ALGORITHMIQUE ET PROGRAMMATION

Mr Lawakilia Wilfried DRABO
ingénieur informaticien

&

Mr Téeg-wendé ZOUGMORE
enseignant à l'UPB/ESI

OBJECTIFS DU COURS

- Ecrire un programme en langage C
- Lire un programme écrit en langage C

DÉFINITION D'UN LANGAGE DE PROGRAMMATION

- Un langage de programmation est un langage informatique permettant à un être humain d'écrire un programme (informatique) destiné à être exécuté sur une machine qui est généralement un ordinateur.
- Les programmes demandent à l'ordinateur d'effectuer des actions. Ils sont partout et permettent de faire tout et n'importe quoi sur un ordinateur.

  Exemple : la calculatrice, les jeux vidéos, etc.

PROBLÉMATIQUE

- L'ordinateur ne comprend que le langage binaire (des 0 et des 1).
- Le langage binaire étant incompréhensible pour un être humain alors comment communiquer plus simplement avec la machine.
- L'idée est venue d'inventer des langages qui seront plus facile à comprendre par l'être humain et qui seront traduits en binaire.
- Le hic, c'est de réaliser le programme de traduction.

PROBLÉMATIQUE

- Heureusement d'autres ont déjà écrit ce programme. On se contentera tout simplement de s'en servir.
- Ce programme s'appelle compilateur. La traduction, elle, s'appelle la compilation. Le programme binaire crée par le compilateur est l'exécutable.
- Les langages « inventés » sont dits plus ou moins des langages de haut niveau. Il en existe une multitude si bien qu'à l'heure actuelle il est impossible d'en déterminer le nombre total.
- Le programme que nous écrivons dans un langage haut niveau est appelé code source

OUTILS NÉCESSAIRES

- Les outils nécessaires pour programmer sont :
  - Un éditeur de texte pour l'écriture du code source.

    Exemple : le bloc-notes
  - Un compilateur pour compiler(transformer) le code source(programme) en exécutable(code binaire).
  - Un debugger pour traquer les erreurs dans le programme.

OUTILS NÉCESSAIRES

- Pour programmer donc on peut procéder de deux manières :
  - La première consiste à récupérer chacun des trois outils séparément. Cette méthode est compliquée et est surtout utilisée sous linux.
  - La seconde consiste à l'utilisation d'un programme 3 en 1 qui intègre les outils ci-dessus cités. Un tel programme s'appelle IDE (Integrated Development Environnement) ou EDI(Environnement de Développement Intégré).

LES DIFFÉRENTS TYPES DE LANGAGES DE PROGRAMMATION

- Les premiers langages de programmation sont apparus autour des années 1950.
- Chacun ayant la possibilité de créer son langage, il existe une multitude de langages de programmation. Si bien qu'il est impossible à l'heure actuelle de déterminer le nombre total de langages existants.
- Ces langages sont classés selon le style auquel ils appartiennent ou en fonction de leurs utilisations.
- Exemple de langages : C, C++, HTML, JAVA, XML

DESCRIPTION DU LANGAGE C

- Le langage C est apparu au cours de l'année 1972 dans les Laboratoires Bell. Il a été développé par Kenneth Thompson et Dennis Ritchie.
- Il a été inventé pour écrire le système d'exploitation Unix et a gardé de ce fait une très grande efficacité en tout ce qui concerne le développement système. Ainsi le noyau de grands systèmes d'exploitation tel que Windows ou Linux ont été développé en grande partie en C.
- C 'est actuellement le langage le plus utilisé au monde(index tiobe d'avril 2012).

LANGAGE C/CODE MINIMAL

- Le programme C le plus simple que l'on puisse écrire est :

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5 return 0;
6 }
```

Ce programme est composé de mots clés tels include, int, return et main

Il présente aussi des symboles tels «#», «(» , «)», «{», «}», «;»

Pour n'importe quel programme il faudra ce minimum de code. De nos jours les EDI le génèrent et même avec d'autres éléments en plus.

LANGAGE C/CODE MINIMAL

- Que signifient les deux première lignes?
- Ce sont les directives du **préprocesseur**

le préprocesseur est un programme qui se lance au début de la compilation

LANGAGE C/CODE MINIMAL

- Facilement reconnaissables car elles débutent par un «#»(dièse)
- Elles ont pour rôle ici d'inclure les bibliothèques (librairies)
- Les bibliothèques sont des fichiers qui existent déjà et qui contiennent du code tout prêt pour nous permettre de faire certaines choses facilement tel afficher une valeur sur l'écran

LANGAGE C/CODE MINIMAL

- int un mot-clé du C est le type pour désigner les entiers naturels.
- Placé en début d'une fonction signifie que celle-ci doit retourner un entier à la sortie
- Une fonction est un ensemble de commandes pour faire quelque chose de précis
- main en est une et est même la plus importante car c'est toujours par cette fonction que va débuter un programme
- Ces commandes disposées (pas forcément) en lignes sont dites instructions. Une instruction est cette ligne à l'intérieur d'une fonction qui se termine par un point-virgule

LANGAGE C/MOTS CLÉS

- Ce sont des mots réservés dans un langage à un usage bien défini. Ils ne peuvent pas être utilisés comme des identificateurs.
- Il existe un bon nombre de mots clés dont char, int, break, if, case, continue, etc

LANGAGE C/CODE MINIMAL

- Int un mot-clé du C est le type pour désigner les entiers naturels.
- Placé en début d'une fonction signifie que celle-ci doit retourner un entier à la sortie
- Une fonction est un ensemble de commandes pour faire quelque chose de précis
- main en est une et est même la plus importante car c'est toujours par cette fonction que va débuter un programme
- Ces commandes disposées (pas forcément) en lignes sont dites instructions. Une instruction est cette ligne à l'intérieur d'une fonction qui se termine par un point-virgule

LANGAGE C/CODE MINIMAL

- « {», « } » désignent respectivement le début et la fin d'une suite d'instructions.
- return 0; est l'instruction qui signifie qu'on est à la fin de notre fonction main et demande de renvoyer la valeur 0.
- C'est juste une manière de s'assurer que le programme s'est bien passé (0 si tout s'est bien passé et autre valeur sinon)

LANGAGE C/ COMMENTAIRES

- Comme tout langage évolué, le C autorise la présence de commentaires. Il s'agit de textes explicatifs destinés au lecteur du programme.
- Ce texte n'a donc aucune incidence sur la compilation du programme. Il existe maintenant deux manières d'exprimer un commentaire:
- /*commentaire*/ : sied quand on veut mettre en commentaire tout un bloc d'instructions
- //commentaire: convient pour la mise en commentaire d'une seule instruction

LANGAGE C/OPÉRATEURS

- Arithmétiques
  - +: addition
  - -: soustraction
  - *: multiplication
  - /: division entière
  - %: modulo
- Relationnels
  - <: inférieur
  - <= : inférieur ou égal
  - >: supérieur
  - == : égal
  - != : différent

LANGAGE C/OPÉRATEURS

- Logiques
  - Et : &&
  - Ou: ||
  - Non: !
- Exemples
  - (a<b) && (c<d) vrai si a<b et si c<d
  - (a<b) || (c<d) vrai si a<b ou si c<d c'est-à-dire vrai si l'une des conditions est vraie
  - !(a<b) vrai si a>=b

LANGAGE C/OPÉRATEURS

- Opérateurs d'incrémentation

- ++i: expression qui incrémente(augmente) la valeur de i et sa valeur est celle de i après incrémentation. Elle dite pré-incrémentation

- Exemple:

```
n= ++i - 5;
Si i=5  alors n=1
```

Explication	
i=i+1;	i=5+1=6
n=i-5;	n=6-5=1

- i++: expression qui incrémente(augmenter) la valeur de i et sa valeur est celle de i après incrémentation. Elle dite post-incrémentation

- Exemple:

```
n= i++ - 5;
Si i=5 alors n=0
```

Explication	
n=i-5;	n=5-5=0
i=i+1;	i=5+1=6

LANGAGE C/OPÉRATEURS

- Opérateurs de décrémentation

- --i: expression qui décrémente(diminue) la valeur de i et sa valeur est celle de i après décrémentation. Elle dite pré-décrémentation

- Exemple:

```
n= –i - 5;
Si i=6 alors n=0
```

Explication	
i=i-1;	i=6-1=5
n=i-5;	n=5-5=0

- i--: expression qui décrémente(diminue) la valeur de i et sa valeur est celle de i après décrémentation. Elle dite post-décrémentation

- Exemple:

```
n= i– - 5;
Si i=6 alors n=1
```

Explication	
n=i-5;	n=6-5=1
i=i-1;	i=6-1=5

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- Déclaration de variables

- Type nom_variable;
- nom_variable :nom ou identifiant qui permet de reconnaître la variable
- Type : ce qui va permettre l'allocation optimisée d'une taille mémoire à la variable

- **Quelques contraintes du nom d'une variable**

- Le nom doit commencer par une lettre
- Les espaces sont interdits dans les noms composés
- Les accents ne sont pas permis
- Il est préférable que les noms soient significatifs pour le programmeur

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- Initialisation de variables
- nom_variable=valeur;
- nom_variable: nom de la variable
- valeur: la valeur que la variable doit stocker
- = le est le signe de l'affectation

- NB: l'initialisation se fait en début de programme

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- Le langage c fournit des types permettant de manipuler des nombres (entiers et réels) ou des caractères et permet de construire des types dérivés(tableau, structure). Il existe donc des types pour manipuler:
  - Des nombres entiers
    - petite taille:
      - Signés : char
      - Non signés: unsigned char
    - Taille Moyenne
      - Signés: short
      - Non signés: unsigned short
    - Grande taille
      - Signés: long
      - Non signés: unsigned long

Type particulier à détailler plus bas

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- NB: **signed** et **unsigned** sont des modificateurs qui permettent d'obtenir un type signé (le signe du nombre est représenté)et non signé (seuls les nombres nuls ou positifs sont représentés)

- La norme C contraint les domaines de valeurs des types signés entre $-(2^{N-1}-1)$ et $2^{N-1}-1$, où N est un entier quelconque, et les types non signés entre 0 et $2^{N}-1$. **N** est le nombre de bits et dépend de l'implémentation

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- Tableau des domaines de valeurs minimales des types entiers

type	Taille minimale en octet(8bits)	Valeurs stockables
char	1	-127 à +127
unsigned char	1	0 à +255
short	2	-32767 à +32767
unsigned short	2	0 à +65535
int	2	-32767 à +32767
unsigned int	2	0 à +65535
long	4	-2 147 483 647 à +2 147 483 647
unsigned long	4	0 à +4 294 967 295

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- Des nombres réels ou flottants
  - Simple : float
  - Double précision: double
  - Précision encore plus grande: long double

Type	Taille minimale en octets(8bits)	Valeurs stockables
float	4	$1,2 \times 10^{-38}$ à $3,4 \times 10^{+38}$
double	8	$2,2 \times 10^{-308}$ à $1,8 \times 10^{+308}$

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- Des caractères
- char est le type qui permet de représenter un caractère
- Une constante représentant un caractère est délimitée par des apostrophes. Exemple: ‘a’, ‘1’
- En c les caractères ne sont ni plus ni moins que des nombres. Il est possible de faire ‘a’-5. on aura donc l’entier correspondant qui est 92 et le caractère correspondant qui est \
- La constante a une valeur définie par un jeu de caractères compatibles avec ASCII

ASCII(AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)

- Le jeu de caractères codés **ASCII** est la norme de codage de caractères en informatique la plus connue, la plus ancienne et la plus largement compatible. ASCII contient les caractères nécessaires pour écrire en anglais.

- L'ASCII définit 128 caractères numérotés de 0 à 127 et codés en binaire de 0000000 à 1111111.7 bits suffisent donc pour représenter un caractère codé en ASCII.

ASCII(ASCII(AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)

- Toutefois, les ordinateurs travaillant presque tous sur un multiple de huit bits (octet)
- chaque caractère d'un texte en ASCII est stocké dans un octet dont le 8^{e} bit est 0.
- Les caractères de numéro 0 à 31 et le 127 ne sont pas affichables ; ils correspondent à des commandes de contrôle de terminal informatique. Le caractère numéro 32 est l'espace. Les autres caractères sont les chiffres arabes, les lettres latines majuscules et minuscules et quelques symboles de ponctuation.

ASCII(ASCII(AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)

- Extrait du tableau de caractères ASCII

Code base 10	caractères
48	Le chiffre zéro
. .	. .
65	A
. .	
90	Z
. .	. .

Code base 10	caractères
97	a
98	b
. .	. .
122	z

Langage c/représentation des variables et leurs types

Caractères spéciaux

Des constantes caractères avec des significations particulières

constante	caractère
‘ \’’	Une aspostrophe
‘\ »’	Un guillement
‘\?’	Un point d’interrogation
‘\\’	Un backslash
‘\b’	Un espace arrière
‘\f’	Un saut au début de la page suite
‘\n’	Un saut de ligne
‘\t’	Une tabulation

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- Des booléens

- La norme c99 introduit le type bool et les valeurs true et false. La définition du type bool est faite dans le fichier d'en-tête *stdbool.h*
- Avant il n'y avait pas de type booléen. Ce qui se faisait et se fait toujours: la valeur entière 0 prend la valeur de vérité faux et toutes les autres prennent la valeur de vérité vrai

- Exemple:

Avec le type bool

```
bool b;
b = true;
 if (b)...
```

Sans le type bool

```
typedef enum {
FALSE, TRUE} BOOL;
BOOL b;
b=FALSE;
if(b)...
```

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- Chaine de caractères

- Il n'existe pas de type spécifique pour désigner une chaine de caractères

- Une chaine va être donc représentée en C par un tableau de char dont le dernier élément est le caractère nul '\0'

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- Déclaration d'une chaine de caractères

- char chaine[taille]; //taille représente la zone réservée par le compilateur pour stocker les éléments.

- char chaine[]; // ici la taille appropriée est déduite par le compilateur.

- char *chaine; //ici pas de réservation de mémoire, seule l'adresse de la chaine est stockée.

- NB: taille = nombre de caractères voulus+1 pour éviter des comportements indéfinis

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- Initialisation d'une chaine de caractères

- chaine[8]=« bonjour »;
- chaine[8]={'b', 'o', 'n', 'j', 'o', 'u', 'r', '\0'};
- Chaine[]=« bonjour »;

MANIPULATION DES CHAINES DE CARACTÈRES

- Il existe des fonctions de la bibliothèque standard qui permettent de manipuler les chaines de caractères. Mais avant il faut inclure le fichier d'entête string.h avant de pouvoir les utiliser. Ce sont entre autre:
  - **char *strcpy (char *s, const char *ct);**

Copie la chaîne *ct* dans *s* (y compris le caractère de fin de chaîne), retourne *s*.

  - **char *strncpy (char *dst, const char *src, size_t n);**

identique à strcpy à la différence que le nombre d'éléments est limité par n.

MANIPULATION DES CHAINES DE CARACTÈRES

- **char *strcat (char *s, const char *ct);**

concatène la chaine ct à la suite de la chaine s.

- **int strcmp (const char *cs, const char *ct);**

Compare les chaînes *cs* et *ct* et renvoi:

  - Une valeur négative si, lexicalement, *cs* < *ct*
  - Une valeur nulle si *cs* == *ct*
  - Une valeur positive si *cs* > *ct*

- **size_t strlen (const char *cs);**

Retourne le nombre de caractères de *cs* sans tenir compte du caractère de fin de chaîne

- Bien d'autres (voir sur le net)

LANGAGE C/REPRÉSENTATION DES VARIABLES ET LEURS TYPES

- En plus des types cités, le langage C fournit un autre type , void qui représente rien, le vide.
- Il n'est pas possible de déclarer une variable de ce type
- Il est utile au niveau des fonctions et des pointeurs

LANGAGE C/REPRÉSENTATION DES ACTIONS

- Actions élémentaires

- = est le signe qui permet de désigner l'affectation d'une valeur à une variable

```
Exemple:
Int main()
{
int n;//déclaration de la variable n
n=10;//affectation de la valeur 10 à n
Return 0;
}
```

- NB: la déclaration et l'affectation peuvent se faire en une seule instruction comme suit: int n=10;

LANGAGE C/REPRÉSENTATION DES ACTIONS

- Ecriture d'informations

- **printf** est l'instruction qui permet l'écriture d'une valeur sur l'écran.
- la valeur à afficher est entre deux doubles quottes
- La valeur à afficher peut être du texte simple ou la valeur d'une variable

- Affichage de texte simple

- Exemple :

```
int main()
{
printf(«  bonjour»); //affiche bonjour sur l'écran
return 0;
}
```

LANGAGE C/REPRÉSENTATION DES ACTIONS

- Affichage de la valeur d'une variable
- On utilise printf de la même manière que pour afficher un texte sauf qu'on rajoute un symbole spécial à l'endroit où on veut afficher la valeur de la variable
- Le symbole est % suivi d'une lettre
  - La lettre d désigne un entier
  - La lettre f désigne un réel
  - La lettre c désigne un caractère
  - La lettre s désigne une chaine de caractères

LANGAGE C/REPRÉSENTATION DES ACTIONS

- Exemple

```
int main()
{
int n=10;
float m=10.5;
char x='c';
char nomFamille []=«ouedraogo»;
printf(« valeur=%d »,n);
printf(« valeur=%f »,m);
printf(« valeur=%c »,x);
printf(« valeur=%s »,nomFamille);
return 0;
}
```

LANGAGE C/REPRÉSENTATION DES ACTIONS

- Lecture d'informations

- scanf est l'instruction qui permet la lecture (ce qui est tapé depuis le clavier)d'une information

- Scanf à l'image de printf possède un caractère spécial qui désigne le type de la variable à renseigner

- scanf a aussi un autre caractère spécial pour désigner l'emplacement mémoire (appelée adresse) de la variable saisie. Son symbole est **&**

LANGAGE C/REPRÉSENTATION DES ACTIONS

- Exemple :

```
int main()
{
int n;
printf(« saisissez un nombre entier svp »);
scanf(« %d »,&n);
printf(« le nombre saisi est :%d »,n);
printf(« le double du nombre saisi est :%d »,n*2);
return 0;
}
```

- Les caractères spéciaux utilisés par scanf sont identiques à ceux de printf(là il faut les souligner aussi qu'il existe d'autres)

LANGAGE C/REPRÉSENTATION DES ACTIONS

- A savoir

- Il existe d'autres instructions de lecture/ecriture:
- putch: permet d'afficher un seul caractère à l'écran
- getch : permet de lire un seul caractère depuis le clavier.

Exemple de putch

```
int main()
{
char x;
x='e';
printf(« lettre =»);
putch (x);
return 0;
}
```

Resultat: lettre=e

Exemple de getch

```
int main()
{
char c;
printf(« donnez une lettre: »);
c=getch();
printf(« merci pour la lettre %c»,c);
return 0;
}
```

LANGAGE C/REPRÉSENTATION DES ACTIONS

- A savoir

- printf et scanf permettent de lire et d'afficher simultanément des informations de type quelconque.
- Par contre gets et puts ne traitent qu'une chaine à la fois.
- De plus la délimitation de la chaine lue ne s'effectue pas de la même façon.
- Avec le code de format %s de scanf, les délimiteurs sont classiquement l'espace, la tabulation ou la fin de ligne.
- Avec la fonction gets, seule la fin de ligne sert de délimiteur.

Exemple de printf et scanf

```
int main()
{
char ville[25];
printf(«saisir nom d'une ville»);
scanf («%s»,ville);
Printf(«%s»,ville);
return 0;
}
```

Resultat: si on saisit Bobo Dioulasso, on verra seulement Bobo sur l'écran

Exemple de gets et puts

```
int main()
{
char ville[25];
printf(«saisir nom d'une ville»);
gets(ville);
puts(ville)
return 0;
}
```

resultat: si on saisit Bobo Dioulasso, on verra Bobo Dioulasso sur l'écran

Possible

printf(« la ville saisie est %s »,s);
printf(«%d»,n*n); si on initialise n comme un entier dans le programme

impossible

puts(« la ville saisie est %s »,s);
puts(«%d»,n*n); si on initialise n comme un entier dans le programme

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Structures de contrôle itératives
  - Si condition Alors
    - Elle est traduite par l'instruction if
    - Sa syntaxe est la suivante:

```
if(expression ou condition)
{
Bloc d'instruction
}
```

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Exemple

```
int main()
{
int a,x;
printf(« entrez un nombre divisible par  10 »);
scanf(« %d »,&a);
if(a!=0)
{
x=10/a;
printf(« %d »,x);
}
return 0;
}
```

Résultat: si a est différent de 0, la division est possible dans le cas contraire, rien ne se passe

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Structures de contrôle itératives
  - SI condition Alors SINON
    - Elle est traduite par l'instruction if else
    - Elle se présente comme suit:

```
if(expression ou condition)
{
Bloc d'instruction
}
else
{
Bloc d'instruction
}
```

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

❖ Exemple

```
int main()
{
int a,x;
printf(«entrez un nombre divisible par 10»);
scanf(«%d »,&a);
if(a!=0)
{
x=10/a;
printf(« %d »,x);
}
```

```
else
{
printf(« la division par zéro est impossible »);
}
return 0;
}
```

Résultat: si a est différent de 0, la division est possible dans le cas contraire, il est indiqué qu'il n'est pas possible d'avoir un dénominateur nul

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- NB : on peut imbriquer des if, c'est-à-dire qu'on peut faire des if à l'intérieur d'un autre if

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

Exemple d'imbrication de if

```
int main()
{
int a,x;
printf(«entrez un nombre  divisible par  10»);
scanf(« %d »,&a);
if(a==0)
{
printf(«la division par zéro est impossible»);
}
else
{
        if(a>10)
        {
printf(«entrez un nombre inférieur à 10»);
        }
else
    {
        if(a<0)
        {
printf(«entrez un nombre positif svp»);
        }
        else
        {
        x=10/a;
        printf(« %d »,x);
        }
    }
}
return 0;  }
```

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Étude de cas

- Ecrire un programme qui permet de lire le prix hors taxe et calculer le prix TTC correspondant à un taux de TVA de 19%. La remise est établie de la manière suivante:
- 0% pour un montant TTC<1000fcfa
- 1% pour un montant TTC>=1000 et <2000 fcfa
- 3% pour un montant TTC >=2000 et <5000 fcfa
- 5% pour un montant TTC >=5000

PROGRAMME EN C

```
int main()
{
float pht, pttcn,r ,pttc;
Printf(« donnez le prix hors taxe);
scanf(« %f »,&pht);
pptc=pht + (19*pht)/100 ;
if(pptc<1000)
        {
                r=0;
                pttcn=pttc-r;
        }
else
{
    if(pttc>=1000 && pttc<2000)
        {
        r = pttc/100;
        pttcn=pttc-r;
        }
```

```
else{
        if(pttc>=2000 && pttc<5000)
          {
                r = (3*pttc)/100;
                pttcn=pttc-r;
          }
        else
          {
          r=(5*pttc)/100;
          pttcn=pttc-r;
          }
}
}
printf(« le prix ttc brut est %f, la remise est: %f, le prix ttc net est %f »,pttc,r,pttcn);
 return 0;
}
```

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

Choix multiples

- Cas expression vaut
  - Elle se traduit par l'instruction switch
  - Sa syntaxe est la suivante:

```
switch(expression)
{
Case constante 1: bloc d'instruction 1
.
.
Case constante n: bloc d'instruction n
default: bloc d'instruction par défaut
}
```

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

❖ Exemple de switch

```
int main
{
int n;
printf(« donner un nombre correspondant à un jour de la semaine\n»);
scanf(« %d»,&n);
switch(n)
        {
        case 1: printf(«Lundi»);
                        break;
        case 2:printf(«Mardi»);
                        break;
        case 3:printf(«Mercredi»);
                        break;
case 4:printf(«Jeudi»);
        break;
case 5:printf(«Vendredi»);
        break;
case 6:printf(«Samedi»);
        break;
case 7:printf(«Dimanche»);
        break;
default: printf («ce numéro ne correspond à aucun jour de la semaine »);
        }
return 0;
}
```

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Résultat : selon la valeur de l'expression n, l'exécution se poursuit au bloc correspondant à la même valeur de case. Si n=1, c'est lundi qui est affiché à l'écran car c'est l'étiquette case 1 qui est concernée
- L'intérêt de l'instruction break est de sortir du block switch
- L'étiquette default permettra l'exécution d'un bloc d'instructions au cas où aucune valeur satisfaisante n'a été rencontrée auparavant . Elle n'est pas obligatoire.

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Boucles
  - Tant que
    - Elle est traduite par while
    - sa syntaxe est la suivante:

```
while(condition)
{
      bloc d'instructions
}
```

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Exemple: additionner des nombres jusqu'à atteindre au moins 100

```
int main()
{
int n, som;
som=0;
while(som<100)
{
printf(«donnez un nombre : » );
scanf(« %d »,&n);
som=som + n;
}
printf(«  la somme =%d»,som);
return 0;
}
```

Resultat: tant que la variable som est inférieure à 100, on effectue l'opération d'addition

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Etude de cas: donnez le programme en C de la somme des entiers naturels avec la boucle tant que

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Solution de l'étude de cas

```
int main()
{
int s=0,n;
printf(«  entrez un nombre entier svp\n»)
scanf(«  %d  »,&n);
while(n>0)
 {
            s=s+n;
            n=n-1;
 }
 printf(« la somme des n entiers est : %d »,s);
 return 0;
}
```

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Boucles
  - Faire Tant que
    - Elle est traduite par do while
    - Un peu moins utilisée, elle est similaire à while sauf que la position de la condition change. La condition est à la fin au lieu d'être au début
    - sa syntaxe est la suivante:

```
do
{
        bloc d'instructions
}
while(condition);
```

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Exemple de la boucle Faire Tant Que : afficher n nombres entiers

```
int main()
{
int i=0,n;
printf(« entrez un nombre »);
scanf(« %d »,&n);
do
{
i=i+1;
printf(« %d»,i);
}
while(i<n);
}
```

Résultat : seront affichés les entiers compris entre 1 et n

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Etude de cas
- Somme des n entiers avec la boucle faire tant que

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Solution de l'étude de cas

```
int main()
{
int n, som;
som=0;
printf(«entrez un nombre un nombre svp\n»);
scanf(« %d »,&n);
do
{
som=som+n;
 n=n-1;
}
while(n>0);
printf(«la somme des n entiers  est %d»,som);
}
```

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

- Boucles
- La boucle pour
  - Elle est traduite par l'instruction for
  - Sa syntaxe est

```
for([expression1];[expression2];[expression])
        {
                bloc d'instruction
        }
```

LANGAGE C/REPRÉSENTATION DES STRUCTURES DE CONTRÔLE

❖ Exemple de la boucle for

```
int main()
{
int i,n;
printf(«entrez le nombre de fois à dire bonjour»);
scanf(«%d»,&n);
for(i=0;i<n;i++)
    {
        printf(«bonjour»);
    }
return 0;
}
```

Résultat: il y aura autant de bonjour à l'écran que la valeur n entrée

LANGAGE C/TABLEAUX

- Le C permet l'utilisation des tableaux. Un tableau est un ensemble d'éléments de même type désigné par un identificateur unique. Chaque élément est repéré par un indice précisant sa position au sein de l'ensemble.
- Déclaration d'un tableau monodimensionnel :

```
type nom_du_tableau[taille];
```

- Exemple : `float tab[10];`

- On accède à l'élément d'indice i d'un tableau nommé T par la notation T[i].

LANGAGE C/TABLEAUX

- Le C permet également la déclaration des tableaux multidimensionnels.
- Déclaration d'un tableau multidimensionnel

```
type nom_du_tableau[taille 1][taille 2];
```

- Exemple : int tab[5][4];

- Chaque élément du tableau est repéré par deux indices.

LANGAGE C/TABLEAUX

- On peut initialiser un tableau monodimensionnel grâce à la syntaxe suivante :

  type nom_du_tableau[taille]={valeur 1, ..., valeur n};

- Exemple : int tab[4] = {12, 45, 5, 1};

- La déclaration de l'exemple permet de placer les nombres entre les accolades dans chacun des quatre éléments du tableau.

LANGAGE C/TABLEAUX

- Remarque : il n'est pas obligé d'indiquer toutes les valeurs du tableau. Il est possible de ne mentionner dans les accolades que certaines valeurs du tableau.

- Exemple :
- int tab[4] = {12,45};
- int tab[4] = {12, ,5,1};

LANGAGE C/TABLEAUX

- L'initialisation d'un tableau multidimensionnel peut se faire de deux manières. La première se présente comme suit :

```
type nom_tableau[taille 1][taille 2] =
{{val11,val12,...,val1n},...,{valn1,valn2,...,valnn}};
```

- Exemple :

```
int tab[2][3]={{1,45,3},{4,12,19}};
```

- Dans ce premier cas le tableau est considéré comme formé de deux tableaux de 3 éléments.

LANGAGE C/TABLEAUX

- La deuxième méthode se présente de la manière suivante :

type nom_du_tableau[taille 1][taille 2] = {val1, val2, val3,..., valn};

- Exemple : int tab[2][3] = {1,45,3,4,12,19};

- Cette deuxième méthode exploite la manière dont les éléments sont rangés en mémoire et se contente d'énumérer les valeurs du tableau suivant cet ordre.

LANGAGE C/TABLEAUX

Etude de cas :

Traduire en langage C l'algorithme qui à partir de 20 notes d'étudiants fournies en donnée détermine combien d'entre elles sont supérieures à la moyenne de la classe.

LANGAGE C/TABLEAUX

- Traduction de l'étude de cas en langage C :

```
int main()
{
    int i,cpt = 0;//compteur
    float som = 0;//somme
    float moy;//moyenne
    float t[10];
    for(i=0;i<10;i++){
        printf("entrez la note %d : ",i+1);
        scanf("%f",&t[i]);
        printf("\n");
        som = som + t[i];
    }
```

```
moy=som/10;
    for(i=0;i<10;i++){
        if(t[i]>moy){
            cpt++;
        }
    }
    printf("le nombre de notes
supérieur à la moyenne est %d", cpt);
    return 0;
}
```

LANGAGE C/STRUCTURES

- Rappel : la structure ou enregistrement permet de designer sous un seul nom un ensemble de valeur pouvant être de type différent.
- Déclaration d'une structure

```
struct  nom_structure
{
        type champ;
        [type champ;
        [....]]
};
```

LANGAGE C/STRUCTURES

- Exemple : déclaration d'une structure 'produit ' qui contient les champs numero, quantite, prix.

```
struct produit
{
        int numero;
        int quantite;
        float prix;
}
```

LANGAGE C/STRUCTURES

- Remarque : l'exemple précédent définit un modèle de structure mais ne réserve pas de variable correspondant à cette structure. Pour ce faire il faut déclarer les variables correspondant au modèle.

- Pour déclarer des variables correspondants au modèle on procède comme suit :

```
struct nom_structure nom_variable;
```

- Exemple : struct produit art1;

LANGAGE C/STRUCTURES

- Il est également possible de regrouper la définition du modèle de structure et la déclaration des variables de ce type dans une seule instruction.

```
struct produit
{
        int numero;
        int quantite;
        float prix;
}art1;
```

LANGAGE C/STRUCTURES

- Utilisation des champs d'une structure

- Chaque champ d'une structure peut être manipulé comme n'importe quelle variable du type correspondant. La désignation d'un champ se note en faisant suivre le nom de la variable structure du nom de champ tel qu'il a été défini dans le modèle.

LANGAGE C/STRUCTURES

- Exemple : écrire un programme qui permet de saisir cinq produits et de les afficher.

LANGAGE C/STRUCTURES

```
struct produit{
    int numero;
    int quantite;
    float prix;
};
int main()
{
    struct produit article[5];
    int i;
    //saisi des cinq articles de type produit
    for(i=0;i<5;i++){
        printf("entrez le numero de l'article n°%d : ",i+1);
        scanf("%d",&article[i].numero);
        printf("entrez la quantite de l'article n°%d : ",i+1);
        scanf("%d",&article[i].quantite);
        printf("entrez le prix de l'article n°%d : ",i+1);
        scanf("%f",&article[i].prix);
    }
```

LANGAGE C/STRUCTURES

```
//affichage des différents elements
    for(i=0;i<5;i++){
        printf("article %d\n",i+1);
        printf("numero : %d\tquantite : %d\tprix : %f\n",article[i].numero,
                article[i].quantite,article[i].prix);
    }
    return 0;
}
```

LANGAGE C/STRUCTURES

- Utilisation globale d'une structure

- Il est possible d'affecter à une structure le contenu d'une structure définie à partir du même modèle.
- Exemple : prenons l'exemple de la structure définie précédemment :
- struct produit pdt1, pdt2;
- pdt1 = pdt2;

LANGAGE C/STRUCTURES

- Initialisation des structures

- Il existe plusieurs manières d'initialiser les valeurs d'une structure. L'une d'entre elles consiste à l'initialiser lors de la définition du modèle.
- Exemple :

```
struct produit
{
        int numero;
        int quantite;
        float prix;
}art1={15,205,1500.85};
```

LANGAGE C/STRUCTURES

- Une autre manière consiste à l'initialiser pendant la déclaration.
- Exemple :

```
struct produit art1 = {15,205,1500.85};
```

LANGAGE C/STRUCTURES

- on cherche à dériver de façon automatique des polynômes.
- A) proposer une structure pour représenter un polynôme afin qu'il soit exploitable par ordinateur.
- B) écrire un programme en C qui permet de dériver un polynôme

LANGAGE C/STRUCTURES

- Traduction de l'étude de cas en langage C

```
struct monome{
    float coefficient;
    int exposant;
};

int main()
{
    struct monome a[5];
    int i,n;
    printf("entrez le nombre de monome du polynome : ");
    scanf("%d",&n);
```

LANGAGE C/STRUCTURES

```
//saisi des monomes
   for(i=0;i<n;i++){
      printf("donnez un coefficient : ");
      scanf("%f",&a[i].coefficient);
      printf("\n");
      printf("donnez un exposant : ");
      scanf("%d",&a[i].exposant);
   }
//deviration
   for(i=0;i<n;i++){
      a[i].coefficient = a[i].coefficient * a[i].exposant;
      a[i].exposant = a[i].exposant - 1;
   }
```

LANGAGE C/STRUCTURES

```
//affichage
for(i=0; i<n; i++){
        if(i==n-1){
            if(a[i].coefficient!=0){
                printf("%f X %d",a[i].coefficient,a[i].exposant);
            }
        }else{
            if(a[i].coefficient!=0){
                printf("%f X %d +",a[i].coefficient,a[i].exposant);
            }
        }
}
    return 0;
}
```

LANGAGE C

- Procédures et fonctions

LANGAGE C/FONCTIONS

- Beaucoup de langages de programmation permettent l'utilisation de sous-programmes appelés modules.
- Cette manière de découper les programmes en modules présentent des avantages comme :
  - La maitrise d'un programme qui dépasse une dizaine de pages
  - L'évitement des instructions répétitives
  - Le partage des outils écrits et mis au point une fois.

LANGAGE C/FONCTIONS

- On distingue deux types de modules à savoir :
  - La fonction : assez proche de la notion mathématique correspondant comme sin(x) et cos(x) fournit un unique résultat.
  - La procédure qui élargit la notion de fonction, reçoit et produit une ou plusieurs informations.

LANGAGE C/FONCTIONS

- En langage C, il n'existe qu'une seule sorte de module qui est la fonction. On l'utilise à la manière d'une fonction mathématique et pourtant cette fonction pourra jouer un rôle aussi général que la procédure des autres langages.

LANGAGE C/FONCTIONS

Déclaration d'une fonction:

- La syntaxe C pour déclarer une fonction est :

```
type nom_fonction(type arg 1, type arg 2, ..., type arg n)
{
        liste d'instructions;
}
```

LANGAGE C/FONCTIONS

- L'entête de la fonction définit le type des objets qu'elle reçoit ou qu'elle renvoie. Cela peut être tout type primitif(int, float, double, …) ou tout type struct (structures).
- Si le type de la fonction n'est pas indiqué, le compilateur suppose qu'il s'agit d'une fonction à résultat entier.
- NB: cette manière de faire(ne pas indiquer le type) est déconseillée.

LANGAGE C/FONCTIONS

- Exemple : fonction permettant de dire « bonjour »

```
void direBonjour()
{
        printf(« bonjour »);
}

Int main()
{
        //appel de la fonction direBonjour
        direBonjour();
        return 0;
}
```

LANGAGE C/FONCTIONS

- Arguments d'une fonction

- Les arguments se situant dans la définition de l'entête de la fonction n'ont de signification qu'au sein de la fonction et n'ont aucun rapport avec d'éventuelles variables qui pourraient être définies en dehors de la fonction. On parle alors dans ce cas d'**argument muet** ou **argument formel**.
- Lors de l'appel de la fonction les arguments utilisés sont appelés arguments effectifs et peuvent prendre la forme de n'importe quelle expression.

LANGAGE C/FONCTIONS

- Exemple :

```
void ecris(int n)
{
    Printf(« valeur : %d\n »,n);
}

int main()
{
    int a = 10, b = 20;
    ecris(a);
    ecris(b);
    ecris(a+b);
}
```

Argument muet ou formel

Arguments effectifs

LANGAGE C/FONCTIONS

- Fonctions fournissant un résultat
- Exemple : fonction permettant d'additionner deux nombres

```
int addition(int a, int b)
{
        int resultat;
        resultat=a + b;
        return resultat;
}

int main(){
int n,m,res;
printf(« entrez le premier nombre\n »);
scanf(« %d »,&n);
printf(« entrez le deuxième nombre\n »);
scanf(« %d »,&m);
res = addition(n,m);
printf(« la somme de %d et %d est %d », n, m, res);
return 0;
}
```

LANGAGE C/FONCTIONS

- Le mot clé **int** placé avant le nom de la fonction dans l'entête de la fonction « addition » indique le type de résultat que fournira celle-ci.

```
int addition(int a,  int b){
.....
}
```

- L'instruction return spécifie le résultat qui sera fourni par la fonction lors de son appel.

```
int addition(int a,  int b){
.....
return resultat;
}
```

LANGAGE C/FONCTIONS

- Fonctions sans résultat

- Lorsqu'une fonction ne renvoie pas de résultat c'est-à-dire quand elle correspond à une procédure on la déclare comme renvoyant un objet de type void.

- Toute tentative d'utilisation du résultat de la fonction provoquera une erreur à la compilation.

LANGAGE C/FONCTIONS

- Exemple : la fonction suivante ne renvoie aucun résultat.

```
void direBonjour()
{
        printf(« bonjour »);
}

Int main()
{
        direBonjour();
        return 0;
}
```

LANGAGE C/FONCTIONS

- Les prototypes

- Lorsqu'une fonction fournit un résultat, le fait de placer sa définition après les fonctions qui l'utilisent implique l'obligation d'introduire des déclarations supplémentaires. On définit ce que l'on appelle un prototype.
- Le prototypage consiste à déclarer l'entête de la fonction avant la fonction principale et à donner la définition après.
- Le prototypage permet de corriger les erreurs de déclaration du type entre les arguments.

LANGAGE C/FONCTIONS

- Exemple : utilisation de prototype pour la fonction permettant d'additionner deux nombres.

```
int addition(int a, int b);

int main(){
Int n,m,res;
printf(« entrez le premier nombre\n »);
scanf(« %d »,&n);
printf(« entrez le deuxième nombre\n »);
scanf(« %d »,&m);
res = addition(n,m);
printf(« la somme de %d et %d est %d », n, m, res)
return 0;
}
int addition(int a, int b)
{
        int resultat;
        resultat=a + b;
        return resultat;
}
```

LANGAGE C/FONCTIONS

- Etude de cas :

- Ecrire :
  - une fonction, nommée f1, se contentant d'afficher ''bonjour'' (elle ne possédera aucun argument ni valeur de retour),
  - une fonction, nommée f2 qui affiche "bonjour" un nombre de fois égal à la valeur reçue en argument (int) et qui ne renvoie aucune valeur,
  - une fonction, nommée f3 qui fait la même chose que f2 mais qui, de plus, renvoie la valeur (int) 0.
- Ecrire un petit programme appelant successivement chacune de ces trois fonctions, après les avoir convenablement déclarées sous forme d'un prototype.

LANGAGE C/FONCTIONS

```
void f1()
{
printf(«bonjour»);
}
```

```
void f2(int n)
{
int i;
for(i=0;i<n;i++)
{
printf(«bonjour»);
}
}
```

```
int f3(int n)
{
int i;
for(i=0;i<n;i++)
{
printf(«bonjour»);
}
return 0;
}
```

LANGAGE C/FONCTIONS

```
int main()
{
int n;
printf(«donnez un entier»);
scanf(«%d»,&n);
        f1();
        f2(n);
        f3(n);
return 0;
}
```