École de technologie supérieure

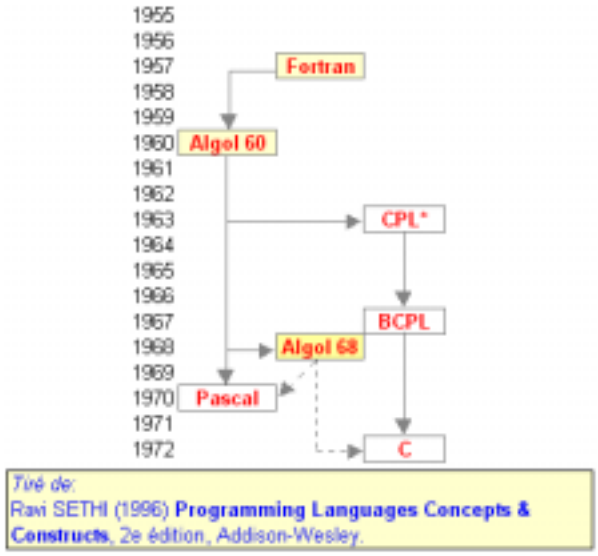
THÈME 1
Vol au-dessus d'un nid de symboles

INF125 Introduction à la programmation
Sylvie Ratté et Hugues Saulnier

Préambule: Langages de programmation et ordinateurs

<<Est-ce qu'un langage de programmation est un outil permettant de rédiger des instructions pour une machine? Est-ce plutôt un moyen de communication entre des programmeurs? Est-ce une façon d'exprimer des solutions de haut niveau? Est-ce une notation formelle pour écrire des algorithmes? Est-ce une façon d'exprimer des relations entre des concepts? Est-ce un outil d'expérimentation? Est-ce une manière de contrôler une machine électronique? La conclusion que j'en tire est qu'un langage de programmation général doit être tout cela afin d'être au service de tous ces types d'utilisateurs. La seule chose qu'un langage ne pourra jamais être c'est "d'être une belle collection de caractéristiques précises et simples".>> B. Stroustrup, le créateur de C++.

Pour un aperçu rapide du concept de langage de programmation, rendez-vous ici...



Historique des influences (langages impératifs)
Pour l'historique des influences globales parmi un sous-ensemble important, rendez-vous ici...

1. Premiers contacts : quelques bons hôtels pour découvrir le C

COMPÉTENCE VISÉE:
Générer des sorties, déclarer des variables, effectuer des calculs, compiler et exécuter des programmes C.

A. Un premier programme C: c'est idiot mais il faut bien commencer quelque part!

Un programme est une COLLECTION de fonctions. Au minimum, il y en a une seule qui se nomme "main". Chaque fonction est en fait un GROUPE D'ÉNONCÉS auquel on a donné un nom. Ce nom constitue le NOM de la fonction.

WELCOME.C	

`main()`
`{`
`  printf("Welcome to the Joy of C!\n");`
`}`
	En-tête de la fonction principale "main"
	Regroupement des énoncés dans un bloc. Le "\n" permet d'effectuer un changement de ligne.

OBSERVEZ: L'en-tête et le corps de la fonction, le regroupement des énoncés avec les accolades, la définition d'une fonction, l'appel d'une fonction, les particularités du "main", la fonction "printf", la notion de paramètres entre parenthèses, l'utilisation des guillemets, le point-virgule qui termine l'énoncé, le "\n" qui représente le changement de ligne.

Compilation et exécution

- Ouverture de TC: éditer, compiler puis exécuter.
- Ouverture de UltraEdit: éditer, appel à gcc puis exécuter.

ESSAYEZ: de compiler et d'exécuter (si possible) le code de ce programme en effectuant un changement de ligne à l'intérieur des guillemets. Qu'arrive-t-il? Il se pourrait que GCC soit plus tolérant que Turbo C. N'est-ce pas?

Insertion de commentaires

WELCOME2.C	

`/*`
` * A program to print a welcoming message.`
` */`
	Les commentaires sont délimités par "/*" et "*/".

`main()`
`{`
`  printf("Welcome to the Joy of C!\n");`
`}`	

OBSERVEZ: l'ouverture /* et la fermeture */ des commentaires.

Avertissements du compilateur

```
Appel à GCC: gcc welcome2.c -Wall
welcome2.c:5: warning: return-type defaults to `int'
welcome2.c: In function `main':
welcome2.c:6: warning: implicit declaration of function `printf'
welcome2.c:7: warning: control reaches end of non-void function
```

Sans l'option "-Wall" (qui signifie "affiche tous les avertissements"), le compilateur GCC n'en produit aucun. Compilé en Turbo C, on obtient un seul message:

```
Warning welcome2.c 7: Function should return a value
```

Introduction des prototypes et des fichiers standards

Le prototype d'une fonction donne une information cruciale au compilateur: le nombre de paramètres et leur type.

Les fichiers "header" (en-tête) présentent l'ensemble des prototypes de fonctions standards. La librarie STDIO.H contient l'ensemble des prototypes des fonctions d'entrée/sortie en C.

WELCOME3.C	

`/*`
` * An improved version of our program to print a welcoming message.`
` *`
` * Revisions:`
` *  1) Added includes for function prototypes and EXIT_SUCCESS.`
` *  2) Added return value to main's header.`
` *  3) Added return statement to main's body.`
` */`
	Commentaires généraux sur le programme.

`#include` `#include`	Inclusion des librairies nécessaires.
`int main()`	La valeur de retour est indiquée. Le "main" en C DOIT TOUJOURS RETOURNER un "int".

`{`
`  printf("Welcome to the Joy of C!\n");`
`  return EXIT_SUCCESS;`
`}`
	Le retour de la valeur s'effectue au moyen de l'énoncé "return". La constante EXIT_SUCCES est définie dans la librairie "stdlib".

OBSERVEZ: l'inclusion de la librairie #include <stdio.h>, le type de retour de la fonction "main", l'énoncé permettant de retourner ce résultat, les mots réservés EXIT_SUCCESS (et EXIT_FAILURE), la différence entre un prototype d'une fonction (= SA DÉCLARATION) et la présence du corps de la fonction (= SA DÉFINITION).

Exemple 1: contenu partiel de "math.h"

```
double cos (double x);
double tan (double x);
double sinh (double x);
double cosh (double x);
double tanh (double x);
double asin (double x);
double acos (double x);
double atan (double x);
double atan2 (double y, double x);
double exp (double x);
```

Exemple 2: contenu partiel de "stdio.h"

```
int fprintf (FILE* filePrintTo, const char* szFormat, ...);
int printf (const char* szFormat, ...);
int sprintf (char* caBuffer, const char* szFormat, ...);
```

Toutes ses fonctions sont un peu spéciales puisque les points de suspension indiquent qu'elles acceptent un nombre variable de paramètres. Les fonctions que nous allons écrire dans le cadre de ce cours auront TOUJOURS un nombre fixe de paramètres.

Vous connaissez déjà la fonction "printf" mais à quoi servent les deux autres? Et bien les deux font exactement la même chose que "printf" à une seule différence près. La première, "fprintf", inscrit les données fournies dans un fichier plutôt qu'à l'écran et la seconde, "sprintf", inscrit ces mêmes données dans une chaîne de caractères plutôt qu'à l'écran. C'est pourquoi, dans ces deux cas, on retrouve un paramètre de plus: "filePrintTo" et "caBuffer". Le reste des arguments est semblable à ceux d'un "printf".Nous utiliserons ces deux fonctions supplémentaires en temps utile. Pour le moment, habituez-vous à lire les prototypes des fonctions appartenant aux librairies standards afin d'identifier: le nombre d'arguments, le type de chacun des arguments et le type de la valeur de retour.

Fonction	Nombre d'arguments	Type de chaque argument	Type de la valeur retournée
fprintf	2 arguments ou plus (...)	FILE*, const char*	int
printf	1 argument ou plus (...)	const char*	int
sprintf	2 arguments ou plus (...)	char*, const char*	int

B. Un second programme

PRICE.C

```
/*
 * Compute the actual cost of buying 23 items at $13.95.  We assume an
 * 18% discount rate and a 7.5% sales tax.
 */
#include < stdio.h >
#include < stdlib.h >

int main()
```

```
{
  int    items;                  /* # of items bought */
  double list_price;             /* list price of item */
  double discount_rate;          /* discount percentage */
  double sales_tax_rate;         /* sales tax percentage */
  double total_price;            /* total price of all items */
  double discount_price;         /* discount price of all items */
  double sales_tax;              /* amount of sales tax */
  double final_cost;             /* cost including sales tax */
```

Déclarations des variables nécessaires avec leur commentaire associé.

```
  items = 23;
  list_price = 13.95;
  discount_rate = .18;
  sales_tax_rate = .075;
```

Initialisation des valeurs de départ.

```
  total_price = items * list_price;
  discount_price = total_price - total_price * discount_rate;
  sales_tax = discount_price * sales_tax_rate;
  final_cost = discount_price + sales_tax;
```

Calcul de chacun des résultats désirés.

```
  printf("List price per item: %f\n", list_price);
  printf("List price of %i items: %f\n", items, total_price);
  printf("Price after %f%% discount: %f\n",
         discount_rate * 100, discount_price);
  printf("Sales tax at %f%%: %f\n",
         sales_tax_rate * 100, sales_tax);
  printf("Total cost: %f\n", final_cost);
```

Affichage des résultats. %f permet d'afficher un "double", "%%" permet d'afficher un %.

```
  return EXIT_SUCCESS;
}
```

OBSERVEZ: la déclaration des variables, les deux types de base (int et double), l'instruction d'assignation, les caractères de formattage dans le "printf", l'emplacement des point-virgules.

Définition de constantes et formattage de sorties élégantes

PRICE2.C

```
#include
#include
```

Les librairies usuelles.

```
#define ITEMS           23
#define LIST_PRICE      13.95
#define DISCOUNT_RATE     .18
#define SALES_TAX_RATE    .075
```

Les constantes.

```
int main()
{
  double discount_price;           /* discounted price of item */
  double total_price;              /* total price of all items */
  double sales_tax;                /* amount of sales tax */
  double final_cost;               /* cost including sales tax */
```

Les variables qui restent.

```
  total_price = ITEMS * LIST_PRICE;
  discount_price = total_price - total_price * DISCOUNT_RATE;
  sales_tax = discount_price * SALES_TAX_RATE;
  final_cost = discount_price + sales_tax;
```

Utilisation des constantes dans les calculs.

```
  printf("List price per item:\t\t%7.2f\n", LIST_PRICE);
  printf("List price of %i items:\t\t%7.2f\n", ITEMS, total_price);
  printf("Price after %.1f%% discount:\t%7.2f\n",
         DISCOUNT_RATE * 100, discount_price);
  printf("Sales tax at %.1f%%:\t\t%7.2f\n",
         SALES_TAX_RATE * 100, sales_tax);
  printf("Total cost:\t\t\t%7.2f\n", final_cost);

  return EXIT_SUCCESS;
}
```

Affichage plus joli: %7.2f formattage sur 7 espaces avec deux chiffres après le point d'un double. %.1 formattage avec 1 chiffre après le point. \t permet d'effectuer une tabulation.

OBSERVEZ: notre première directive au "prépocesseur" (#define), l'absence du point-virgule après la directive, la séquence de base pour formatter l'affichage des nombres réels "%7.2f".

Indentation

MESSY.C

```
#include
#include
int main() {double dp;double tp;double st;double fc;
tp=23*13.95;dp=tp-tp*.18;st=dp*.075;fc=dp+st;
printf("List price per item:\t\t%7.2f\n",13.95);
printf("List price of %i items:\t\t%7.2f\n",23,tp);
printf("Price after %.1f%% discount:\t%7.2f\n",.18*100,dp);
printf("Sales tax at %.1f%%:\t\t%7.2f\n",.075*100,st);
printf("Total cost:\t\t\t%7.2f\n",fc);return EXIT_SUCCESS;}
```

ce programme effectue le même traitement que PRICE.C... sans commentaire!

Vaut mieux prendre l'habitude d'indenter les énoncés à l'intérieur d'un BLOC et de placer (pour l'instant du moins) un SEUL énoncé par ligne. Vous ne trouvez pas?

2. Survol en douceur du C: Visitez tous les méandres du C en 13 semaines!

COMPÉTENCE VISÉE:
Écrire des petits boucles et des décisions simples et lire des données.

Dans les programmes qui suivent, seules les lignes pertinentes à la discussion sont illustrées. Pour visualiser le programme complet, cliquez sur son hyperlien.

A. Notre première boucle

INTRATP1.C

```
#define PRINCIPAL  5000.00              /* start with $5000 */
#define INTRATE       0.06              /* interest rate of 6% */
#define PERIOD        7                 /* over 7-year period */
```

Les trois constantes de départ.

```
int main()
{
  printf("Interest Rate:       %7.2f%%\n", INTRATE * 100);
  printf("Starting Balance:  $ %7.2f\n", PRINCIPAL);
  printf("Period:              %7i years\n\n", PERIOD);
  printf("Year     Balance\n");

  return EXIT_SUCCESS;                  /* assume program worked! */
}
```

Affichage simple.

OBSERVEZ: l'utilisation de <stdio.h> pour obtenir la déclaration du "printf" et de <stdlib.h> pour obtenir celle de EXIT_SUCCESS et

EXIT_FAILURE.

Énoncé "while"

SYNTAXE:

```
while (expression)
{
   énoncés à répéter
}
```

NOTE: S'il n'y a qu'un seul énoncé à répéter, on peut ommetre les accolades pour marquer le bloc.

FONCTIONNEMENT: <expression> est évaluée. Si l'expression est vraie, les énoncés sont exécutées puis l'expression est évaluée de nouveau et ainsi de suite. Dès que le résultat de l'évaluation de l'expression est faux, le processus se rend à l'énoncé suivant le "while". Pas question de remonter!

Les OPÉRATEURS RELATIONNELS ("<", ">", "==", ">=", "<=" et "!=") permettent de construire des expressions dont l'évaluation donne VRAI ou FAUX.

INTRATP2.C

`int main()`
`{`
`  int  year;`
	Déclaration de la variable compteur

`  printf("Interest Rate:  %7.2f%%\n", INTRATE * 100);`
`  printf("Starting Balance:  $ %7.2f\n", PRINCIPAL);`
`  printf("Period:  %7i years\n\n", PERIOD);`
`  printf("Year  Balance\n");`
	Affichage de départ.

`  year = 1;`	Initialisation de la variable compteur.
`  while (year <= PERIOD)` `  {`	Début de la boucle.
`    printf("%4i\n", year);`	énoncé à répéter
`    year = year + 1;` `  }`	incrément de la variable compteur
`  return EXIT_SUCCESS;}`	Succès

SCHÉMA D'UNE BOUCLE QUI COMPTE:

```
int variable-compteur;
  ...
variable-compteur = valeur-initiale;
while (variable-compteur <= valeur-finale)
{
   énoncés à répéter
   mise à jour de la variable-compteur
}
```

INTRATE.C

`int main()`
`{`
`  double balance;  /* balance at year's end */`
`  int  year;  /* year of period */`
	Déclarations usuelles

`  printf("Interest Rate:  %7.2f%%\n", INTRATE * 100);`
`  printf("Starting Balance:  $ %7.2f\n", PRINCIPAL);`
`  printf("Period:  %7i years\n\n", PERIOD);`
`  printf("Year  Balance\n");`
	Affichage de départ.

`  balance = PRINCIPAL;` `  year = 1;`	Initialisations
`  while (year <= PERIOD)` `  {`	Début des itérations
`    balance = balance + balance * INTRATE;` `    printf("%4i  $ %7.2f\n", year, balance);`	Mise à jour du solde et affichage.
`    year = year + 1;` `  }`	Mise à jour de la variable compteur
`  return EXIT_SUCCESS;  /* assume program worked! */` `}`	Succès.

OBSERVEZ: la déclaration de la variable compteur, sa mise à jour dans le BLOC de la boucle, son initialisation AVANT l'entrée dans la boucle, l'instruction d'assignation permettant sa mise à jour.

Quoi faire avec les erreurs de compilation?

COMPILEZ OOPS.C et OOPS2.C et examinez leurs différences.

Voici les erreurs obtenues avec GCC:

```
OOPS.C: In function `int main()':
OOPS.C:16: parse error before `('
OOPS.C:25: parse error before `='
OOPS.C:28: `EXIT_SUCCESS' undeclared (first use this function)
OOPS.C:28: (Each undeclared identifier is reported only once
OOPS.C:28: for each function it appears in.)
OOPS.C:25: warning: label `year' defined but not used
OOPS.C:29: warning: control reaches end of non-void function `main()'
```

B. Deux versions plus intéressantes

Énoncé "for"

Voici une variante du programme INTRATE. Elle utilise un autre type de boucle, la boucle FOR. Cette version est légèrement plus compacte.

INTRATE2.C

```
int main()
{
  double balance;
  int    year;
  printf("Interest Rate:     %7.2f%%\n", INTRATE * 100);
  printf("Starting Balance: $%7.2f\n\n", PRINCIPAL);
  printf("Period:            %7i years\n\n", PERIOD);
  printf("Year     Balance\n");
```

```
  balance = PRINCIPAL;
  for (year = 1; year <= PERIOD; year = year + 1)
  {
    balance = balance + balance * INTRATE;
    printf("%4i   $ %7.2f\n", year, balance);
  }
```

le while est remplacé par un "for".

```
  return EXIT_SUCCESS;}
```

SYNTAXE:

```
for (initialisation ; condition ; action)
{
  énoncés à répéter
}
```

NOTES:<initialisation>, <condition> et <action> peuvent être N'IMPORTE QUELLE EXPRESSION EN C. S'il n'y a qu'un seul énoncé à répéter, on peut ommetre les accolades pour marquer le bloc.

FONCTIONNEMENT: D'abord l'expression <initialisation> est exécutée, puis la <condition> est évaluée. Si cette dernière s'évalue à FAUX, le processus se rend à l'énoncé suivant le "for". Si, par contre, cette condition s'évalue à VRAI les <énoncés à répéter> sont exécutés, puis <action> est exécuté puis la condition est évaluée de nouveau et ainsi de suite: <énoncés à répéter>, <action>, <condition>, <énoncés à répéter>, <action>, <condition>, etc.

On utilise habituellement un "for" pour implémenter les boucles "qui comptent"...

Lecture de valeurs avec l'énoncé "scanf"

La fonction "scanf" se trouve dans la librarie <stdio.h>. Voyons le contenu de cette librairie à nouveau:

```
int fscanf (FILE* fileReadFrom, const char* szFormat, ...);
int scanf (const char* szFormat, ...);
int sscanf (const char* szReadFrom, const char* szFormat, ...);
```

La fonction "scanf" exige deux paramètres: une chaîne de caractères s'apparentant à celle de "printf" et la liste des emplacements (les variables) où les valeurs lues seront placées. Pour lire des entiers, on utilise les séquences "%i" ou "%d" MAIS, pour lire des "double", on utilise "%lf" (une abbréviation de "long float"). La fonction "fscanf", vous l'aurez deviné, nous servira à effectuer nos lectures dans un fichier tandis que la fonction "sscanf" nous permettra d'effectuer nos lectures à l'intérieur d'une chaîne de caractères.

INTRATE3.C

```
int main()
{
  double intrate;
  double balance;
  int    year;
  int    period;
```

Les constantes sont désormais des variables..

```
  printf("Enter interest rate, principal, and period: ");
  scanf("%lf%lf%i", &intrate, &balance, &period);
```

dont les valeurs sont fournies par l'utilisateur.

```
  printf("Interest Rate:      %7.2f%%\n", intrate * 100);
  printf("Starting Balance: $ %7.2f\n\n", balance);
  printf("Period:             %7i years\n\n", period);
  printf("Year     Balance\n");
```

affichage de départ.

```
  for (year = 1; year <= period; year = year + 1)
  {
    balance = balance + balance * intrate;
    printf("%4i   $ %7.2f\n", year, balance);
  }
```

calcul et affichage des intérêts.

```
  return EXIT_SUCCESS;
}
```

OBSERVEZ: la construction du "scanf", la différence des séquences pour lire des "double" avec "scanf" et les afficher avec "printf", la présence du symbole "&" OBLIGATOIRE devant chaque variable qui recevra une valeur. On nomme cet OPÉRATEUR: l'opérateur D'ADRESSAGE.

Lecture répétitive de valeurs

En fait, si vous êtes curieux, vous constaterez que le prototype de la fonction "scanf" suggère que cette fonction retourne une valeur. Pour vous en convaincre, ouvrez le fichier "stdio.h" et examinez le prototype de la fonction "scanf":

```
int scanf (const char* szFormat, ...);
```

La fonction "scanf" retourne le nombre de valeurs effectivement lues avec succès. On peut donc placer le résultat qu'elle retourne dans une variable du bon type (c'est un "int") afin de réutiliser cette information plus loin dans le programme.

INTRATE4.C

```
int main()
{
  double intrate;                    /* interest rate */
  double balance;                    /* balance at year's end */
  int    year;                       /* year of period */
  int    period;                     /* length of period */
  int    n;                          /* number of values read */
```

Une variable de plus. "n" contient le nombre de valeurs correctement lues par "scanf".

```
  printf("Enter interest rate, principal, and period: ");
  n = scanf("%lf%lf%i", &intrate, &balance, &period);
```

Lecture avec récupération de la valeur retournée par "scanf".

```
  while (n == 3)
  {
```

Lecture de manière répétitive. Dès que "n2" sera différent de 3, on sortira de la boucle.

```
    printf("Interest Rate:      %7.2f%%\n", intrate * 100);
    printf("Starting Balance: $ %7.2f\n", balance);
    printf("Period:             %7i years\n\n", period);
    printf("Year     Balance\n");
    for (year = 1; year <= period; year = year + 1)
    {
      balance = balance + balance * intrate;
      printf("%4i   $ %7.2f\n", year, balance);
    }
```

Caculs et affichages pour les données lues précédemment...

```
    printf("\nEnter interest rate, principal, and period: ");
    n = scanf("%lf%lf%i", &intrate, &balance, &period);
  }
```

Lecture de trois nouvelles données...

```
  return EXIT_SUCCESS;
}
```

OBSERVEZ: l'utilisation de la variable "n" pour détecter le nombre de valeurs lues, la construction des deux boucles imbriquées notamment la plus externe. Dans quelles conditions s'arrête le programme?

La boucle externe est en fait une "boucle de lecture" typique.

SCHÉMA D'UNE BOUCLE DE LECTURE

```
affichage d'un message d'invite
n = scanf(...);
```

```
while (n == nombre-attendu-de-valeurs)
{
   traiter les valeurs qui viennent tout juste d'être lues
   affichage du message d'invite
   n = scanf(...);
}
```

C. Intéractivité

Autre manière de faire des lectures répétitives

Le programme suivant effectue la lecture d'une série de notes tout en comptant le nombre de notes fournies.

GRADEP1.C	

`int main()`
`{`
`  int next_score;  /* current input value */`
`  int n;  /* input values read with last scanf */`
`  int score_count;  /* count of scores read */`
	Nos variables.

`  score_count = 0;` `  printf("Score? ");`	initialisation de notre compteur de notes et affichage du message d'invite (c'est plus civilisé!)...
`  n = scanf("%i", &next_score);`	Lecture de la première note.
`  while (n == 1)` `  {`	Répétition jusqu'à ce que l'utilisateur saisisse une valeur non valide ou encore un CTRL-Z (sur PC).
`    score_count = score_count + 1;`	on incrémente notre compteur de notes
`    printf("%i\n", next_score);` `    printf("Score? ");`	on affiche la dernière note lue puis... le message d'invite...
`    n = scanf("%i", &next_score);` `  }`	on lit une nouvelle note... l'ancienne valeur de la variable "next_score" est remplacée.
`  printf("\n%i scores entered.\n", score_count);`	Lors de la sortie de la boucle, on affiche le nombre de notes lues...
`  return EXIT_SUCCESS;` `}`	c'est un succès!

OBSERVEZ: la construction de la boucle de lecture où l'on test n==1 plutôt que n==3.

Prendre des décisions avec l'énoncé "if"

Le programme peut produire deux résultats pour chaque notes. Pour ce faire, il utilise l'énoncé "if".

GRADEP2.C	

`int main()`
`{`
`  int next_score;  /* current input value */`
`  int n;  /* input values read with last scanf */`
`  int score_count;  /* count of scores read */`	

`  int pass_count;  /* count of passing scores */` `  int fail_count;  /* count of failing scores */`	Deux variables supplémentaires pour compter le nombre de succès et le nombre d'échecs.

`  score_count = 0;`
`  pass_count = 0;`
`  fail_count = 0;`
 	
`  printf("Score? ");`
`  n = scanf("%i", &next_score);`
	initialisations préliminaires des compteurs et lecture de la première note.

`  while (n == 1)`
`  {`
`  score_count = score_count + 1;`
	Incrément de la variable compteur pour toutes les notes

`  if (next_score >= PASSING_SCORE)`
`  {`
`  printf("%i - Passes\n", next_score);`
`  pass_count = pass_count + 1;`
`  }`
	Incrément (si cela est pertinent) de la variable compteur pour les succès

`  else`
`  {`
`  printf("%i - Fails\n", next_score);`
`  fail_count = fail_count + 1;`
`  }`
	Incrément (si cela est pertinent) de la variable compteur pour les échecs

`  printf("Score? ");`
`  n = scanf("%i", &next_score);`
`  }`
	Lecture de la prochaine note

`  printf("\n%i scores entered, %i pass, %i fail.\n",`
`  score_count, pass_count, fail_count);`
 	
`  return EXIT_SUCCESS;`
`}`
	Affichage des résultats demandés.

SYNTAXE:

```
if (expression)
{
   énoncés à faire lorsque vrai
}
else
{
   énoncés à faire lorsque faux
}
```

FONCTIONNEMENT: L'expression est évaluée. Si le résultat est VRAI, les énoncés du premier bloc sont exécutés puis le processus se rend à l'énoncé qui SUIT le "if". Si le résultat est FAUX, les énoncés du second bloc sont exécutés puis le processus se rend à l'énoncé qui SUIT le "if".

Version plus complète

Nous avons appris comment bâtir une boucle qui compte. Voici une version qui permet d'obtenir, presque gratuitement, la moyenne de toutes les notes, le nombre d'échecs et le nombre de succès.

GRADE.C

```
int main()
{
  int next_score;
  int n;
  int score_count;
  int pass_count;
  int fail_count;
```

```
  int avg_score;          /* average score */
  int total_score;        /* total score */
```

Deux variables de plus... "total_score" servira à additionner toutes les notes.

```
  score_count = 0;  pass_count = 0;  fail_count = 0;  total_score = 0;

  printf("Score? ");
  n = scanf("%i", &next_score);
```

initialisations... Le C possède une syntaxe libre. On peut donc écrire plusieurs énoncés sur une ligne. C'est souvent le cas lors des initialisations.

```
  while (n == 1)
  {
```

```
    score_count = score_count + 1;
    total_score = total_score + next_score;
```

On incrémente la variable compteur pour toutes les notes et on cumule dans "total_score" la note que l'on vient de lire.

```
    if (next_score >= PASSING_SCORE)
    {
      printf("%i - Passes\n", next_score);
      pass_count = pass_count + 1;
    }
    else
    {
      printf("%i - Fails\n", next_score);
      fail_count = fail_count + 1;
    }
```

```
    printf("Score? ");
    n = scanf("%i", &next_score);
  }
```

```
  if (score_count == 0)
    avg_score = 0;
  else
    avg_score = total_score / score_count;
```

on calcule la moyenne mais ATTENTION, il est possible qu'aucune note n'ait été rentrée.

```
  printf("\n%i scores entered, %i pass, %i fail.\n",
         score_count, pass_count, fail_count);
  printf("Average: %i\n", avg_score);

  return EXIT_SUCCESS;
}
```

OBSERVEZ: L'initialisation et la mise à jour des variables "score_count", "pass_count", "fail_count" et "total_score".

Quoi faire avec les erreurs de l'utilisateur

Dans les programmes précédents, dès que l'utilisateur fait une erreur le programme s'arrête normalement et l'erreur n'est même pas mentionnée. En fait, seule le CTRL-Z (sur PC) devrait permettre de sortir élégamment de notre programme.

INTRATE5.C

```
int main()
{
  double intrate;
  double balance;
  int    year;
  int    period;
  int    n;
  printf("Enter interest rate, principal, and period: ");
  n = scanf("%lf%lf%i", &intrate, &balance, &period);

  while (n == 3)
  {
     ... comme INTRATE4.C ...
  }
```

Le programme est semblable à la version 4.

```
  if (n != EOF)
    printf("Warning: Input reading terminated by error.\n");

  return EXIT_SUCCESS;
}
```

Si l'utilisateur utilise autre chose que CTRL-Z pour sortir, on affichera un message... on pourrait également retourner un EXIT_FAILURE dans ce cas...

OBSERVEZ: l'utilisation de EOF pour minimalement avertir qu'une erreur éventuelle s'est produite, l'énoncé "if" sans bloc "else" lorsque rien n'est prévu lorsque la condition est fausse.

"Débugger" ou déverminer

Dans un environnement minimal de développement, el celui offert par GCC, on doit avoir recours à des "printf" supplémentaires pour traquer l'ennemi ultime: le programme se compile mais ne produit pas les bons résultats (le vrai "bug"). Dans un environnement intégré, tel celui offert par TC, on peut observer pas-à-pas la valeur de chaque variable.

BADGRADE.C et FIXGRADE.C

D. Le l'utilisation professionnel de l'opérateur d'assignation avec "scanf"

Le programme suivant illustre une autre manière de lire des données de façon répétitive.

INTRATE6.C

```
int main()
{
  double intrate;                   /* interest rate */
  double balance;                   /* balance at year's end */
  int    year;                      /* year of period */
  int    period;                    /* length of period */
  int    n;                         /* number of values read */

  printf("Enter interest rate, principal, and period: ");
```

Préalables usuels...

```
  while ((n = scanf("%lf%lf%i", &intrate, &balance, &period)) == 3)
  {
```

On effectue le test sur le résultat de l'assignation... énoncé concis typique du C...

```
    printf("Interest Rate:      %7.2f%%\n", intrate * 100);
    printf("Starting Balance: $ %7.2f\n", balance);
    printf("Period:             %7i years\n\n", period);
    printf("Year     Balance\n");

    for (year = 1; year <= period; year = year + 1)
    {
      balance = balance + balance * intrate;
      printf("%4i   $ %7.2f\n", year, balance);
    }
```

La boucle pour l'affichage du tableau...

```
    printf("Enter interest rate, principal, and period: ");
  }
```

Pas nécessaire d'effectuer un "scanf" ici puisque ce dernier est effectué lors de l'évaluation de la condition.

```
  if (n != EOF)
    printf("Warning: Input reading terminated by error.\n");

  return EXIT_SUCCESS;                    /* assume always succeeds */
}
```

la fin usuelle

SCHÉMA D'UNE BOUCLE DE LECTURE PLUS PROFESSIONNELLE

```
afficher le message d'invite
while ((n = scanf(...)) == nombre-attendu-de-valeurs)
{
   traiter les valeurs qui viennent tout juste d'être lues
   affichage du message d'invite
}
```

Il est évidemment impossible d'éviter d'afficher le message d'invite deux fois. Attention cependant à ne pas confondre l'opérateur d'égalité (==) avec l'opérateur d'assignation (=).

E. Appel à votre intuition toute neuve!

Terminons ce thème par un petit exercice intuitif. Vous savez maintenant que l'on peut déclarer une variable "toto" de type "double" comme suit:

```
double toto;
```

Si nous vous disons maintenant qu'en ajoutant un nombre entier N entre deux crochets à cette déclaration vous obtenez N variables de type "double" toutes accessibles par le même nom:

```
double toto[100];
```

Comment pensez-vous que nous pourrons accéder à chacune des 100 variables individuelles? On fera, très naturellement ceci: toto[0] sera la première, toto[1] sera la seconde, ..., toto[99] sera la dernière. Vous voici devant le concept de tableau. C'est un concept simple et nous l'utiliserons beaucoup dans ce cours. Gardez-le en tête. Nous y reviendrons très bientôt.