci-dessous, le traduire en langage C et tester le programme.

```
Algorithme racine
Var réel nombre, result; co déclaration des variables fco ;

Co programme principal fco ;
Début
        Ecrire ("Entrer un nombre entier") ;
        Lire (nombre);
        Result := racine carrée de (nombre) ; co en C racine s'ecrit « sqrt(x) » fco ;
        Ecrire ("La racine carrée de (nombre) est (result) ");
Fin.
```

Exercice 8: Analyser l'algorithme ci-dessous, le traduire en langage C et tester le programme.

```
Algorithme racine_si
Var réel nombre, result;

Co programme principal fco ;
Début
        Ecrire ("Entrer un nombre entier");
        Lire nombre ;
        Si nombre > 0
                Alors   Result := racine carrée de (nombre) ;
                                Ecrire ("La racine carrée de (nombre) est (result) ");
                Sinon   Ecrire ("impossible le nombre est négatif");
        Fsi ;
Fin.
```

Le langage C admet des écritures contractées dans les expressions de test:

char reponse;
printf("Voulez-vous jouer ?");
reponse = getchar();
if(reponse == 'o')
printf("BONJOUR\n");
else printf("TANT-PIS"\n);
	est équivalent à	char reponse;
		printf("Voulez-vous jouer ?");
		
		if((reponse = getchar()) =='o')
		printf("BONJOUR\n");
		else printf("TANT-PIS"\n);

L'INSTRUCTION CAS OU ... FAIRE ...

L'instruction switch permet des choix multiples **uniquement sur des entiers (int) ou des caractères (char).**

Syntaxe:

```
switch(variable)
{
        case 'valeur1':{action1;break;}
        case 'valeur2':{action2;break;}
        …
        case 'valeurN':{actionN;break;}
        default:{actionM ;}                    //pas de "break" ici
}                                              //le bloc "default" n'est pas obligatoire.
```

L'instruction switch correspond à une cascade d'instructions if ...else

Exercice 9: Cet algorithme permet de faire des calculs simples (+ - * /) entre deux valeurs en invitant l'utilisateur à entrer son calcul sous la forme : val1 opérateur val2 (exemple 2-6 ou 58.2*5). Le message « ce n'est pas une opération connue » s'affichera s'il se trompe.
Analyser l'algorithme ci-dessous, le traduire en langage C et tester le programme.

```
Algorithme opération
Var       réel val1, val2, result;
          Car operation ;

Co programme principal fco ;
Début
          Ecrire ("Entrer une opération sous la forme val1 operation val2") ;
          Lire (val1, opération, val2)  ;
          Selon (opération)
                    (*) :      result := val1*val2 ;
                                        Ecrire ("le produit de (val1) et (val2) est (result) ") ; fselon ;
                    (/) :      result : = val1/val2 ;
                                        Ecrire ("le quotient de (val1) et (val2) est (result) ") ; fselon ;
                    (+) :      result := val1+val2 ;
                                        Ecrire ("la somme de (val1) et (val2) est (result) ") ; fselon ;
                    (-) :      result : = val1-val2 ;
                                        Ecrire ("la différence de (val1) et (val2) est (result) ") ; fselon ;
                    Autrement : Ecrire (" ce n'est pas une opération connue ") ;
          Fselon ;
Fin.
```

6. Structures itératives

LA BOUCLE FAIRE …TANT QUE ...

Il s'agit de l'instruction: **faire**{BLOC D'INSTRUCTIONS}
tant que (expression vraie)

Syntaxe en C:

```
do
 {
.....;                /* bloc d'instructions */
.....;
}
while (expression) ;
```

Le test se fait **à la fin,** le bloc d'instructions est forcément exécuté au moins une fois.

Exercice 10: Cet algorithme reprend l'exercice 9 en y ajoutant une condition tant que l'utilisateur ne se trompe pas d'opérateur.
Analyser l'algorithme ci-dessous, le traduire en langage C et tester le programme.

```
Algorithme opération
Co déclaration des constantes ou variables fco ;
Var       réel val1, val2, result;
          Car operation ;
          entier erreur = 0; co initialisation de la variable fco ;


Co programme principal fco ;
Début
Faire
          Ecrire ("Entrer une opération sous la forme val1 operation val2") ;
          Lire val1, opération, val2  ;
          Selon (opération)
          (*) :      result := val1*val2 ;
                     Ecrire ("le produit de (val1) et (val2) est (result) ") ; fselon ;
          (/) :      result := val1/val2 ;
                     Ecrire ("le quotient de (val1) et (val2) est (result) ") ; fselon ;
          (+) :      result := val1+val2 ;
                     Ecrire ("la somme de (val1) et (val2) est (result) ") ; fselon ;
          (-) :      result := val1-val2 ;
                     Ecrire ("la différence de (val1) et (val2) est (result) ") ; fselon ;
          Autrement :      Ecrire (" ce n'est pas une opération connue ") ;
                           Erreur :=1 ;
          Fselon ;
Tant que (erreur = 0) ;
Fin.
```

LA BOUCLE TANT QUE ... FAIRE ...

Il s'agit de l'instruction: **tant que** (expression vraie)
Faire {BLOC D'INSTRUCTIONS}

Syntaxe en C:

```
while (expression)
          {
          .....;              /* bloc d'instructions */
          .....;
          }
```

Le test se fait **d'abord,** le bloc d'instructions n'est pas forcément exécuté.

Remarque: les {} ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.

On peut rencontrer la construction suivante: **while (expression);** terminée par un ; et sans la présence du bloc d'instructions. Cette construction signifie: **"tant que l'expression est vraie attendre".**

L'INSTRUCTION POUR ...

Il s'agit de l'instruction:
pour (initialisation; condition de continuité vraie; modification)
{BLOC D'INSTRUCTIONS}

Syntaxe en C:

```
for(initialisation ; condition de continuité ; modification)
                    {
                     .....;          /* bloc d'instructions */
                     .....;
                    }
```

Remarques:
Les {} ne sont pas nécessaires lorsque le bloc ne comporte qu'une seule instruction.
Les 3 instructions du for ne portent pas forcément sur la même variable.
Une instruction peut être omise, mais pas les ;

Exemples:

```
for(i = 0 ; i<10 ; i++)
{
 .....;          /* bloc d'instructions */
 .....;
}
```

Exercice 11: Analyser l'algorithme ci-dessous, le traduire en langage C et tester le programme.

```
Algorithme étoile
Var entier i;

Co programme principal fco ;
Début
        Pour i variant de 1 à 30 par pas de 1 faire
                Afficher ("*") ;
                Temporiser 500ms ;
        Fpour ;
fin
```

Remarques:
Utilisation de variables différentes:

```
resultat = 0;
for(i = 0 ; resultat<30 ; i++)
        {
         .....;          /* bloc d'instructions */
         .....;
         resultat = resultat + 2*i;
        }
```

Exercice 12 : Saisir un réel x, saisir un entier n, calculer x puissance n, où n correspond à la puissance désirée et afficher le résultat. Compléter l'algorithme ci-dessous, le traduire en langage C et tester le programme.

```
Algorithme puissance
Var                    co partie déclarative fco ;


Co programme principal fco ;
Début




fin
```

7. Utilisation d'une bibliothèque

Ce petit chapitre vise à expliquer comment se servir d'une bibliothèque de fonctions.

Les fichiers de type ".h" (conio.h, dos.h stdio.h etc...), appelés ***fichiers d'en tête*** contiennent la définition des ***prototypes*** des fonctions utilisées dans le programme. Le prototype précise la syntaxe de la fonction: son nom, les paramètres éventuels à passer, la valeur éventuelle retournée au programme.
Grâce aux lignes "#include", le compilateur lit les fichiers de type ".h" et vérifie que la syntaxe de l'appel à la fonction est correcte.

8. Les pointeurs

L'étude des pointeurs montre l'adaptation du langage C à la conduite de processus. On verra dans ce chapitre et les suivants la puissance de cette notion par ailleurs de concept simple pour un informaticien industriel.
Définition: Un pointeur est une adresse mémoire. On dit que le pointeur pointe sur cette adresse.

L'OPERATEUR ADRESSE &

L'opérateur adresse & retourne l'adresse d'une variable en mémoire.

Exemple:
```
int i = 8;
printf("VOICI i: %d\n",i);
printf("VOICI SON ADRESSE EN HEXADECIMAL: %p\n",&i);
```

On remarque que le format d'une adresse est **%p** (hexadécimal) ou **%d** (décimal) dans printf.

DECLARATION DES POINTEURS

Une variable de type pointeur se déclare à l'aide de l'objet pointé précédé du symbole *** (opérateur d'indirection).**

Exemple:
```
char *pc;    pc est un pointeur pointant sur un objet de type char
int *pi;     pi est un pointeur pointant sur un objet de type int
float *pr;   pr est un pointeur pointant sur un objet de type float
```

L'opérateur * désigne en fait le contenu de l'adresse.

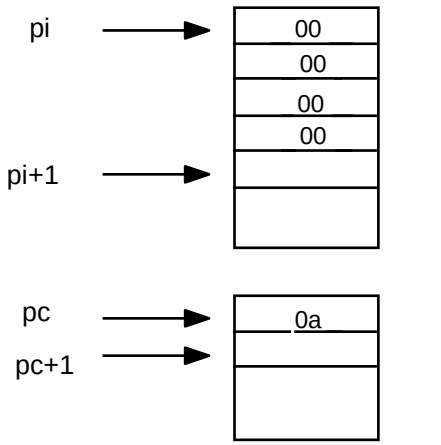
Exemple:
```
char *pc;
*pc = 34 ;
printf("CONTENU DE LA CASE MEMOIRE: %c\n",*pc);
printf("VALEUR DE L'ADRESSE EN HEXADECIMAL: %p\n",pc);
```

ARITHMETIQUE DES POINTEURS

On peut essentiellement **déplacer** un pointeur dans un plan mémoire à l'aide des opérateurs d'addition, de soustraction, d'incrémentation, de décrémentation. On ne peut le déplacer que **d'un nombre de cases mémoire multiple du nombre de cases réservées en mémoire pour la variable sur laquelle il pointe.**

C STANDARD

```
int *pi;
char *pc;
*pi = 0;
*pc = 0x0a;
```



PLAN MEMOIRE

Exemples:

```
int *pi;          /*  pi pointe sur un objet de type entier */
float *pr;        /*  pr pointe sur un objet de type réel */
char *pc;         /*  pc pointe sur un objet de type caractère */
*pi = 421;        /*  421 est le contenu de la case mémoire pi et des 3 suivantes */
*(pi+1) = 53;     /* on range 53  4 cases mémoire plus loin   */
*(pi+2) = 0xabcd; /* on range 0xabcd  8 cases mémoire plus loin */
*pr = 45.7;       /* 45,7 est rangé dans la case mémoire r et les 3 suivantes  */
```