



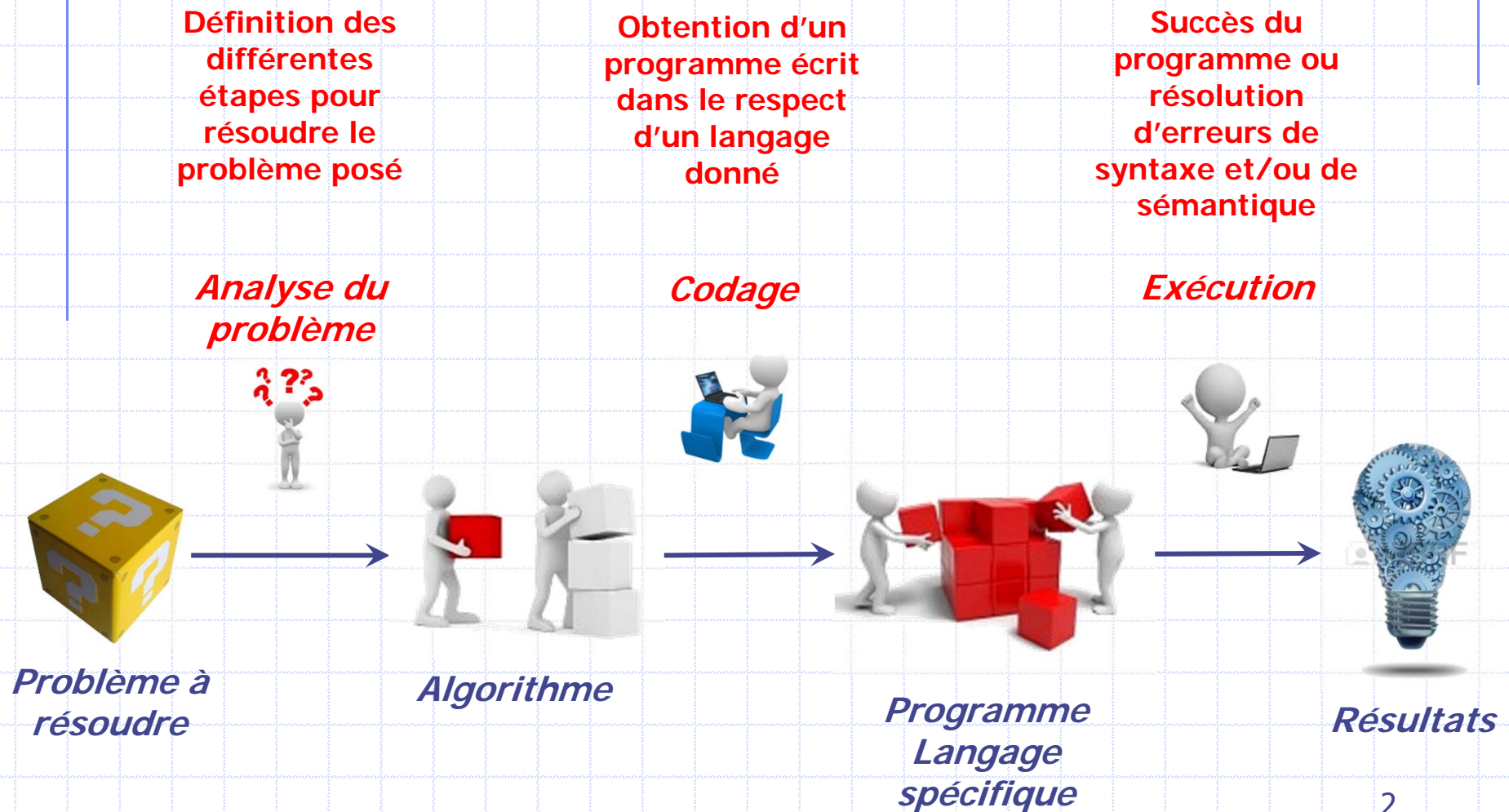
Introduction au C

Cours A112

Christel DARTIGUES-PALLEZ

dartigue@unice.fr

Rappel: Étapes pour résoudre un problème



Généralités sur le C

- ◆ Créé par D. Ritchie et B.W. Kernighan
- ◆ Au début des années 70
- ◆ Objectif
 - Développer un langage qui permet d'obtenir un système d'exploitation
- ◆ Langage normalisé
 - À cause du grand nombre de compilateurs qui ont été définis
- ◆ Langage compilé
 - Utilisation d'un **compilateur C**

Généralités sur le C

◆ Définition d'un compilateur

- Logiciel, programme informatique
- Chargé de traduire
 - ◆ Le code source d'un programme
 - ◆ En un langage machine compréhensible par un ordinateur

Généralités sur le C

◆ Étapes nécessaires à l'exécution d'un programme C

*Peut-être dans
plusieurs fichiers*

Écrire un
programme
source en C

*Sous forme
de texte

Dans un
programme
appelé éditeur*

**Erreurs de
syntaxes**

Compiler le
programme

*Traduction
en langage
machine*

Lier les
fichiers du
programme

**Erreurs dans
l'algorithme**

Exécuter le
programme

Généralités sur le C

◆ Atouts du C

- Un des langages les plus utilisés
 - ◆ Instructions de haut niveau
 - ◆ Génération d'un code très rapide
- Un langage portable
 - ◆ Un programme écrit en C (respectant une norme) est portable sans modification sur n'importe quel système d'exploitation
 - Seule obligation: avoir un compilateur C
- Des programmes compacts et rapides
 - ◆ Le compilateur ne vérifie pas un certain nombre de points
 - Adressage, pointeurs
 - ◆ Nécessite que le programmeur sache ce qu'il fait

Caractéristiques du C

◆ Langage évolué, structuré

- Différents types de données élémentaires (caractères, entiers, réels, pointeurs)
- Possibilité de définir de nouvelles structures de données plus évoluées (tables, vecteurs, structures personnalisées)
- Structures de contrôle classiques (boucles, conditionnelles, sous-programmes)
- Composition et imbrication de structures de contrôle
- Possibilité d'utiliser la récursivité
- Modularité
 - ◆ Décomposition d'une application en plusieurs modules compilés séparément

Difficultés du C

- ◆ Faiblement typé à la compilation
 - Peu de contraintes \Rightarrow risque d'erreurs important
 - Nécessite une grande rigueur de programmation
- ◆ Existence d'opérateurs pouvant rendre un programme illisible et donc difficile à maintenir
- ◆ Peu de protection à l'exécution
 - Pas de contrôle de dépassement des bornes d'un tableau par exemple
- ◆ Pas d'instructions pour la manipulation de chaînes, de tableaux, de listes chaînées
 - \Rightarrow solution ad-hoc (pointeurs)

Structure d'un programme C

- ◆ C'est d'abord un texte écrit en format libre
 - Une seule instruction par ligne,
 - Ou une instruction sur plusieurs lignes
 - Ou plusieurs instructions sur une ligne
- ◆ Distinction entre minuscules et majuscules
- ◆ Conventions
 - Tous les mots réservés et toutes les fonctions prédéfinies sont en minuscules
 - Seuls quelques constantes prédéfinies sont en majuscule (NULL, EOF, O_RDONLY)

Structure d'un programme C

◆ Programme C

- Suite de fonctions (= suite d'actions)

◆ Une fonction particulière est commune à tous les programmes C

- Fonction principale du programme
- `main ()`
- Fait appel à toutes les fonctions qui vont être utilisées dans le programme
- C'est le **point d'entrée** de tout programme
- Indispensable pour exécuter le programme
 - ◆ Il faut toujours en avoir 1
 - ◆ Il ne faut jamais en avoir plusieurs!



Structure générale

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #include "PremierProgramme.h"
5
6
7 void main () {
8
9
10     //Suite d'instructions...
11
12
13     system ("PAUSE") ;
14 }
15
```

Structure d'un programme C

◆ Fichier source d'un programme écrit en C

- Fichier texte
- Extension du fichier : .c
- Exemple
 - ◆ Test.c

◆ Fichier compilé d'un programme écrit en C

- Fichier illisible
- Extension du fichier : .exe
- Exemple
 - ◆ Test.exe

Majuscules/Minuscules

◆ Langage sensible à la casse

■ Exemple

- ◆ **Main** est une fonction **inconnue** du compilateur
- ◆ **main** est une fonction **connue** du compilateur

- Toujours écrire l'en-tête de la fonction principale de la même manière

```
int main ()  
{  
...  
}
```

Un programme simple

◆ Exemple de programme simple

```
#include <stdio.h>

int main ()
{
    printf("hello world !\n\n") ;

    system ("PAUSE") ;

    return 0 ;
}
```



The screenshot shows a Windows command prompt window with the title bar "C:\Enseignement\2015-2016\ModuleProg_Christel\CodeModuleA112\bin\...". The window contains the output of the program: "hello world !" followed by a blank line, and then the text "Appuyez sur une touche pour continuer..." (Press a key to continue...). The window has standard Windows controls (minimize, maximize, close) and a scrollbar on the right.

Un programme simple

◆ Analyse du code de l'exemple précédent

- Fichier .c contenant un main
- Instructions du main
 - ◆ Afficher "Hello World" à l'écran
 - ◆ Attendre que l'utilisateur appuie sur une touche pour fermer la console
 - ◆ Renvoyer la valeur 0

Pour dire que l'exécution du programme s'est bien passée, qu'il n'y a pas eu de problème

```
#include <stdio.h>

int main(void)
{
    printf("Hello World\n");
    system ("pause");
    return 0;
}
```

Un programme simple

- ◆ Utilisation d'une fonction définie dans le langage C : printf
- ◆ Fonctions, constantes définies dans le langage C
 - Placées dans des fichiers auxquels le compilateur se réfère au moment de l'écriture et de la compilation du programme
 - Librairies de fonctions
 - Appelées : fichiers de définition
 - ◆ Extension des fichiers de définition : **.h**

Un programme plus compliqué...

- ◆ Cas des programmes séparés en plusieurs fichiers \Rightarrow **modularité**
- ◆ Indispensable en programmation
- ◆ Intérêt
 - Mieux vaut résoudre 10 problèmes simples qu'un seul problème complexe
 - Fait gagner un temps précieux
 - Une partie peut être réutilisée par plusieurs parties d'un programme ou par plusieurs programmes

Librairies

◆ Pour écrire un programme C

- Nécessité d'utiliser des fonctions standards du C

- ◆ \Rightarrow Il faut indiquer où ces fonctions standards du C sont définies

- ◆ **#include** <fichier_de_définition_standard>

- ◆ Exemple

- La fonction printf est définie dans le fichier stdio.h
(Standard Input Output / Entrées Sorties Standards)
- `#include <stdio.h>`

Librairies

◆ Pour écrire un programme C

- Nécessité de définir les fonctions qu'on définit soi-même dans un fichier de définition
 - ◆ **#include** "fichier_de_définition_définit"
 - ◆ Exemple
 - #include "Test.h"

Librairies

- ◆ Librairie \approx bibliothèque contenant toutes les briques prédéfinies du langage
- ◆ 15 librairies standards dans la norme ANSI
 - `stdio.h`
 - ◆ Ouverture de fichier, gestion du clavier, envoi sur la console
 - `stdlib.h`
 - ◆ Gestion de la mémoire, communication avec les systèmes d'exploitation
 - `string.h`
 - ◆ Gestion des chaînes de caractères
 - `math.h`
 - ◆ Fonctions mathématiques usuelles (sinus, cosinus, tangente, logarithmes, racine carrée, etc.)
 - `time.h`
 - ◆ Permet la manipulation des dates
 - `stdarg.h`, `errno.h`, `float.h`, `stddef.h`, `limits.h`, ...

Les types de base

- ◆ Le langage C fournit un ensemble de types prédéfinis

- ◆ Déclaration d'une variable

- *Type_Variable* nomVariable ;

ou

- *Type_Variable* nomVariable = valeur ;

- Exemples

- int a ; // A a une valeur par défaut fournie par le compilateur

- int b = 1 ;

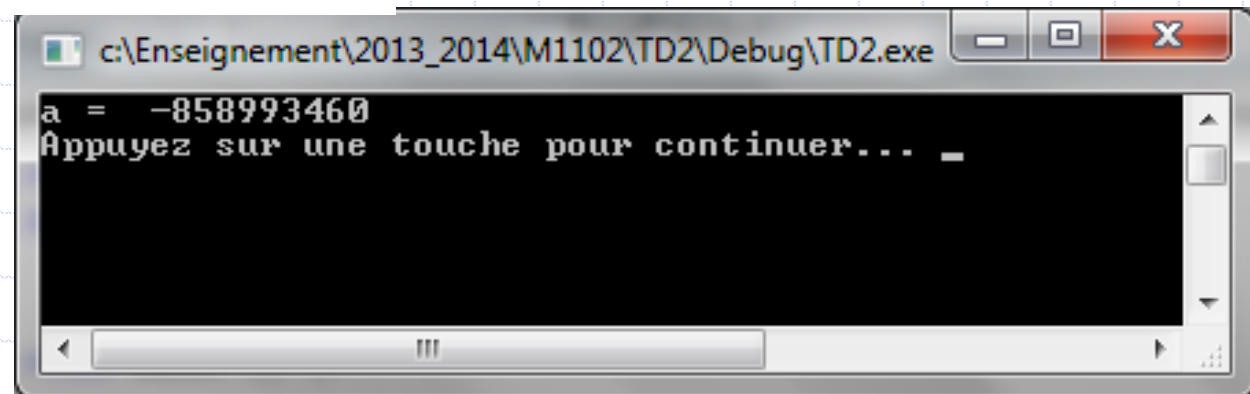
- char c = 'd' ;

- ...

Les types de base

◆ Exemple

```
1 #include <stdio.h>
2 #include "PremierProgramme.h"
3
4
5 void main () {
6
7     int a ;
8     EcrireEntier ("a = ", a) ;
9     system ("PAUSE") ;
10 }
```



The screenshot shows a Windows command prompt window titled "c:\Enseignement\2013_2014\M1102\TD2\Debug\TD2.exe". The window displays the output of the program: "a = -858993460" followed by "Appuyez sur une touche pour continuer...". The window has standard Windows controls (minimize, maximize, close) in the title bar.

Les types de base

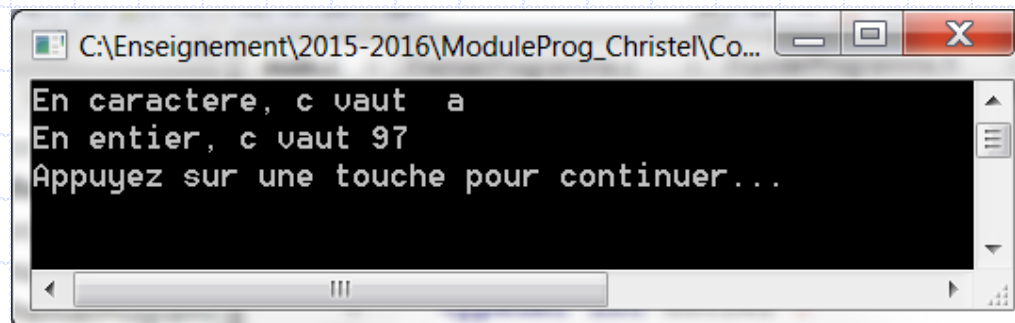
◆ Type caractère

- Mot clé : char
- Peut contenir le code de n'importe quel caractère de l'ensemble des caractères utilisé sur la machine
- Attention
 - ◆ Utilisé comme un entier
 - ◆ Exemple
 - `char c='a' ;`
 - `c = c + 1 ;` est une expression valide \Rightarrow donne le caractère suivant dans le code utilisé par la machine

Les types de base

◆ Exemple

```
int main()  
{  
  
    char c = 'a' ;  
    EcrireCaractere ("En caractere, c vaut ", c) ;  
    EcrireEntier ("En entier, c vaut", c) ;  
  
    system ("PAUSE") ;  
  
    return 0;  
}
```



Les types de base

◆ Type entier

- Mot clé : int
- Différents types d'entiers existent
 - ◆ Prennent plus ou moins de place en mémoire (entre 16 et 32 bits)
 - ◆ long int, int ou short int

◆ Type flottant

- Mots clés : float, double et long double (de la précision la plus faible à la plus forte)

Les types de base

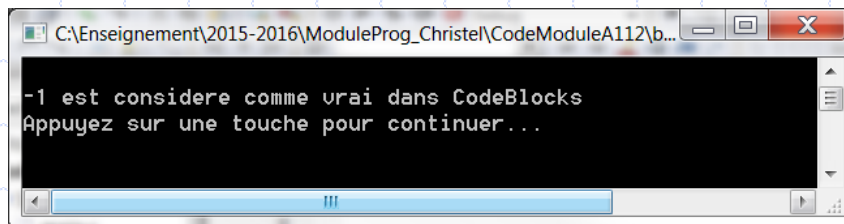
◆ Attention

- Pas de type booléen
 - ◆ Utilisation d'un type numérique
 - 0 = faux
 - Toute autre valeur = vrai

```
int main()
{
    int a = -1 ;
    if (a)
        EcrireMessage("-1 est considere comme vrai dans CodeBlocks") ;
    else
        EcrireMessage("-1 est considere comme faux dans CodeBlocks")

    system ("PAUSE") ;

    return 0;
}
```



Les types de base

◆ Attention

- Pas de type chaîne de caractères
 - ◆ Obligation d'utiliser les pointeurs
 - ◆ Pointeur = accès direct à une zone en mémoire
 - ◆ Rend la programmation **beaucoup plus complexe**

Déclaration des données

◆ Déclaration d'une constante

- Utilisation du mot clé CONST

◆ Exemple

```
const float PI = 3.1415926535897932384626433
```

◆ Remarque

- Par convention, les constantes sont souvent notées en majuscules pour les différencier plus facilement des variables

Les opérateurs

◆ Opérateurs numériques

- $+$, $-$, $*$, $/$ (! à la division **entière**), $\%$ (reste d'une division entière)

◆ Opérateurs de comparaison

- $<$, \leq , $>$, \geq , $==$, $!=$

◆ Opérateurs logiques

- $\&\&$ (et logique), $\|\|$ (ou logique), $!$ (négation logique)

◆ Opérateurs d'affectation

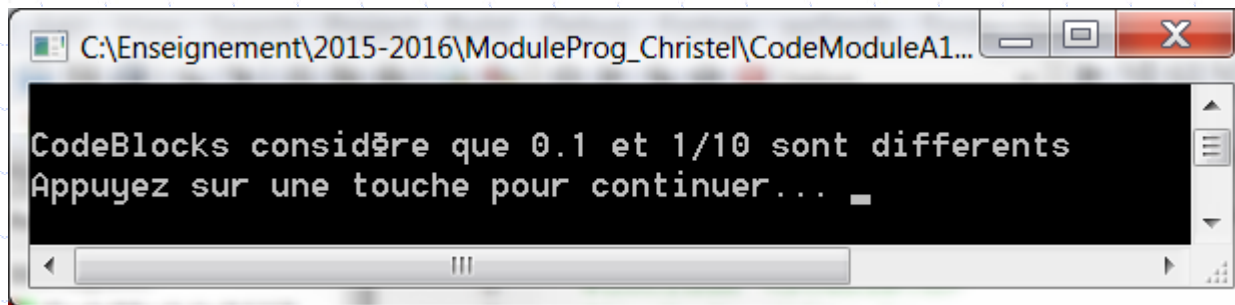
- $=$, $+=$, $-=$, $*=$, $/=$, $\%=$

Les opérateurs

◆ Attention à certains pièges

- Codage des réels...

```
int main()  
{  
  
    int a = 10 ;  
    float x = 0.1, y = 1/a ;  
  
    if (x == y)  
        EcrireMessage("CodeBlocks considère que 0.1 et 1/10 sont égaux") ;  
    else  
        EcrireMessage("CodeBlocks considère que 0.1 et 1/10 sont différents") ;  
  
    system ("PAUSE") ;  
  
    return 0 ;  
}
```



The screenshot shows a Windows-style window titled "C:\Enseignement\2015-2016\ModuleProg_Christel\CodeModuleA1...". The window contains a black console area with white text that reads: "CodeBlocks considère que 0.1 et 1/10 sont différents" followed by "Appuyez sur une touche pour continuer..." and a cursor. The window has standard minimize, maximize, and close buttons in the title bar.

Les opérateurs

- ◆ Attention à certains pièges
 - Division entre 2 entiers...

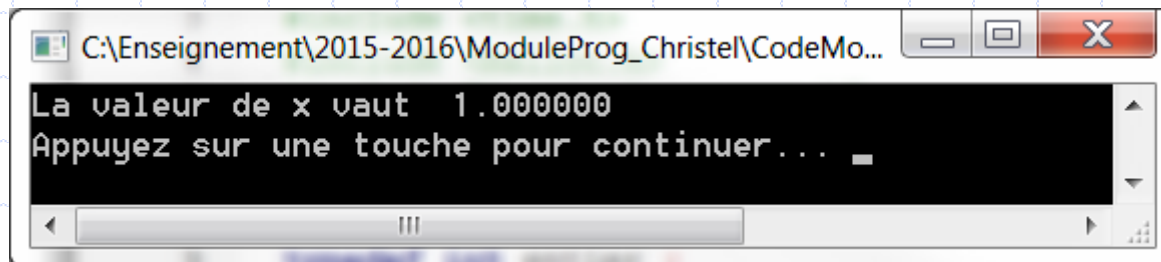
```
int main()
{
    int a = 15, b = 10 ;
    float x ;

    x = a / b ;

    EcrireReel("La valeur de x vaut ", x) ;

    system ("PAUSE") ;

    return 0;
}
```



The screenshot shows a Windows command prompt window with the title bar "C:\Enseignement\2015-2016\ModuleProg_Christel\CodeMo...". The window contains the text "La valeur de x vaut 1.000000" and "Appuyez sur une touche pour continuer..." followed by a cursor. The window has standard Windows controls (minimize, maximize, close) and a scrollbar.

Les opérateurs

◆ Opérateurs d'incrémentation

- `i++`, `++i`, `i--`, `--i`
- Équivalent à : `i=i+1`;
- L'ordre est important !

◆ Opérateur conditionnel ternaire: ?

◆ Syntaxe

`exp0 ? exp1 : exp2`

- Si `exp0` s'évalue à vrai, le résultat de l'expression globale est `exp1`, sinon c'est `exp2`

◆ Exemple

- `max = (a<b)? b : a ;`

Les expressions

◆ Une expression est construite avec des variables et des opérateurs

- Expressions numériques

```
int a=10, b=13, c=5, resultatEntier ;
```

```
float x=4.2, y=7.1, z=9.1, resultatDecimal ;
```

```
resultatEntier = a + b ;
```

```
resultatDecimal = x + y + z ;
```

```
resultatEntier = resultatEntier + a + c ;
```

```
a + b = resultatEntier ;
```

Impossible

```
-a = resultatEntier ;
```

Impossible

Délimiteurs

◆ Délimiteurs

- ;
 - ◆ Termine une déclaration ou instruction
 - ◆ Toute instruction se termine par un **point-virgule**
- ,
 - ◆ Sépare les éléments d'une liste
 - ◆ Dans les paramètres d'une fonction par exemple
- ()
 - ◆ Encadre une liste (éventuellement vide) d'arguments de fonction
- []
 - ◆ Encadre la taille ou un indice de tableau
- { }
 - ◆ Encadre un bloc d'instructions ou des valeurs d'initialisations (tableau, structure)

Les commentaires

◆ Ajout d'un commentaire

/ Ceci est un commentaire
sur plusieurs lignes */*

// Le reste de la ligne est mis en commentaire

- Outil **indispensable** pour programmer

◆ Règles sur les commentaires

- Les commentaires peuvent être placés n'importe où dans le fichier source
- Les commentaires ne peuvent contenir le délimiteur de fin de commentaire (*/)

Les commentaires

◆ Règles sur les commentaires

- Les commentaires ne peuvent être imbriqués
- Les commentaires peuvent être écrits sur plusieurs lignes
- Les commentaires ne peuvent pas couper un mot du programme en deux

◆ Attention

- En cas d'oubli de `*/`, tout le reste du programme sera mis en commentaire

Entrées/Sorties standard

◆ Lire un caractère

- `getchar ()`
- Lit un caractère à partir de l'entrée standard et le renvoie sous forme d'un entier
- Exemple
 - ◆ `char c ;`
 - ◆ `c = getchar () ; /* lit un caractère à partir de l'entrée standard et l'affecte dans c */`

Entrées/Sorties standard

◆ Afficher un caractère

- putchar (c)
- Écrit un caractère passé en argument sur la sortie standard
- Exemple
 - ◆ char c ;
 - ◆ putchar (c) ; /* écrit sur la sortie standard le caractère contenu dans c */

Entrées/Sorties standard

◆ Lire un format spécifique (formaté)

■ scanf()

- ◆ Scrute l'entrée standard et essaie de convertir les données rencontrées dans le format spécifié,
- ◆ S'arrête à la fin des spécificateurs de format, à la fin de fichier ou quand il ne réussit pas à convertir une donnée dans le format spécifié

■ Syntaxe

- ◆ `scanf (const char * format,&arg1,...,&argn)`
- ◆ Format doit contenir des spécificateurs de formats pour les types de données à lire

Entrées/Sorties standard

◆ Lire un format spécifique (formaté)

■ Exemple

```
int n;  
scanf ("%d", &n);
```

- ◆ Permet de lire un entier en notation décimale */

◆ Remarque

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

Entrées/Sorties standard

◆ Afficher un format spécifique

- printf()
- Convertit ses arguments suivant les spécificateurs de format fournis et écrit le résultat sur la sortie standard

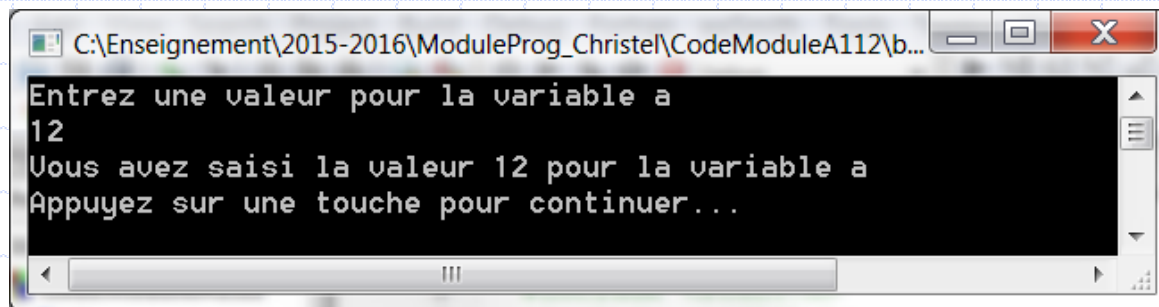
◆ Syntaxe

- printf (const char * format, arg1,...,argn)
- Format doit contenir des spécificateurs de formats correspondant au type de données à écrire

Entrées/Sorties standard

◆ Afficher un format spécifique

```
int main()  
{  
    int a ;  
  
    printf ("Entrez une valeur pour la variable a\n") ;  
    scanf ("%d", &a) ;  
    printf ("Vous avez saisi la valeur %d pour la variable a\n", a) ;  
  
    system ("PAUSE") ;  
  
    return 0 ;  
}
```



```
C:\Enseignement\2015-2016\ModuleProg_Christel\CodeModuleA112\b...  
Entrez une valeur pour la variable a  
12  
Vous avez saisi la valeur 12 pour la variable a  
Appuyez sur une touche pour continuer...
```

Entrées/Sorties standard

◆ Les différents formats

- d, i : entier
- c : caractère
- s : chaîne de caractère (jusqu'à '\0')
- f : double et float
- x, X : hexadécimal
- o : nombre octal

Les erreurs à éviter...

◆ printf

```
int a, b ;  
float x, y ;
```

- Mettre le mauvais format

```
printf ("Vous avez saisi la valeur %d pour la variable x\n", x) ;
```

- Oublier de mettre la variable

```
printf ("Vous avez saisi la valeur %d pour la variable a\n") ;
```

- Oublier de mettre le format

```
printf ("Vous avez saisi la valeur pour la variable x\n", x) ;
```

- Ne pas mettre le bon nombre de variables

```
printf ("Les valeurs saisies sont %d %d %f %f\n", a, b, x) ;
```

- Ne pas mettre les variables dans le bon ordre

```
printf ("Les valeurs saisies sont %d %d %f %f\n", x, a, y, b) ;
```

Les erreurs à éviter...

◆ scanf

```
int a, b ;  
float x, y ;
```

- Mettre le mauvais format
- Oublier de mettre la variable
- Oublier de mettre le & devant la variable
- Ne pas mettre le bon nombre de variables
- Ne pas mettre les variables dans le bon ordre

Entrées/Sorties standard

- ◆ Attention quand on lit des caractères
 - Le caractère de retour à la ligne reste stocké et pose problème lors de la lecture suivante
 - Solution: vider le tampon après chaque lecture
 - ◆ `fflush (stdin) ;`

Les conversions de type

◆ Conversion implicite

- Possibilité de faire naturellement des conversions entre types

- ◆ Exemple de conversion implicite

```
int a=100;
```

```
long b;
```

```
char c = 'v';
```

```
b=a; /* Conversion implicite d'un entier en long */
```

```
a=c; /* Conversion implicite d'un caractère en entier */
```

- ◆ Exemple de conversion qui engendre un message

```
long a=101;
```

```
int b;
```

```
b=a; /* marche, mais déclenche un message d'alerte (Warning)  
dans certains compilateurs*/
```

- Risque de perte d'information

Les conversion de type

◆ Conversion explicite

- Possibilité de forcer une conversion
- Exemple

```
long a=101;
```

```
int b;
```

```
b=(int)a; /* convertit un long en un entier */
```

```
b=(char)a; /* conversion explicite d'un long en char  
puis conversion implicite d'un char en int */
```


Règles

◆ Quelques règles

- L'ordre est important
- Toujours commencer par déclarer les variables
- Toutes les instructions doivent être placées dans une fonction
 - ◆ Soit une fonction qu'on définit
 - ◆ Soit dans le main
- Sauf pour les déclarations de variables qui peuvent se faire en dehors de toute fonction
 - ◆ Ce seront alors des **variables globales** visibles n'importe où dans le programme

Règles

◆ Quelques règles

- Des commentaires, des noms et des décalages...
 - ◆ Les commentaires servent à expliquer les instructions qui apparaissent dans le code
 - Pas assez de commentaires : le code est difficilement compréhensible, il faut simuler le comportement des fonctions pour savoir ce qu'elles font
 - Trop de commentaires : le code est noyé dans les commentaires et la recherche d'un élément précis du code devient fastidieuse
 - Difficulté : trouver le juste milieu entre trop et pas assez de code

Règles

◆ Quelques règles

- Des commentaires des noms et des décalages...
 - ◆ Les noms expressifs pour les variables et les fonctions améliorent la lisibilité des programmes
 - ◆ Les décalages servent à rendre le code plus lisible
 - ◆ Exemple de code illisible

```
/* Que fait ce programme ? */  
#include <stdio.h>  
int main(void)  
{  
    int zkmlpf, geikgh, wdxaj; scanf("%u", &zkmlpf); for (wdxaj=0,  
        geikgh=0; ((wdxaj+=++geikgh), geikgh)<zkmlpf;);  
    printf("%u", wdxaj); return 0;  
}
```

Les instructions

◆ Jusqu'à présent

- Possibilité de lire/écrire des valeurs
- Possibilité de faire des calculs
- Comme une simple calculatrice

◆ Objectif

- Donner à l'ordinateur la possibilité de prendre des décisions
- \Rightarrow Instructions conditionnelles

Les instructions conditionnelles

◆ Les conditionnelles

- Permettent de déterminer le comportement d'un programme en fonction du résultat d'un test
- Outil indispensable en programmation

Les instructions conditionnelles

◆ Les conditionnelles

■ En Algo

Si Condition **alors**
Instructions
FinSi

OU

Si Condition1 **alors**
Instructions1
Sinon Si Condition2 **alors**
Instructions2

...
Sinon
InstructionsN
FinSi

■ En C

if Condition { Instructions }

OU

if Condition { Instructions }
else { Instructions }

OU

if Condition1
{ Instructions }
else if Condition2
{ Instructions }
...
else
{ Instructions }

Les instructions conditionnelles

◆ Quelques éléments d'utilisation

- La condition peut s'exprimer sous la forme d'une comparaison de valeurs
 - ◆ Utilisation des opérateurs de comparaison (<, >, ==, !=, etc.)
 - ◆ La condition peut éventuellement être la combinaison de plusieurs expressions
 - Exemple: `if ((x>2) && (x<10))`
 - Dans ce cas, l'utilisation des parenthèses est indispensable
- Afin de délimiter les blocs, chacun d'eux commence par le caractère "{" et se termine par le caractère "}"
 - ◆ Entre ces deux accolades, on peut placer autant d'instructions qu'on le souhaite, y compris aucune.

Les instructions conditionnelles

◆ Les conditionnelles - Exemples

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

```
int main(void)
{
    int nombre;
    scanf("%d", &nombre);
    if (nombre < 0)
    {
        printf("négatif");
    }
    else
    {
        printf("positif");
    }
    return 0;
}
```


Portée des variables

◆ Jusqu'à maintenant

- Les déclarations consistaient à associer un nom à une valeur dans toute la suite du programme
 - ◆ ie: Une variable était connue dans tout le programme
- Ce n'est pas le cas en général
 - ◆ Un nom peut être associé à une valeur uniquement durant un petit morceau du programme
 - ◆ On parle de portée des variables
- La portée peut être soit locale soit globale

Portée des variables

◆ Exemple de code valide

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

```
int main(void)
```

```
{
```

```
    int nombre;
```

```
    scanf("%d", &nombre);
```

```
    if (nombre < 0)
```

```
    {
```

```
        int valeur_absolue;
```

```
        valeur_absolue = -nombre;
```

```
        printf("Votre nombre est négatif et a pour valeur absolue %d", valeur_absolue);
```

```
    }
```

```
    else
```

```
    {
```

```
        printf("Votre nombre est positif et a pour valeur absolue %d", nombre);
```

```
    }
```

```
    return 0;
```

```
}
```

Portée des variables

◆ Exemple de code invalide

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

```
int main()
```

```
{
```

```
    int nombre;
```

```
    scanf("%d", &nombre);
```

```
    if (nombre < 0)
```

```
    {
```

```
        int valeur_absolue;
```

```
        valeur_absolue = -nombre;
```

```
        printf("Votre nombre est négatif\n");
```

```
    }
```

```
    else
```

```
    {
```

```
        printf("Votre nombre est positif\n");
```

```
    }
```

```
    printf("La valeur absolue de votre nombre est %d\n", valeur_absolue);
```

```
    return 0;
```

```
}
```

La variable
valeur_absolue
n'est pas connue
dans ce bloc
d'instructions



Portée des variables

- ◆ Une déclaration effectuée dans un bloc a une portée limitée à ce bloc
- ◆ Ne pas tout déclarer en variable globale
 - Perte de lisibilité
 - Problème de place mémoire
- ◆ Attention aux noms des variables!

Réduction des blocs

- ◆ Possibilité d'alléger les notations dans les structures conditionnelles quand des blocs sont formés d'une seule instruction

- ◆ Exemple

```
scanf("%d%d", &nombre1, &nombre2);  
if (nombre1 > nombre2)  
{  
    printf("%d\n", nombre1);  
}  
else  
{  
    printf("%d\n", nombre2);  
}
```

Réduction des blocs

◆ Écriture équivalente

```
scanf("%d%d", &nombre1, &nombre2);  
if (nombre1 > nombre2)  
    printf("%d\n", nombre1);  
else  
    printf("%d\n", nombre2);
```

◆ Règle à retenir

- Lorsqu'un bloc ne contient qu'une seule instruction, les accolades peuvent être retirées

Les instructions conditionnelles

◆ Switch

■ Syntaxe

```
switch ( expression )  
Corps
```

■ Exemple

```
j = 0;  
switch (i) {  
    case 3:  
        j++;  
    case 2:  
        j = j + 3;  
    case 1:  
        j = j - 2;  
}
```

Les instructions répétitives

- ◆ Les instructions conditionnelles permettent de faire un grand nombre d'opérations
- ◆ Mais ce n'est pas suffisant
- ◆ Il faut aussi pouvoir répéter des opérations
 - ⇒ Les instructions répétitives
 - ◆ Boucles while (Tant que)
 - ◆ Boucles for (Pour)

Les instructions répétitives

◆ Boucles Tant que

- Condition booléenne pour entrer et sortir de la boucle
- La condition est modifiée à l'intérieur de la boucle
- Nombre **indéterminé** de passages

Tant que condition faire
Instructions ;
FinTantQue

◆ Boucles Pour

- Compteur incrémenté/décroché pour déterminer le nombre de boucles
- Le compteur de boucle est incrémenté/décroché automatiquement
- Nombre **déterminé** de passages

Pour Compteur allant de
ValDeb à ValFin faire
Instructions ;
FinPour

Les instructions répétitives

◆ Boucles tant que

■ Syntaxe

```
while (Condition)
{
    Instructions;
}
```

■ Exemple

```
int i = 0 ;
while (i<4)
{
    printf ("La valeur de i est %d\n",i) ;
    i++ ;
}
```

Les instructions répétitives

◆ Boucles tant que

■ Syntaxe

```
do
{
    Instructions;
} while (Condition)
```

■ Exemple

```
int i = 0 ;
do
{
    printf ("La valeur de i est %d\n",i) ;
    i++ ;
} while (i<4)
```

Les instructions répétitives

◆ Boucles pour

■ Syntaxe

```
for (Expr_Initiale; Expr-Arrêt; Expr_Incrémentation)
{
    Instructions;
}
```

■ Exemple

```
int i ;
for (i=0; i<4; i++)
{
    printf ("La valeur de i est %d\n",i) ;
}
```

Les instructions répétitives

◆ Boucles pour

- Possibilité de définir la variable servant de compteur dans la boucle

- ◆ Exemple

```
for (int i=0; i<4; i++)  
{  
    printf ("La valeur de i est %d\n",i) ;  
}
```

- Possibilité de faire une boucle infinie

- ◆ Exemple

```
for (; ;)  
{  
    printf ("La valeur de i est %d\n",i) ;  
}
```

Les instructions répétitives

◆ Les boucles tant que

■ Erreurs à éviter

- ◆ Une variable utilisée pour la condition n'est pas initialisée
- ◆ La condition test est toujours fausse
 - → On ne rentre jamais dans la boucle : **boucle inutile**
- ◆ La condition est toujours vraie
 - → On ne sort jamais de la boucle : **boucle infinie**

◆ Les boucles pour

■ Erreurs à éviter

- ◆ Le compteur est modifié à l'intérieur de la boucle
→ Interdit !!
- ◆ Utiliser une même variable comme compteur pour plusieurs boucles imbriquées

Programmation propre

◆ Quelques règles

- Faire le plus simple possible
 - ◆ Souvent plus efficace et plus facile à maintenir
- Utiliser des noms de fonctions et de variables explicites
- Se fixer une convention d'écriture et s'y tenir
 - ◆ Ex: Mettre des espaces entre les mots clés, les identifiants et les opérateurs, toujours commencer un nom de variable ou de fonction par une majuscule, etc.
- Écrire une instruction par ligne
- Indenter le code