Tests et itérations

Programmes séquentiels ne résolvent pas tous les problèmes

exemple simple : calcul des racines d'un polynôme de d° 2 dans $\mathbb{R}$

Algorithme pour la résolution de $a.x^2+b.x+c=0$

calculer $\Delta=b^2-4ac$

si $\Delta < 0$: pas de solution

si $\Delta \geq 0$: solutions : $x_1 = \dfrac{-b+\sqrt{\Delta}}{2a}$ et $x_2 = \dfrac{-b-\sqrt{\Delta}}{2a}$

Test sur la valeur de Δ

Tests et itérations

Programmes séquentiels ne résolvent pas tous les problèmes

autre exemple simple : saisie en contrôlant la valeur entrée

une valeur à saisir doit absolument être supérieure à une borne *b_inf*.

algorithme :

saisir la valeur

tant que la valeur est < *b_inf*, recommencer la saisie.

Réaliser un test : conditions logiques

`if` sert à réaliser un test selon une **condition logique**

cette condition sera évaluée et aura une valeur de vérité : ***vraie*** ou ***fausse***. C'est selon cette valeur, vraie ou fausse, que l'on peut choisir de faire certaines instructions ou non.

Utilisation d'opérateurs de comparaison :

2 valeurs sont elles-égales ? $\Rightarrow$ réponse de type vrai ou faux

1 valeur est-elle supérieure (au sens strict ou au sens large) à une autre ? $\Rightarrow$ réponse de type vrai ou faux

Réaliser un test : conditions logiques

Opérateurs de comparaison :

Syntaxe	**emploi**	**vrai si**
==	val1 == val2	val1 est égal à val2
!=	val1 != val2	val1 est différent de val2
<	val1 < val2	val1 strictement inférieur à val2
<=	val1 <= val2	val1 inférieur ou égal à val2
>	val1 > val2	val1 strictement supérieur à val2
>=	val1 >= val2	val1 supérieur ou égal à val2

Attention à l'opérateur d'égalité ==, il ne faut pas le confondre avec l'opérateur d'affectation =

Opérateurs logiques

En fait, une condition est une expression qui donne une valeur de type vrai ou faux.

Il existe des opérateurs logiques permettant de construire des expressions logiques (de même que l'on construit des expressions mathématiques avec des opérateurs mathématiques).

Arithmétique des valeurs logiques :

opérateurs ET, OU et NON

ET et OU : composent 2 valeurs logiques

NON : s'applique à une valeur logique

Opérateurs logiques

Tables de vérité des opérateurs logiques. Soient A et B deux conditions, qui peuvent prendre les valeurs VRAI ou FAUX.

- L'expression A ET B a pour valeur VRAI ssi A et B ont pour valeur VRAI en même temps, sinon A ET B vaut FAUX.
- L'expression A OU B a pour valeur VRAI si A est VRAI ou si B est VRAI.
- L'expression NON A a pour valeur VRAI si A a pour valeur FAUX, et vaut FAUX si A a pour valeur VRAI.

Traduction en C :

ET : &&

OU : || (2 barres verticales, AltGr 6)

NON : !

Opérateurs logiques

Remarque sur le parenthésage des expressions logiques :

&& est prioritaire sur ||.

Il faut systématiquement utiliser des parenthèses (les plus prioritaires) pour construire les expressions.

Ne pas avoir à connaître par cœur les priorités des opérateurs;

Rendre les expressions lisibles.

Mettre des parenthèses autour de chaque condition.

Syntaxe de l'instruction **if**

Réaliser un test simple : si une condition est vraie alors faire une instruction.

```
if (condition) instruction; /* cas d'une instruction simple */
```

ou

```
if (condition) { instructions } /* cas d'un bloc d'instruction */
```

si la condition est fausse, l'instruction (ou le bloc) ne sera pas effectué.

l'instruction **if** dans un programme

Programme exemple : test d'une valeur

```
#include <stdio.h>

void main()
{
	unsigned char age;

	printf("SNCF Bijour\n");
	printf("Entrez votre age:");
	scanf("%c",&age);

	if (age > 26)
	{
		printf("Vous n'avez pas de réduction\n");
	}
}
```

l'instruction **if** dans un programme

```
#include <stdio.h>

void main()
{
	unsigned int mystere;
	unsigned int solution;

	solution = 1806;
	printf("quelle est la date de la bataille d'Iéna ?:");
	scanf("%d",&mystere);

	if (mystere != solution)
	{
		printf("Révisez vos classiques !\n");
	}
	if (mystere == solution)
	{
		printf("Bravo !\n");
	}
}
```

l'instruction **if** dans un programme

```
#include <stdio.h>

void main()
{
	short int uneValeur;

	printf("entrez une valeur comprise entre 0 et 10:");
	scanf("%d",&uneValeur);

	if ((uneValeur <0) || (uneValeur > 10))
	{
		printf("mais mous savez lire ?\n");
	}
	if ((uneValeur>3) && (uneValeur <7))
	{
		printf("en plein milieu de l'intervalle !\n");
	}
}
```

L'alternative : **if … else**

Sur l'exemple avec la date mystère : 2 tests, dont 1 inutile !

Les conditions sont exclusives : si l'une est vraie, alors l'autre est forcément fausse !

Reformuler le programme précédent :

si (la date est incorrecte) **alors** afficher message d'erreur

sinon afficher le message de réussite

avec **if** : instruction ou bloc d'instruction exécuté si et seulement si la condition est vraie. Avec **else** : instruction ou bloc d'instruction exécuté si et seulement si la condition testée par **if** est fausse.

L'alternative : **if … else**

Syntaxe :

```
if (condition)
{
      instructions; /* si la condition est vraie */
}
else
{
      autres instructions; /* si elle est fausse */
}
```

L'alternative dans un programme

```
#include <stdio.h>

void main()
{
	unsigned int mystere;
	unsigned int solution;

	solution = 1806;
	printf("quelle est la date de la bataille d'Iéna ?:");
	scanf("%d",&mystere);

	if (mystere != solution)
	{
		printf("Révisez vos classiques !\n");
	}
	else   /* si la condition est fausse */
	{
		printf("Bravo !\n");
	}
}
```

Conditions et alternatives

Avec une condition de type

val1 == val2

val1 != val2

val1 < val2

val1 <= val2

de même pour > et >=

les instructions du bloc lié au **else** seront faites si

val1 différent de val2

val1 est égal à val2

val1 supérieur ou égal à val2

val1 strictement supérieur à val2

Application

Écrire le programme de résolution d'une équation de d° 2 dans R en utilisant l'alternative : on ne prend pas en compte le cas particulier $\Delta=0$.

```
#include <stdio.h>
void main()
{
	float coeff_a, coeff_b, coeff_c;
	float delta, sol_1, sol_2;

	/* saisie des coefficients, calcul de delta */
	printf("Entrer les 3 coefficients à la suite:");
	scanf("%f",&coeff_a);
	scanf("%f",&coeff_b);
	scanf("%f",&coeff_c);

	delta = (coeff_b*coeff_b)-(4.0*coeff_a*coeff_c);
}
```

Application (suite)

```
        if (delta < 0)
        {
                printf("pas de solutions réelles\n");
        }
        else   /* donc si delta >= 0 */
        {
                sol_1=(-coeff_b-sqrt(delta))/(2.*coeff_a);
                sol_2=(-coeff_b+sqrt(delta))/(2.*coeff_a);
                printf("solutions : %f et %f\n",sol_1,sol_2);
        }

        printf("Au revoir \n");
}
```

Alternatives multiples

if…else permet de traiter deux conditions exclusives l'une de l'autre. Mais pour 3 conditions exclusives ? Ou plus ?

Exemple : résolution d'une équation de d° 2 dans R, en tenant compte du cas $\Delta=0$: 3 cas à traiter

si $\Delta < 0$: pas de solutions

si $\Delta = 0$: une racine réelle double

si $\Delta > 0$: deux racines réelles distinctes

3 conditions exclusives :

Δ est forcément strictement négatif ou strictement positif ou nul !

Alternatives multiples

Tableau récapitulatif **:**

<table>
<tr><td colspan="3">$\Delta < 0$?</td></tr>
<tr><td>oui</td><td colspan="2">non (donc $\Delta >= 0$)</td></tr>
<tr><td rowspan="3">pas de solutions</td><td colspan="2">$\Delta > 0$?</td></tr>
<tr><td>oui</td><td>non (donc $\Delta=0$)</td></tr>
<tr><td>2 solutions distinctes</td><td>1 solution double</td></tr>
</table>

Chaque point d'interrogation est un test (if).

Alternatives multiples

```
#include <stdio.h>
void main()
{
	/* on garde les mêmes variables ainsi que la */
	/* saisie des coefficients, calcul de delta */
	if (delta < 0.)
	{
		/* traitement du cas sans solutions */
	}
	else
	{
		if (delta > 0.)
		{
			/* traitement du cas à 2 solutions */
		}
		else
		{
			/* traitement du cas à une solution */
		}
	}
}
```

Alternatives multiples

Autre syntaxe : on peut enchaîner les if…else if …else au lieu de les imbriquer :plus lisible

reprise de l'exemple précédent :

```
if (delta < 0.)
{
       /* traitement du cas sans solutions */
}
else if (delta > 0.)
{
       /* traitement du cas à 2 solutions */
}
else
{
              /* traitement du cas à une solution */
}
```

Instruction de sélection

Autre instruction pour agir selon la valeur d'une expression entière (de type *char* ou *int*) : **switch** … **case**.

Selon la valeur de l'expression, on liste les actions à faire : presque équivalent à une suite de **if…else if…else if**.

Syntaxe :

```
switch(expression)
        {
                case valeur_1 : instructions;

                case valeur_2 : instructions;

                default : instructions;
        }
```

Instruction de sélection

À la suite de chaque **case** : ensemble d'instructions sans la notation utilisée pour les blocs {}.

Le mot réservé **default** indique les instructions effectuées lorsque la valeur de l'expression ne correspond à aucune valeur listée par les **case**.

Attention aux comportement : quand un **case** est effectué, tous les **case** suivants ainsi que le **default** seront aussi effectués !

Il faut alors utiliser **break**, instruction de rupture de séquence, pour sortir du **switch**.

Instruction de sélection

Exemple et contre-exemple : selon la valeur de la somme de deux variables, on souhaite afficher un message en utilisant switch…case.

Début du programme :

```
#include <stdio.h>

void main()
{
        char    carac1, carac2;

        printf("entrez deux valeurs entières :');
        scanf("%c",&carac1);
        scanf("%c",&carac2);

        /* ici utilisation de switch…case pour choisir le message */

        printf("au reboir !\n");
}
```

Instruction de sélection

Exemple 1 : sans utiliser **break**

Si les valeurs entrées sont 2 et -1 : la somme vaut 1, le premier **case** correspond : le programme affiche 'message numéro 1' puis exécute les instructions de tous les autres **case** et du **default** !

```
switch(carac1+carac2)
{
        case 1 : printf("message numéro 1\n");

        case 2 : printf("message numéro 2\n");

        case 5 : printf("message numéro 5");

        default : printf("message par défaut\n");
}
```

Résultat : affichage de tous les messages

Instruction de sélection

Exemple 1 : sans utiliser **break** (suite). L'ordre des **case** est important : inverser les deux premiers **case**, on suppose que carac1 et carac2 ont les mêmes valeurs que précédemment:

```
switch(carac1+carac2)
{
        case 2 : printf("message numéro 2\n");
        case 1 : printf("message numéro 1\n");

        case 5 : printf("message numéro 5");

        default : printf("message par défaut\n");
}
```

Résultat : affichage des trois derniers messages (1,5,défaut)

Instruction de sélection

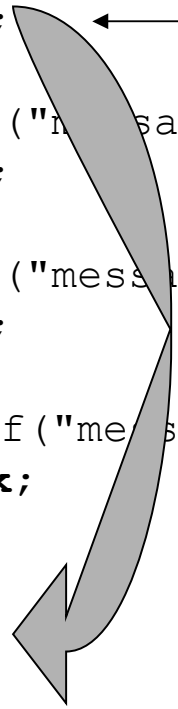
Exemple 1 : utilisation de **break**

```
switch(carac1+carac2)
{
        case 1 :  printf("message numéro 1\n");
                  break;

        case 2 : printf("message numéro 2\n");
                  break;

        case 5 : printf("message numéro 5");
                  break;

        default : printf("message par défaut\n");
                   break;
}
```



*Rupture de séquence : sortie du **switch***

Résultat : affichage du message 1 seul.

Application

Écrire un programme, qui, selon la valeur d'une variable entière nommé error, affiche les messages suivants :

0 : pas d'erreur

1 : erreur de lecture

2 : erreur d'écriture

3 : erreur de taille

4 : erreur de format

5 et + : erreur inconnue.

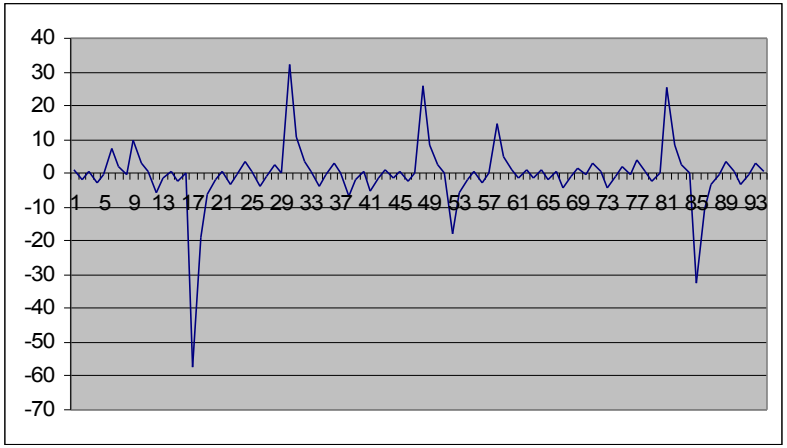
Les itérations

Opération à répéter un certain nombre de fois :

exemple des suites mathématiques :

$$\begin{cases} U_0 \text{ donné,} \\ U_{n+1} = f(U_n) \end{cases}$$

exemple : $U_0 = 1$, $U_{n+1} = \frac{1}{3}.U_n - \frac{2}{U_n}$



Les itérations

On veut la valeur de U6 : calcul par suite d'instructions simples

illustration :

```
#include <stdio.h>
void main()
{
	float terme_U;

	terme_U = 1;  /*initialisation du U avec U0 */
	terme_U = (terme_U/3.0)-(2.0/terme_U); /* calcul U1 */
	terme_U = (terme_U/3.0)-(2.0/terme_U); /* calcul U2 */
	terme_U = (terme_U/3.0)-(2.0/terme_U); /* calcul U3 */
	terme_U = (terme_U/3.0)-(2.0/terme_U); /* calcul U4 */
	terme_U = (terme_U/3.0)-(2.0/terme_U); /* calcul U5 */
	terme_U = (terme_U/3.0)-(2.0/terme_U); /* calcul U6 */

	printf("U6 = %f\n",terme_U);
}
```

Les itérations

6 fois la même instruction !

Si l'on veut calculer U_{150}, on devra répéter cette instruction 150 fois !

Réécrire le programme pour chaque valeur différente !

Autre exemple : l'ordinateur choisit un chiffre, l'utilisateur doit le deviner en proposant des valeurs. A chaque proposition, l'ordinateur indique si la proposition faite est supérieure, inférieure ou égale.

Essayons d'écrire le programme correspondant :

Les itérations

```
#include <stdio.h>
void main()
{
	int nombreMystere, proposition;
	/* initialiser nombreMystere au hasard */
	printf("votre
proposition :");scanf("%d",&proposition);

	if (proposition == nombreMystere)
	{
		printf("gagné !\n");
	}
	else if (proposition < nombreMystere)
	{
		printf("votre nombre est trop petit\n");
	}
	else
	{
		printf("votre nombre est trop grand\n");
	}
}
```

Les itérations

On ne fait les tests qu'une fois ! Il faudrait faire ces tests…

jusqu'à ce qu'on trouve effectivement le bon résultat !

Il existe en C des instructions permettant des itérations : répétition d'instructions quand une condition est satisfaite :

while : tant que

do … while : faire tant que

syntaxe :

while (condition) instruction;

ou

while (condition) {bloc d'instruction}

Les itérations

Effet de **while** : *tant que* la condition précisée est vraie, l'instruction ou le bloc d'instruction suivant est effectué.

On teste d'abord la condition ! Si elle est fausse, on ne fait pas l'instruction ou bloc d'instruction.

Reprise de l'exemple avec la suite :

on se base sur l'indice du terme de la suite à calculer : on réalisera la calcul tant que cet indice est inférieur ou égal à l'indice du terme demandé.

Introduction d'une nouvelle variable :indice, qui indiquera l'étape du calcul.

Utilisation du tant que

Algorithme du programme pour le calcul d'un terme quelconque de la suite :

variables : terme_U : float indice : entier

initialiser terme_U à 1

initialiser indice à 0 /* insister sur ces phases init/test/incrément */

tant que (indice <= 6) faire

    terme_U = terme_U/3-2/terme_U

    indice = indice+1

Les itérations

syntaxe :

do instruction; **while** (condition);

ou

do {bloc d'instruction} **while** (condition);

comme pour le while, la condition est une condition de répétition : tant que cette condition sera vraie, on continuera la boucle.

Différence : l'instruction ou le bloc d'instructions est situé avant le test de la condition, et cet ordre est respecté lors de l'exécution du programme :

on fait les instructions, puis on teste la condition. Si elle est vraie, on recommence les instructions, sinon, on sort de la boucle.

Les itérations

Avec **do…while**, on fait au moins une fois les instructions du corps de la boucle

Avec **while**, on peut ne pas faire les instructions de la boucle, si la condition est fausse dès le premier test !.

Exemple d'applications : saisie sécurisée d'une valeur.

Pour tester si une valeur est correcte, il fait d'abord la saisir !

Si cette valeur est incorrecte, on la ressaisira : on utilise alors une boucle do…while, puisqu'il faut faire les instructions (ici la saisie) AVANT de faire le test de la condition !

Algorithmes utilisant le tant..que

L'ordinateur devine un nombre

saisie sécurisée

le jeu de Nim

Répétition avec la boucle for

Dernier type de boucle : la boucle for, qui sert essentiellement à répéter une ou des instructions un nombre de fois déterminé.

Conceptuellement très proche de la boucle while.

La boucle **for** intègre naturellement la notion de compteur : une variable est utilisée pour compter le nombre de fois où cette boucle doit être effectuée. C'est, de préférence, une variable entière.

Présentation simple (simpliste) de la boucle for dans un premier temps, uniquement avec un compteur entier.

En fait beaucoup plus puissante.

Répétition avec la boucle for

La plupart du temps, on considère que la boucle for est utilisée ainsi :

- **for(initialisation du compteur; condition; incrémentation du compteur) instruction;**

ou

- **for(initialisation du compteur; condition; incrémentation du compteur) {bloc d'instructions}**

ne jamais écrire : for(initialisation du compteur; condition; incrémentation du compteur)**;** {bloc d'instructions};

car le ; indique une instruction vide !

Répétition avec la boucle for

L'initialisation du compteur est faite une seule fois avant le début du premier passage dans la boucle.

La condition est évaluée (testée) avant chaque début de boucle : si elle est vraie, la boucle est faite, puis on re-teste la condition et ainsi de suite (comme pour le while)

L'incrémentation du compteur est faite à chaque fin de boucle, avant de revenir au test de la condition pour le passage suivant dans la boucle

ces rubriques sont séparées par des ;

Répétition avec la boucle for

La condition est une condition sur le compteur en général, c'est toujours une condition de répétition. Si elle est vraie, la boucle continue.
En général, cette condition est un test pour voir si le compteur a dépassé ou non une valeur limite.

Un exemple: emploi de la boucle for pour afficher les carrés de tous les nombres de 1 à 10.
On emploiera une variable entière, nommée compt, qui prendra toutes les valeurs de 1 à 10, et dans la boucle, on affichera son carré.

Répétition avec la boucle for

Recherchons :
la valeur à laquelle doit être initialisé le compteur : on commence à 1.

La condition de la boucle : on continue tant que le compteur est <=10 (ou <11, ce qui revient au même).

L'incrémentation du compteur : additionner 1 à chaque fois.

La boucle s'écrit donc :

```
for (compt=1; compt <11;compt++)
        {
        printf("le carre de %d est %d\n",compt,compt*compt);
        }
```

Répétition avec la boucle for

```
for (compt=1; compt <11;compt++)
        {
        printf("le carre de %d est %d\n",compt,compt*compt);
        }
```

déroulement chronologique :

Initialisation : compt=1.
Test : compt<11, le résultat est : VRAI, la boucle est effectuée : affichage de 'le carre de 1 est 1'.
Incrémentation : compt++, compt vaut 2
Test : compt<11, le résultat est : VRAI; la boucle est effectuée…
...compt vaut 10,
Test compt<11, résultat VRAI, la boucle est effectuée : affichage de 'le carre de 10 est 100'
Incrémentation, compt++, compt vaut 11
Test compt < 11, résultat FAUX, on sort de la boucle.

Répétition avec la boucle for

Autre exemple, pour montrer la souplesse de la boucle for : pas obligé de faire varier le compteur dans un seul sens. En fait, la partie incrémentation peut recevoir n'importe quelle instruction qui sera effectuée en toute fin de boucle.

Affichage des cubes des nombres de 10 à 2 (dans cet ordre).
Valeur initiale du compteur : 10, on ira en 'descendant'
condition de boucle : compteur >=2 ou encore compteur >1
instruction de fin de boucle (plutôt que incrémentation) : compteur--.

```
for (compt=10; compt > 1;compt--)
        {
        printf("le cube de %d est %d\n",compt,compt*compt*compt);
        }
```

Répétition avec la boucle for

En général : boucle for utilisée avec des compteurs entiers, mais similaire à une boucle while !
Exemple : traitement du problème des suites avec la boucle for :

rappel de l'algorithme avec **while** :

variables : terme_U : float indice : entier

initialiser terme_U à 1

initialiser indice à 0 /* insister sur ces phases init/test/incrément */

tant que (**indice <= 6**) faire

    terme_U = terme_U/3-2/terme_U

    indice = indice+1

Répétition avec la boucle for

On retrouve les rubriques pour la boucle for !

```
for(indice = 0; indice <=6; indice++)
{
        terme_U = terme_U/3-2/terme_U
}
```

il y a équivalence entre for et while (pas entre for et do...while).

Comment choisir ? En général (pas de règle absolue), on utilisera les boucles de type for lorsque l'on travaille avec un compteur entier ou un tableau (vu plus tard);
boucles de type while avec les autres cas.

Équivalence for/while

formes équivalentes :

WHILE

```
initialisation
while (condition)
        {
        instructions;
        incrémentation de compteur
        }
```

FOR

```
for(initialisation; condition;incrémentation de compteur)
{
        instructions;
}
```