

IUT ANNECY
Département Mesures Physiques
Cours d'informatique
Initiation au langage C

myriam.chesneau@univ-savoie.fr

Mots clefs :

Initiation à la programmation Langage C Variables Opérations Structures alternatives et itératives Tableaux Fonction Algorithmes

La plus part des exercices proposés sont de grands classiques de l'initiation à la programmation. Ils sont présentés dans de nombreux ouvrages, livres, photocopiés, cours en ligne, et sont indépendants du langage.

CH 1 : Variables et opérations _____	1
Annexe : les caractères _____	12
CH 2 : Structures alternatives _____	13
CH 3 : Structures itératives _____	21
CH 4 : Tableaux _____	26
CH 5 : Fonctions _____	32
Annexe : visibilité _____	40
CH 6 : Bilan _____	41
TEST 2009-2010 _____	43
ANNEXES	
Annexe 1 : utiliser Code::blocks _____	45
Annexe 2 : installer Code::blocks _____	49
Annexe 3 : compléments de langage C _____	50
Annexe 4 : structure d'un programme en C _____	51

CH 1 : VARIABLES ET OPERATIONS

L'ordinateur utilise une représentation binaire de l'information, l'information élémentaire est le **bit**, il ne peut prendre que deux valeurs 0 ou 1.

Programme, fichier texte, fichier image...tout ce qui est utilisé par un ordinateur est donc codé en un ensemble de 0 et de 1.

Ces deux valeurs correspondent à deux états électriques, utilisés pour mémoriser et traiter l'information au sein de l'ordinateur.

Les circuits de l'ordinateur manipulent des ensembles de bits :

- Quartet 4 bits
- Octet 8 bits
- Mot 16, 32 bits (ou plus, à préciser)
- ko, Mo, Go : multiples de l'octet

Dans ce module, nous utiliserons

- les nombres, entiers et réels, dont le codage est introduit au paragraphe suivant, et
- les caractères : chaque caractère est codé par 7 bits en code ASCII standard, sur 8 bits en code ASCII étendu. Ceci est détaillé en annexe 1, en fin de chapitre.

1. CODAGE DES NOMBRES (VOIR COURS D'INFO. D'INSTRUM.)

Un nombre est représenté et codé par un nombre fini (4, 8, 16...) de bits.

1.1 Les entiers

Code binaire naturel sur n bits

Il permet de représenter les entiers naturels compris entre 0 et 2^n-1

Exemple

sur 8 bits on peut coder les entiers positifs de 0 à $2^8 - 1 = 255$

$$73 = 64 + 8 + 1 = 2^6 + 2^3 + 2^0$$

$$= 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \text{ est codé par } 0100\ 1001 \text{ sur 8 bits.}$$

Code complément à deux sur n bits

C'est le plus utilisé en calcul scientifique.

Il permet de représenter les entiers naturels compris entre -2^{n-1} et $2^{n-1}-1$

- les entiers positifs sont codés en binaire naturel
- les entiers négatifs : la valeur absolue est codée en binaire naturel, le résultat est complété (1 \Leftrightarrow 0) puis on ajoute 1 au nombre obtenu.

Exemple

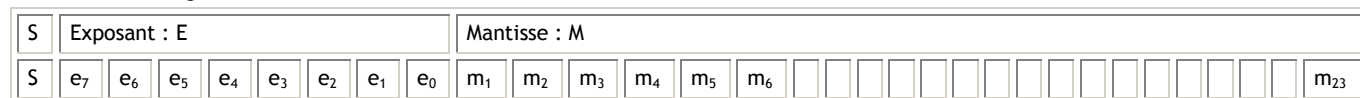
Sur 8 bits on peut coder les entiers de -2^7 à $2^7 - 1$ soit -128 à 127

- 73 est codé par complément de $(0100\ 1001) + 1 = 1011\ 0110 + 1 = \mathbf{1011\ 0111}$ sur 8 bits.

1.2 Les réels

Le codage le plus fréquent est le codage virgule flottante standard (IEEE 754).

Pour un tel codage sur 32 bits



- S bit de signe
- E exposant sur 8 bits à lire en binaire naturel
- M mantisse sur 23 bits à lire comme une partie décimale en binaire ($M = 2^{-1}.m_1 + 2^{-2}.m_2 + \dots + 2^{-23}.m_{23}$)

$$X = (-1)^S \cdot (1+M) \cdot 2^{E-127}$$

2. VARIABLES

2.1 Type et nom des variables

Les nombres et caractères sont stockés dans des variables. Une variable possède un nom et un type adapté.

Les principaux **types de variable** utilisés sont

- les entiers, codés sur 8, 16 ou 32 bits (**int** = integer)
- les réels, codés sur 32 ou 64 bits, (**double** = réel double précision)
- les caractères, codés sur 8 bits, et associés à une lettre ou un signe via le code ASCII.

Exemple

```
char lettre;  
int compte;  
int entier1, entier2;  
double longueur, largeur;
```

On écrit le **nom des variables** en utilisant les lettres minuscules et majuscules, les chiffres, mais pas le _.

On évite d'appeler tous les entiers « n » et tous les réels « x » ou toutes les variables « toto »: on donne des noms significatifs comme « longueur », « taux », « note »...Le programme gagne ainsi en lisibilité.

Les majuscules servent à délimiter les noms, on commencera par une minuscule :

Exemple

```
motDePasse  
bilanJanvier  
reel2  
adresseDeBase
```

2.2 Assignation, initialisation

On peut affecter une valeur à une variable au moment de sa déclaration. On dit qu'on initialise la variable.

Exemple

```
int nbr=2, triple=3*nbr;  
char lettre='A'; // 'A' désigne le caractère A
```

On peut affecter une valeur à une variable après sa déclaration, on dit qu'on assigne une valeur à une variable.

Exemple

```
nbr=3;  
triple = 3 * nbr;  
longueur = 0.12; // réel double précision virgule flottante  
largeur = 1.5e-1; // réel double précision notation scientifique
```

Attention :

```
double quotient;  
quotient = 5/3 ; // 1.00  
quotient = 5./3 ; // 1.67
```

2.3 Conversions

Conversion implicite

Lors d'une opération entre deux variables de types différents, c'est le type « le plus précis » qui est choisi pour évaluer l'opération.

Lors d'une affectation, il y a conversion dans le type de la variable.

A chaque conversion implicite, le compilateur prévient par un WARNING : il est préférable d'être explicite.

Exemples :

```
int nbr1, nbr2 = 3;  
double reel = 1.1;  
nbr1 = nbr2/reel;
```

```
//nbr1 -> double, puis opération sur 2 doubles : 2.73, puis résultat tronqué : 2
cout<<nbr2<<endl;
```

Conversion explicite

Pour transformer le type d'une variable, on fait précéder l'expression par le nouveau type entre parenthèses. Il peut y avoir perte d'information lors d'une conversion explicite.

Exemple

```
double reel;
int partieEntiere;
partieEntiere = (int)reel; // ou encore partieEntiere = int(reel) ;
```

2.4 Constantes

Pour interdire le changement de la valeur d'une variable, on fait précéder son nom, lors de l'assignation, du mot const :

Exemple

```
const int NOMBRE = 3;
const double TAUX = 0.86;
const double PI = 3.14159 ;
```

On écrit le **nom des constantes** en majuscule.

3. OPERATEURS

3.1 Opérateurs arithmétiques

Opérateur	Nom		Exemple	Résultat
+	Identité	unaire	+5	
-	Opposé	unaire	-5	
+	Addition	binaire	3+6	
-	Soustraction	binaire	3.5-2	
*	Produit	binaire	5*2	
/	Quotient	binaire	7/3 11./2.	
%	modulo (n'agit que sur des entiers)	binaire	11%3 12%3	

3.2 Opérateurs relationnels

Une expression utilisant un ou plusieurs opérateurs relationnels vaut 0 si elle est fautive, 1 si elle est vraie.

Opérateur	Nom	Exemple	Résultat pour a=2 b=4
==	égal à	(3==5) (a==b/2)	
!=	différent de	(a!=b)	
<	inférieur	(a<b)	
<=	inférieur ou égal	(a<=b/2)	
>	supérieur	(b>(a%2))	
>=	supérieur ou égal	(b>=a)	

3.3 Opérateurs logiques

Ces opérateurs considèrent toute valeur entière non nulle comme un 1 logique (vrai) et zéro comme un 0 logique (faux).

Opérateur	Nom	Exemple	Résultat pour a=2 b=4
&&	ET logique	(a==2)&&(b>3)	
	OU Logique	(c<d) (c>=d)	
!	NON Logique	!(a>b)	

Evaluation de gauche à droite, jusqu'à ce que le résultat définitif soit trouvé :
 (a<b)&&(c<d) si a>b, la seconde expression n'est pas évaluée.

3.4 Opérateurs de décrémentation et d'incrément

Nous utiliserons juste

```
i++;
```

qui augmente de « 1 » le contenu de i.

Voir en annexe 3 les autres opérateurs.

3.5 Ordre de priorité des opérateurs

On peut bien sûr utiliser des parenthèses lorsqu'on ne connaît pas les priorités : la compréhension peut être plus immédiate. (cf. ex 1.1)

	Opérateur	ordre de priorité		opérateur	ordre de priorité
1	()	gauche à droite	8	&	gauche à droite
2	! ~ ++ -- (type)	droite à gauche	9	^	gauche à droite
3	* / %	gauche à droite	10		gauche à droite
4	+ -	gauche à droite	11	&&	gauche à droite
5	<< >>	gauche à droite	12		gauche à droite
6	< <= > >=	gauche à droite	13	expr condition	droite à gauche
7	== !=	gauche à droite			

1. ANALYSE DE CODE

Déterminer les valeurs de res1, res2 et res3

```
int nbr1, nbr2;
int res1, res2, res3;
nbr1 = 3;
nbr2 = 4;
res1 = nbr2%nbr1;
res2 = 45%4;
res3 = 45%nbr1;
```

Déterminer les valeurs successives de k et x.

```
int i=10, j=4, k;
double x;
k=i+j;
x=i;
x=i/j;
x=(double)i/j;
x=5/i;
k=i%j;
x=105; i=5;
x= x+i;
```

Déterminer les valeurs de x après avoir placé des parenthèses « inutiles » mettant en évidence les priorités :

```
x= 7 + 3 * 6 / 2 - 1 ;
x = 2 % 2 +2*2-2/2 ;
x = (3*9*(3+(9*3/(3)))) ;
```

Déterminer les valeurs successives de q et x

```
int n=5, p=9;
int q;
double x;
q=n<p;
q=n==p;
q=p%n+p>n;
x=p/n;
x=(double)p/n;
x=(int)(p+0.5)/n;
```

Chercher et expliquer les erreurs du programme

```
const int X=10;
int y,z;
y+z = 10;
10 = z;
X=100;
z==10;
```

2. ANALYSES DE PROBLEMES

2.1 Disque

On souhaite réaliser un programme qui

- Reçoit au clavier la valeur numérique du rayon d'un cercle
- Calcule la circonférence et la surface du disque possédant ce rayon
- Affiche les résultats.

Prévoir une application numérique de test, par exemple :

- Si le rayon vaut 1, la circonférence vautet la surface

Définir les variables nécessaires : nom et type.

-
-
-

Ecrire les lignes (ou la ligne) de code correspondantes.

-
-
-

Définir π comme constante. Ligne de code.

-

Quels sont les calculs à réaliser ?

-
-

Quelles sont les opérations d'entrée et de sortie à réaliser (nous verrons comment les coder lors du TP 1).

-
-
-

2.2 Aire

L'aire A d'un triangle dont les côtés ont pour longueur a, b et c peut se faire de la manière suivante :

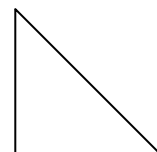
$$A = \sqrt{\text{som} \times (\text{som} - a) \times (\text{som} - b) \times (\text{som} - c)} \text{ où } \text{som} = (a + b + c)/2.$$

Pour un triangle rectangle tel que a = 3, b = 4, calculer c, puis appliquer la formule et la vérifier.

Le programme

- Reçoit au clavier les trois valeurs numériques a, b ,c, longueurs des cotés du triangle.
- Calcul la demi-somme et la surface du triangle.
- Affiche le résultat

Indication : on utilisera la fonction sqrt() pour extraire la racine carré. (ajouter #include <math.h> en tête de programme)



Prévoir une application numérique de test, par exemple :

Dans le cas particulier d'un triangle rectangle tel que a = 3, b = 4, l'aire peut être facilement calculée, elle vaut :

.....Le troisième côté a pour longueur c =, en appliquant la formule, on trouve som =
.....et A = Est-ce bien l'aire attendue ?

Définir les variables nécessaires : nom et type.

-

-

-

Ecrire les lignes de code correspondantes.

-

-

-

-

Quels sont les calculs à réaliser ?

-

-

Quelles sont les opérations d'entrée et de sortie à réaliser (nous verrons comment les coder lors du TP 1).

-

-

-

Consignes valables pour tous les TP

- Tous les programmes sont sauvés dans votre partition.
- Créer un dossier destiné à recevoir vos programmes INFORMATIQUE.
- Les programmes sont recopiés dans un fichier texte (C ou Word) et imprimés une seule fois en fin de TP (ou en dehors de la séance : voir démo prof).

1. PREMIER PROGRAMME : PRISE EN MAIN DU LOGICIEL _____

Ex1 Bonjour.cpp

A l'aide de l'annexe 1, paragraphes 1.1 et 1.2 créer un premier programme

Remplacer le message entre guillemets "Hello World" par votre message personnel...sans accent.

A l'aide de l'annexe 1 paragraphe 1.4, lancer le programme.

Faites-vous expliquer ce programme.

Commenter alors entièrement le programme en ajoutant des commentaires après chaque ligne (// commentaires...).

Remarques :

Par défaut, un répertoire TP1 est créé là où vous avez localisé le projet TP1 : ce répertoire porte le nom du projet.

Il contient des sous-répertoires destinés à recevoir (par défaut) les exécutables et programmes intermédiaires.

Observer le contenu de ce répertoire en passant par l'explorateur.

2. COMPLEMENT DE COURS _____

Tout programme comporte une fonction principale qui commence par la ligne `int main()`, puis comprend le code du programme entre accolades. La dernière ligne est « »

```
int main()  
{  
    ... ;  
    ... ;  
    return 0 ;  
}
```

Quelques explications sur cette structure seront données au chapitre sur les fonctions.

Le squelette général d'un programme en C est donné en annexe 4.

2.1 Ecrire à l'écran

Pour écrire un message à l'écran, suivi d'un retour ligne, on utilise l'instruction **cout** <<suivi du message entre guillemets

```
cout<< "Contenu du message" <<endl ;
```

Pour afficher une valeur numérique, on utilise

```
cout<<3<<endl ;
```

On peut également afficher le résultat d'un calcul :

```
cout<<3*5<<endl ;
```

Pour afficher le contenu d'une variable, on utilise le nom de la variable :

```
int nbr = 5 ;  
cout<<nbr<<endl ;
```

Il est conseillé de terminer chaque instruction cout par un retour ligne (endl).

On peut également afficher par la même instruction des messages et le contenu de variables :

```
double diam = 2 , circonf ;  
circonf = 3.14159*diam ;  
cout<< "La circonference du cercle vaut : " <<circonf<< " metres " <<endl ;
```

Pour utiliser la « fonction » `cout`, on ajoute en début de fichier la ligne

```
#include <iostream>
```

Le programme complet s'écrit alors. :

```
#include <iostream>

using namespace std;

int main()
{
    int nbr = 5 ;
    double diam = 2 , circonf ;
    cout<< " contenu du message " <<endl ;
    cout<<3<<endl ;
    cout<<3*5<<endl ;
    cout<<nbr<<endl ;
    circonf = 3.14159*diam ;
    cout<< " la circonference du cercle vaut: " <<circonf<< " metres"<<endl;
    return 0;
}
```

2.2 Saisir au clavier

Pour saisir une donnée au clavier, et la placer dans une variable, on utilise l'instruction `cin>>`, suivie du nom de la variable. La donnée est frappée au clavier suivie de la touche de validation (Enter : ↵)

```
int i;
cin>>i;
cout<<"le double de "<<i<<" est : "<<2*i<<endl;
```

On peut également saisir plusieurs valeurs successivement à l'aide d'une seule instruction, lors de la saisie, on sépare les entrées par la touche de validation ↵.

```
int cotea, coteb, cotec;
cout<<"Entrez les longueurs des 3 cotes du triangle : "<<endl;
cin>>cotea>>coteb>>cotec;
...
```

Il peut être judicieux, quand on souhaite saisir une donnée, d'en prévenir l'utilisateur par un message :

```
int i;
cout<<"Entrer un nombre entier puis valider"<<endl;
cin>>i;
cout<<"le double de "<<i<<" est : "<<2*i<<endl;
```

3. EXERCICES DE BASE

Ex2 disque.cpp

A l'aide de l'annexe 1, paragraphe 1.3, ranger le premier programme et en créer un nouveau : Ex2Disque.cpp. Ecrire et tester le programme préparé en TD au paragraphe 2.1

Sauver, compiler, exécuter.

Puis, présenter le programme avec des commentaires :

```
// declaration des variables
```

```
...
```

```
// Entrée des donnees
```

```
...
```

```
//Calcul
```

```
...
```

```
//Affichage des résultats
```

```
.....
```

Ex3 aire.cpp

Ecrire et tester le programme préparé en TD au paragraphe 2.2.

Ex4 tva.cpp

Ecrire un programme qui permet de calculer un prix TTC à partir d'un prix hors taxe saisi au clavier, le taux de TVA étant fixé à 19,6 % et **défini comme une constante**.

Ex5 pas a pas.cpp

Lire entièrement l'annexe 1 §2 en repérant les touches citées.

Recopier le programme suivant et l'exécuter pas à pas en recopiant la valeur des variables au fur et à mesure. Attention, la ligne pointée est celle qui sera exécutée au prochain "F7".

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    double r;
    n=2;
    r=n/3;
    r=n/3.;
    n=n/3;
    n=r+6;
    r=n;
    r=r/4;
    n=r;
    r=n;
    return 0;
}
```

Valeur de n	Valeur de r	Expliquer le résultat :
	/	
/		
/		
	/	
	/	
/		
/		
	/	
/		

↔ Faire vérifier

Ex6 operation.cpp

Ecrire un programme qui demande à l'utilisateur d'entrer un réel x et affiche le résultat r_1 de l'opération $\frac{1}{1-x}$.

Modifier le programme pour qu'il affiche en plus le résultat r_2 de l'opération $1 + x + x^2 + x^3$.

Modifier le programme pour afficher l'écart entre les deux résultats ci-dessus calculés : $r_1 - r_2$.

Vérifier, en créant un petit programme, que vous savez :

- Afficher un message à l'écran
- Saisir un nombre au clavier, puis l'afficher à l'écran,
- Afficher un résultat de calcul accompagné d'un message, (ex : resultat = ...)
- Saisir plusieurs nombres à la suite, avec une seule instruction.
- Trouver le fichier en-tête à inclure pour utiliser une fonction quelconque.

4. EXERCICES AVANCES

Ex7 moyenne.cpp

Ecrire un programme qui calcule la moyenne de 2, puis 3 valeurs entières. On utilisera une seule variable pour la saisie des données.

Ex8 papierpeint.cpp

Le but de cet exercice est de calculer le nombre de rouleaux de papiers peint nécessaire à la décoration d'une pièce, en supposant qu'il n'y a pas de raccords.

Voici les données disponibles :

On connaît les dimensions d'un rouleau standard : longueur 10,05 m et largeur 53 cm.

L'utilisateur entre les dimensions de la pièce : hauteur, longueur, largeur.

On souhaite calculer ensuite

- le nombre de bandes par rouleau, le périmètre de la pièce, le nombre total de bandes requises, et enfin le nombre de rouleaux.

Résoudre le problème sans l'ordinateur pour le cas suivant : 2,1 m de hauteur, pièce de 4 m × 3 m.

Définir :

- les constantes, les variables

On utilisera des noms explicites pour chaque constante ou variable.

Les calculs seront effectués sur des réels (double), on utilisera les fonctions d'arrondi à l'entier supérieur (**ceil**) ou inférieur (**floor**), voir l'aide.

Ex9 codeascii.cpp

Lire l'annexe page suivante.

Ecrire un programme qui

- Demande à l'utilisateur d'entrer un nombre compris entre 0 et 255
- Affiche le caractère associé au nombre par le codage ASCII (remarque : les premiers caractères ne donnent pas un caractère à l'affichage).

5. PROJET

P1

- Ecrire un programme qui calcule et affiche la valeur de Pi en utilisant la fonction arc tangente (atan), sachant que $\text{atan}(1) = \pi/4$.
- En utilisant les compléments sur la mise en forme, au chapitre 4, afficher Pi avec 5 décimales.

ANNEXE : LES CARACTERES

Chaque caractère est codé sur 8 bits, ce qui permet de coder 256 caractères.

- 0 et 31 : caractères de contrôle pour l’affichage et l’imprimante, nous ne les utiliserons pas.
- 32 à 127 : caractères de l’alphabet standard US,
- 128 à 255 : dépend du système, différents sous DOS et Windows, caractères non US et semi-graphiques.

Les caractères de l’alphabet standard sont disponible sur le clavier.

Les caractères de code 128 à 255 sont affichés grâce à leur code, en tant que caractères

```
| cout<<char(158)<<endl ; //affiche x
```

Ces caractères ainsi que leur code sont donnés ci-dessous :

code ASCII

0	^@	16	^P	32	ESP	48	0	64	@	80	P	96	`	112	p
1	^A	17	^Q	33	!	49	1	65	A	81	Q	97	a	113	q
2	^B	18	^R	34	"	50	2	66	B	82	R	98	b	114	r
3	^C	19	^S	35	#	51	3	67	C	83	S	99	c	115	s
4	^D	20	^T	36	\$	52	4	68	D	84	T	100	d	116	t
5	^E	21	^U	37	%	53	5	69	E	85	U	101	e	117	u
6	^F	22	^V	38	&	54	6	70	F	86	V	102	f	118	v
7	^G	23	^W	39	'	55	7	71	G	87	W	103	g	119	w
8	^H	24	^X	40	(56	8	72	H	88	X	104	h	120	x
9	^I	25	^Y	41)	57	9	73	I	89	Y	105	i	121	y
10	^J	26	^Z	42	*	58	:	74	J	90	Z	106	j	122	z
11	^K	27	ESC	43	+	59	;	75	K	91	[107	k	123	{
12	^L	28		44	,	60	<	76	L	92	\	108	l	124	
13	^M	29		45	-	61	=	77	M	93]	109	m	125	}
14	^N	30		46	.	62	>	78	N	94	^	110	n	126	~
15	^O	31		47	/	63	?	79	O	95	_	111	o	127	DEL

code ASCII étendu sous DOS

128	Ç	144	Ë	160	á	176	⋮	192	Ł	208	ð	224	ó	240	-
129	ù	145	æ	161	í	177	⋯	193	ł	209	Ð	225	ß	241	±
130	é	146	Æ	162	ó	178	⋰	194	Ł	210	Ê	226	ô	242	▬
131	â	147	ô	163	ú	179	⋱	195	ł	211	Ë	227	ò	243	¾
132	ä	148	ö	164	ñ	180	⋲	196	—	212	È	228	õ	244	¶
133	à	149	ò	165	Ñ	181	Á	197	+	213	***	229	ō	245	§
134	â	150	û	166	ª	182	Â	198	ã	214	í	230	μ	246	+
135	ç	151	ù	167	º	183	Ã	199	Ä	215	î	231	þ	247	.
136	ê	152	ÿ	168	¸	184	Ä	200	Ł	216	ï	232	ƒ	248	°
137	ë	153	Ö	169	©	185	Å	201	ł	217	Ĳ	233	ŧ	249	"
138	è	154	Û	170	¬	186	Å	202	Ł	218	Ĳ	234	ŧ	250	
139	ï	155	ø	171	½	187	Ĳ	203	ł	219	Ĳ	235	ŧ	251	¹
140	î	156	€	172	¼	188	Ĳ	204	ł	220	Ĳ	236	ÿ	252	³
141	ï	157	Ø	173	ı	189	Ĳ	205	=	221	ı	237	Ÿ	253	²
142	Ä	158	×	174	«	190	Ÿ	206	±	222	ı	238	-	254	▬
143	Å	159	f	175	»	191	Ĳ	207	□	223	▬	239	˘	255	DEL

code ASCII étendu sous Windows

128	€	144	•	160		176	°	192	À	208	Ð	224	à	240	ð
129	•	145	˘	161	ı	177	±	193	Á	209	Ñ	225	á	241	ñ
130	,	146	˙	162	ġ	178	²	194	Â	210	Ò	226	â	242	ò
131	f	147	“	163	£	179	³	195	Ã	211	Ó	227	ã	243	ó
132	”	148	”	164	¤	180	´	196	Ä	212	Ô	228	ä	244	ô
133	...	149	•	165	¥	181	µ	197	Å	213	Õ	229	å	245	õ
134	†	150	—	166	¦	182	¶	198	Æ	214	Ö	230	æ	246	ö
135	‡	151	—	167	§	183	¶	199	Ç	215	×	231	ç	247	÷
136	^	152	^	168	¨	184	¶	200	È	216	Ø	232	è	248	ø
137	‰	153	™	169	©	185	¶	201	É	217	Ù	233	é	249	ù
138	Š	154	§	170	ª	186	°	202	Ê	218	Ú	234	ê	250	ú
139	‹	155	›	171	«	187	»	203	Ë	219	Û	235	ë	251	û
140	Œ	156	œ	172	¬	188	¼	204	Ì	220	Ü	236	ì	252	ü
141	•	157	•	173	-	189	½	205	Í	221	Ý	237	í	253	ý
142		158		174	®	190	¾	206	Î	222	Þ	238	î	254	þ
143	•	159	ÿ	175	-	191	¾	207	Ï	223	ß	239	ï	255	DEL

CH 2 : STRUCTURES ALTERNATIVES

1. IF...ELSE

1.1 exemples

```
if (i>0)
{
    cout<<"positif"<<endl;
}
```

```
cout<<"max = "<<endl;
if (a>b)
{
    cout<<a<<endl;
}
else
{
    cout<<b<<endl;
}
```

```
if (nombre>max)
{
    max = nombre;
    cpt++;
}
else if (nombre<min)
{
    min = nombre;
}
```

```
if (note>=12)
{
    cout<<"Admis"<<endl;
}
else if (note<8)
{
    cout<<"Refusé"<<endl;
}
else
{
    cout<<"Oral"<<endl;
}
```

1.2 exercices

En s'inspirant des exemples ci-dessus, compléter les programmes suivants :

Le programme affiche si le nombre est pair ou impair.

```
int nombre;
cout<<"entrer un nombre " <<endl;
cin>>nombre;

--

--
```

Le programme affiche « + » si le nombre est positif, « - » si le nombre est négatif, « 0 » si le nombre est nul.

```
int nombre;
cout<<"entrer un nombre " <<endl;
cin>>nombre;

--

--
```

1.3 structures if...else

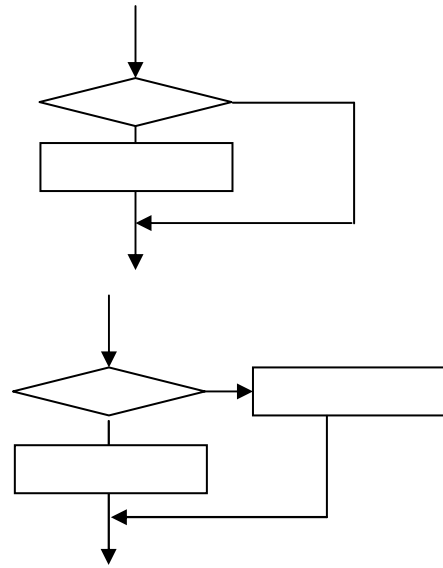
La valeur de l'expression est évaluée et, si elle est vraie, instruction1 est exécutée sinon c'est instruction2 qui est exécutée (si elle existe).

```
if(expression)
{
    instructions1;
}
```

Soit, en « français »:
« si l'expression est vraie, alors on réalise les instructions »

```
if(expression)
{
    instructions1;
}
else
{
    instructions2;
}
```

Soit, en « français »:
« si l'expression est vraie, alors on réalise les instructions 1, sinon, on réalise les instructions 2 »



- L'expression est entourée de parenthèses.
- Une expression est vraie si l'expression entre parenthèse est différente de 0.
- Un ensemble d'instructions est entouré d'accolades.
- Une instruction se termine toujours par un point virgule : ";".
- Plusieurs structures if...else peuvent être imbriquées.

2. SWITCH...CASE

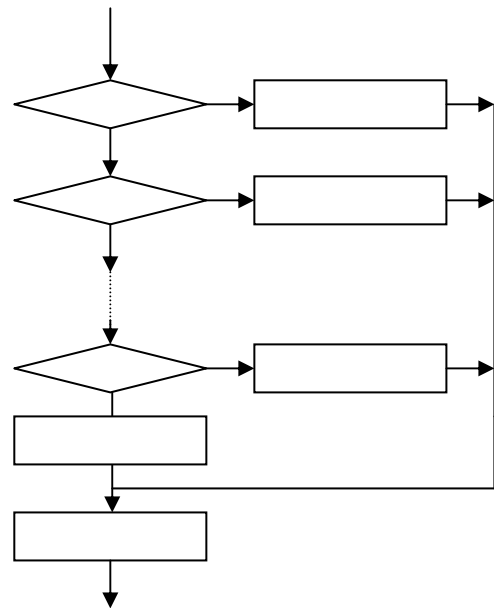
2.1 Exemple

```
int a=2, b=7;
int operation;
cout<<"Choisir une operation : + (1) ou - (2) ou * (3)"<<endl;
cin>>operation;
switch(operation)
{
case 1:
    cout<<"somme = "<<a+b<<endl;
    break;
case 2 :
    cout<<"difference = "<<a-b<<endl;
    break;
case 3 :
    cout<<"produit = "<<a*b<<endl;
    break;
default :
    cout<<"mauvais choix"<<endl;
}
```

2.2 structure

Exécution d'instructions selon la valeur d'expression.

```
switch (expression)
{
    case val1 :
        instructions1;
        break;
    case val2 :
        instructions2;
        break;
    case val3 :
        instructions3;
        break;
    default :
        instructions4;
}
```



Soit, en « français»:

« si l'expression vaut val₁, alors on réalise les instructions 1 et on sort de la structure, sinon, si l'expression vaut val₂, alors on réalise les instructions 2 et on sort de la structure, sinon, si l'expression vaut val₃, alors on réalise les instructions 3 et on sort de la structure, sinon, on réalise les instructions 4 et on sort de la structure.»

- **break** : permet de sortir du switch. Si on omet le break, toutes les instructions suivantes sont exécutées, jusqu'au prochain break.
- **default** : si l'expression ne correspond à aucun des cas, le branchement est réalisé sur default. Si cette instruction est absente, le programme passe à l'instruction suivante.
- On peut mettre n'importe quelle expression dans la parenthèse du switch, par exemple un caractère, un réel, comme le montre l'exemple ci-dessous, similaire à l'exemple précédent.

```
int a=2, b=7;
char operation;
cout<<"Choisir une operation : + ou - ou * "<<endl;
cin>>operation;
switch(operation)
{
    case '+':
        cout<<"somme = "<<a+b<<endl;
        break;
    case '-':
        cout<<"difference = "<<a-b<<endl;
        break;
    case '*':
        cout<<"produit = "<<a*b<<endl;
        break;
    default :
        cout<<"mauvais choix"<<endl;
}
```


Ex 1 Analyse de code

```
int x, c = 0;
cout<<"Entrer un entier"<<endl;
cin>>x;
if((x>=-20)&&(x<=-8))
{
    c = -10;
}
if((x>=1)&&(x<=5))
{
    c = 10;
}
if((x>=34)&&(x<=180))
{
    c = 20;
}
cout<<"c = "<<c<<endl;
```

A quel problème répond ce programme ?

Comment le rendre plus efficace : proposer les nouvelles lignes de code.

Ex 2 Structures if imbriquées

Soient deux réels non nuls x et y , on souhaite déterminer le quadrant dans lequel se trouve le point de coordonnées (x, y) . On suppose que les quadrants sont numérotés de 1 à 4, dans le sens trigonométrique, en partant du quadrant supérieur droit. Représenter par un schéma (organigramme) la démarche qui permet de résoudre ce problème.

Ecrire le code correspondant (x et y sont demandés à l'utilisateur).

Ex 3 Analyse de problème

Le programme calcule les racines réelles d'une équation du second degré et affiche le nombre de solutions réelles et leurs valeurs. ($a x^2 + b x + c = 0$).

Résoudre le problème "sans l'ordinateur" : rappeler les calculs à effectuer pour obtenir la ou les solution(s)

...
...
...
...

Prévoir une application numérique de test,

Trouver trois triplets (a, b, c) permettant de tester chacun des trois cas, quelles sont les solutions attendues dans chaque cas.

- triplet donnant un déterminant nul : $a = \dots, b = \dots, c = \dots, \text{solution} = \dots$
- triplet donnant un déterminant positif : $a = \dots, b = \dots, c = \dots, \text{solutions} = \dots$
- triplet donnant un déterminant négatif : $a = \dots, b = \dots, c = \dots$

Définir les variables nécessaires : nom et type.

-
-
-

Ecrire les lignes (ou la ligne) de code correspondantes.

-
-

-

Quels sont les calculs à réaliser ?

-

-

-

Quelles sont les opérations d'entrée et de sortie à réaliser.

-

-

-

Ex 4 Structure choix

En vous inspirant de l'exemple du cours, écrire un programme qui demande à l'utilisateur d'entrer un nombre réel puis lui propose le menu suivant :

```
| 1 : carré  
| 2 : racine carré  
| 3 : cosinus  
| 4 : sinus
```

Le résultat de l'opération choisie est affiché à l'écran.

Ex 5 Tri de deux ou trois nombres

On souhaite permuter le contenu de deux variables. Proposer une solution, puis un programme complet permettant de tester cette solution (entrée des deux nombres, permutation, affichage).

On souhaite ordonner deux nombres. Le programme reçoit ces deux nombres dans les variables `nbr1` et `nbr2` et échange le contenu des variables si nécessaire, de sorte que `nbr1` contienne le plus petit nombre.

Proposer une démarche, puis écrire le programme complet (entrée des deux nombres, test avec éventuelle permutation, affichage dans l'ordre croissant).

Utiliser les deux exercices précédents pour effectuer le tri dans l'ordre croissant de trois nombres entrés au clavier dans les variables `nbr1`, `nbr2`, `nbr3`.

Tester à la main votre programme pour en vérifier la validité. Trouver les exemples nécessaires au test de tous les cas.

Un petit truc : pour réaligner convenablement votre code : Pluggin/Source Code Formatter... Le style utilisé est le Astyle, pour en savoir plus : en.wikipedia.org/wiki/Indent_style

Un autre petit truc : Pour augmenter la police de caractère dans l'éditeur : Settings / Editor / Choose...

1. DEBUGUER

1.1 Ex 0 Debug

Suivre **pas à pas** l'exécution du programme ci-dessous en visualisant les variables. Justifier les valeurs puis détecter le problème !

```
#include <iostream>
using namespace std;

int main()
{
    int resultat,temp, donnee = 512;
    resultat = 8*donnee*donnee*donnee-1;
    temp = 2*resultat;
    resultat = temp+1;
    temp = resultat+1;
    resultat = resultat*4;
    temp = temp*2;
    resultat = resultat/temp;
    return 0;
}
```

2. EXERCICES DE BASE

Tester les deux programmes du cours, paragraphe 1.2, puis les exercices 2, 3 et 4 du TD :

Ex1 parite.cpp / Ex2 signe.cpp / Ex3 quadrant.cpp / Ex4 racines.cpp / Ex5 operations.cpp

Ex6 tri.cpp

Réaliser le programme d'échange de deux valeurs, le modifier pour obtenir un tri des deux valeurs, puis le modifier pour trier trois valeurs entrées par l'utilisateur.

Ex7 multiple.cpp

Ecrire un programme qui demande à l'utilisateur d'entrer un nombre et affiche si le nombre est multiple de 5 ou non.

Ex8 egal.cpp

Ecrire un programme qui reçoit deux réels x et y et affiche

- « Egaux » si, les deux réels sont alors considérés comme égaux.
- « Différent » sinon.

Ex9 absolue.cpp

Ecrire un programme qui saisit un entier positif ou négatif au clavier et affiche sa valeur absolue. (On n'utilisera pas la fonction valeur absolue du C)

CE QU'IL FAUT FAIRE

Vérifier que vous savez :

- Utiliser un « if » tout seul
- Utiliser un « if/else » avec la bonne indentation (= décalage du texte)
- Permuter le contenu de deux variables à l'aide d'une troisième.

3. EXERCICES AVANCES

Ex10 cryptage.cpp

On dispose de données sous forme d'entiers compris entre 0 et 9999. On choisit de coder ces données avant transmission sur ligne téléphonique, pour les garder secrètes. Le programme doit lire un entier à 4 chiffres et le crypter comme suit :

- chaque chiffre est remplacé par (chiffre + 7) modulo 10, puis,
- le premier et le troisième chiffre sont permutés, puis
- le second et le quatrième chiffre sont permutés.
- le programme doit ensuite afficher le nombre crypté.

On commencera par extraire du nombre et afficher les 4 chiffres correspondants aux milliers, centaines, dizaines et unités.

(Attention, ne pas écrire 0147 mais 147, car un zéro devant un nombre indique - en C - qu'il est codé en octal...)

Ex11 decryptage.cpp

Réaliser ensuite un autre programme pour le décryptage...

Ex 12 jukebox

Récupérer sur le serveur (DUT\MPH\Doc-Etudiants\MPh1\MPh1 Informatique) les 4 fichiers .wav dans le répertoire JUKE. Les copier-coller dans votre répertoire de travail (TP2).

Le programme doit proposer le menu suivant :

```
Entrer votre choix :
A pour Madonna
B pour Mission Impossible
C pour Sex Machine
D pour Gangsta Paradise
```

Le choix de l'utilisateur est saisi au clavier. Selon le choix, le bon morceau (fichier) doit être joué par le PC.

Complément

Pour charger dynamiquement un fichier .wav et jouer ce son

- Placer le fichier .wav dans le même répertoire que votre code source .cpp
- Inclure le fichier qui va permettre d'utiliser le son sous windows.

```
#include <windows.h>
```

- Inclure le fichier d'en-tête mmsystem.h contenant le prototype de la fonction PlaySound() qui va permettre de jouer le fichier .wav, au début du code source

```
#include <mmsystem.h>
```

- Ajouter la bibliothèque « **libwinmm.a** » au projet.
Cette bibliothèque contient entre autre le code d'exécution de la fonction PlaySound().
Elle se trouve dans le répertoire des bibliothèques de CodeBlocks

C:\Program Files\CodeBlocks\MinGW\lib\libwinmm.a ou équivalent....

On l'ajoute au projet en utilisant par exemple **Project / Build Options / Linker Settings / Add / ...**

- Puis, pour jouer le fichier "fichier.wav" et rendre la main au programme à la fin du son, utiliser comme suit la fonction PlaySound :

```
PlaySound ("fichier.wav", NULL, SND_SYNC);
```

4. PROJET

P2

Complément

La fonction rand() renvoie un entier compris entre 0 et 32 767 (tirage pseudo-aléatoire).

La fonction srand permet de réinitialiser le générateur de nombres aléatoires.

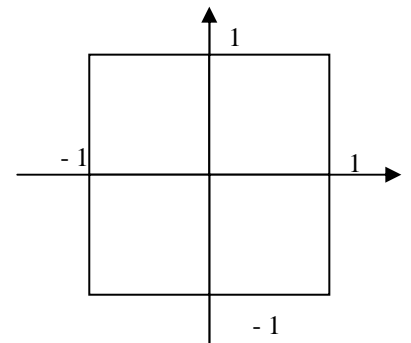
```
srand ( time ( NULL ) ) ;
```

Comment obtenir un réel aléatoire compris entre 0 et 1 ?

(Ces fonctions nécessitent l'appel des fichiers `stdlib.h` et `time.h`)

On souhaite tirer un point $P(x,y)$ au hasard dans le carré ci-contre, puis tester s'il appartient au disque de rayon 1 inscrit dans ce carré.

Proposer une solution.



CH 3 : STRUCTURES ITÉRATIVES : BOUCLES

1. WHILE, DO WHILE

1.1 exemples

```
int k=0, nbr=23;
while (nbr!=0)
{
    nbr = nbr/2;
    k = k+1;
}
cout<<k<<endl;
```

```
int mult;
cout<<"entrer un multiple de 10"<<endl;
do
{
    cin>>mult;
}
while((mult%10)!=0); //parenthèses pour plus de lisibilité
```

1.2 while

Les instructions sont exécutées de façon répétitive aussi longtemps que le résultat de l'expression de continuation à **priori** est vraie.

```
while (expression)
{
    instructions;
}
```

Soit, en « français » : « tant que l'expression est vraie, alors on réalise les instructions »

1.3 do...while

Cette instruction est similaire à la précédente, le test de continuation **a posteriori** a lieu après chaque exécution des instructions, de fait les instructions sont au moins exécutées une fois.

```
do
{
    instructions ;
}
while (expression) ;
```

Soit, en « français » : « On réalise les instructions, tant que l'expression est vraie »

On utilise une structure while ou do...while quand - au moment de l'exécution de la boucle - le « programme » ne sait pas combien de fois la boucle va être exécutée, on utilise une structure for quand on connaît le nombre d'itérations

1.4 exercice

Déterminer les valeurs successives de m, n et de la condition de continuation lors de l'exécution de la boucle :

```
int m=0, n=3;
do
{
    cout<<"m="<<m<<" et n="<<n<<endl;
    m++;
    n--;
}
while(m<n);
```

2. FOR

2.1 exemple

```
int i ;
for (i=0 ; i<4 ; i++ )
{
    cout<<i*i<<endl ;
}
```

2.2 for : utilisation simplifiée

Voici une utilisation simplifiée de la structure for, N est une valeur entière fixée dans le programme :

```
int i ;
for (i=0 ; i<N ; i++ )
{
    instructions ;
}
```

i est initialisé à 0, si i est inférieur à N, les instructions sont réalisées une première fois, puis i est incrémenté, si i < N, les instructions sont à nouveau exécutées, i est incrémenté.... La boucle stoppe quand la condition i < N est fausse.

Donc, « en français » : « Partant de i=0, tant que i est inférieur à N, réaliser les instructions et incrémenter ».

Les instructions sont donc exécutées N fois, pour i variant de 0 à N-1 ; en sortie de boucle, i vaut N

remarques :

i++ est équivalent à i = i+1, c'est à dire une incrémentation de i.

2.3 for : structure générale

La structure générale est

```
for (initialisation ; condition ; mise à jour)
{
    instructions ;
}
```

Soit, « en français » : « Réaliser l'initialisation, puis, tant que la condition est vraie, réaliser les instruction et faire la mise à jour »

On peut donc créer une initialisation, un test (condition) et une mise à jour différentes de celles proposées dans la version simplifiée.

2.4 Exercice

Déterminer l'affichage réalisé lors de l'exécution de cette boucle :

```
int i;
for (i=0; i<=5; i++)
{
    cout<<"i : "<<i<<endl;
}
cout<<"i vaut maintenant : "<<i<<endl;
```

Déterminer la structure utilisée (for ou while) et écrire le contenu de la boucle pour les exercices 1 à 3.

Ex 1

L'utilisateur saisit un nombre entier au clavier et le programme affiche une colonne contenant un nombre d'étoiles correspondant au nombre donné :

```
3
*
*
*
```

Ex 2

Le programme demande à l'utilisateur d'entrer le montant qu'il désire retirer au distributeur. Ce montant doit être un multiple de 20 euros. La question est reposée tant que le montant entré n'est pas correct.

Ex 3

Complément :

La fonction rand() renvoie un entier compris entre 0 et 32 767 (tirage pseudo-aléatoire).

La fonction srand() permet de réinitialiser le générateur de nombres aléatoires.

```
srand(time(NULL));
```

Ces fonctions nécessitent l'appel des fichiers stdlib.h et time.h.

Ecrire un programme qui affiche 10 entiers aléatoires compris entre 0 et 32 767.

Ecrire un programme qui affiche 10 entiers aléatoires compris entre 0 et 10.

Ecrire un programme qui affiche 10 réels aléatoires compris entre 0 et 1.

Ecrire un programme qui affiche 50 nombres aléatoires compris entre 0 et 100, puis le modifier pour qu'il n'affiche que les multiples de 5.

Ex 4

```
int somme, n, i;
somme = 0;
cout<<"Nombre d'iterations ? " <<endl;
cin>>n;
for(i=0 ; i<n ; i = i+1)
    {
        somme = somme + i*i;
    }
cout<<somme<<endl;
```

i	somme

Prévoir l'évolution des variables dans le tableau ci-contre pour n = 5 :

Quelle est la valeur de i en sortie de boucle ?

Que se passe-t-il si on omet la seconde ligne ?

Pourquoi a-t-on choisi une structure for et non une structure while ?

Quelle suite cette boucle calcule-t-elle ?

Ex 5

On souhaite entrer des nombres au clavier (par exemple, saisir les notes d'un examen), puis analyser ces notes. On ignore au départ le nombre de notes à saisir. Les notes étant comprises entre 0 et 20, on décide d'entrer une note négative pour arrêter le traitement.

Ecrire un programme qui

- saisit les notes
- calcule et affiche la somme des notes, attention, on ne désire pas ajouter la valeur négative d'arrêt de boucle.

Complément :

Pour calculer la somme des notes, on initialise la variable somme à 0, puis à chaque fois qu'une note est entrée, on l'ajoute à somme :

```
somme = somme + note ;
```


1. EXERCICES DE BASE

Les trois premiers exercices consistent à programmer des exemples et exercices du cours.

Ex1 while.cpp (§1.1) / Ex2 dowhile.cpp (§1.1) / Ex3 for.cpp (§2.4)

Ex4 etoiles.cpp

Tester l'exercice 1 du TD.

Ex5 distrib.cpp

Tester l'exercice 2 du TD. Modifier ce programme pour, en plus, limiter la somme retirée à 120 euros. Si le montant est supérieur, la question est reposée.

Ex6 alea.cpp

Tester les programmes de l'exercice 3.

Puis, modifier le dernier pour que le programme

- Génère des nombres aléatoires compris entre 0 et 100,
- Affichent parmi ces nombres ceux qui sont multiples de 5 ou de 6,
- S'arrête quand 20 nombres aléatoires multiples de 5 **ou** de 6 sont affichés.

Même question pour afficher 10 multiples aléatoires de 5 **et** de 6.

Ex7 traitement.cpp

Tester le programme de l'exercice 5, le modifier pour afficher la moyenne des notes.

Ex8 suite.cpp

Calculer la somme des (N+1) premiers termes de la suite $1 + x + x^2 + x^3 + x^4 \dots + x^N$.

N sera défini en constante, et x entré par l'utilisateur.

Vérifier que pour x petit devant 1, cette somme approche $1/(1-x)$.

On n'utilisera **pas** la fonction puissance (pow)

CE QU'IL FAUT SAVOIR FAIRE	<p>Vérifier que vous savez :</p> <ul style="list-style-type: none"> ➤ Tester si un entier est multiple d'un autre entier ➤ Combiner des conditions de test (arrêter une boucle si C1 et C2, si C1 ou C2, continuer une boucle si C1 et C2, si C1 ou C2) ➤ Calculer une suite ➤ Utiliser la fonction rand() pour obtenir un nombre aléatoire compris entre 0 et 1 ➤ Utiliser la fonction rand() pour obtenir un entier aléatoire compris entre 0 et N ou entre 1 et N. ➤ Réinitialiser le générateur de nombres aléatoires.
----------------------------	--

2. EXERCICES AVANCES

Ex7 traitement.cpp

Modifier l'exercice 7 pour afficher le maximum et le minimum.

Complément

Pour rechercher le maximum, on utilise une variable max qu'on initialise à 0. Puis, à chaque fois qu'une note est saisie, on compare max à cette note, si la note est supérieure au contenu de max, on place la note dans la variable max, sinon, on ne fait rien.

Dans un cas plus général, le maximum peut être initialisé à n'importe quelle valeur du tableau.

Ex 9 Condensateur.cpp

Soit un condensateur $C = 10 \text{ nF}$, chargé à la tension E_0 .

A l'instant $t=0$, il commence sa décharge dans une résistance $R = 10 \text{ k}\Omega$. La tension à ses bornes est $E(t) = E_0 e^{-t/\tau}$ avec $\tau = RC$.

A quel instant est-il déchargé à 63 % ? ($E = 0,37 E_0$)

Premier programme.

L'utilisateur entre la tension initiale, et un temps t_1 donné. Le programme affiche la tension E_1 du condensateur à t_1 .

Second programme

L'utilisateur entre la tension initiale, et une tension E_1 donnée, inférieure à E_0 . Le programme calcule par valeur approchée le temps au bout duquel cette tension est atteinte, avec une précision de **100 ns**. (On ne demande pas de résoudre l'équation.)

3. PROJET

P3

On souhaite calculer une valeur approchée de pi par la méthode suivante :

- On tire un certain nombre N de points au hasard dans un carré donné.
- On détermine le nombre de points appartenant au cercle inscrit dans ce carré : N_c .
- Pour N grand, le nombre de points N_c est proportionnel à la surface du cercle, N à celle du carré.

On peut ainsi estimer une valeur de pi.

Proposer un programme utilisant cette méthode.

1. TABLEAUX MONODIMENSIONNELS

Un tableau est un ensemble de N données de même type, référencées par un indice.

N est le nombre d'éléments du tableau, un bloc de N variables du type donné est réservé en mémoire.

La taille du tableau est une valeur constante.

1.1 Déclaration sans initialisation:

```
int mois[12];  
double abscisse[100], ordonnee[100];
```

1.2 Déclaration avec initialisation :

```
int tab[6] = {1,3,8,0,5,9};
```

tab[0]	tab[1]	tab[2]	tab[3]	tab[4]	tab[5]
1	3	8	0	5	9

1.3 Initialisation

Elle peut se faire au moment de la déclaration : exemple précédent tab.

Dans le cas où l'initialisation et la déclaration sont simultanées, la taille du tableau peut être omise, le compilateur la calcule d'après le nombre de valeurs d'initialisation :

```
int tab[] = {10,20,30,40,50,60,70,80,90};
```

On peut se contenter de n'initialiser que le début du tableau.

```
double resistances[12]={1., 1.2, 1.8, 2.2}; // le reste à 0
```

Un tableau peut être initialisé plus tard, en affectant une valeur à chaque élément :

```
int notes[56], i;  
for (i=0; i<56; i++)  
{  
    notes[i]=20;  
}  
notes[3] = 13;  
notes[55] = 9;
```

1.4 Taille du tableau

La taille du tableau est une valeur constante.

```
const int NMAX = 10;  
double mesures[NMAX];
```

Remarques

- Si on spécifie trop de valeurs, un message d'erreur le signale.
- Si tout le tableau n'est pas initialisé lors de la déclaration, les éléments non initialisés sont mis à 0 (ce qui correspond au caractère NUL pour un tableau de caractères).
- Il n'y a pas d'affectation, de comparaison, d'opérations arithmétiques directes sur les tableaux.
- `cout<<mesures<<endl;` affiche l'adresse mémoire du tableau mesures.

2. CHAINES DE CARACTERES

Une chaîne de caractères est un tableau qui ne contient que des éléments de type "char", plus un caractère "nul"(\0) après le dernier caractère.

```
| char mot[6] = "I.U.T"
```

mot [0]	mot [1]	mot [2]	mot [3]	mot [4]	mot [5]
I	.	U	.	T	\0

```
| char ville[20] = "annecy";
```

ville [0]	ville [1]	ville [2]	ville [3]	ville [4]	ville [5]	ville [6]	ville [7]	ville [8]	ville [9]	ville [10]	ville [11]	...
a	n	n	e	c	y	\0						

On peut encore omettre la taille du tableau au moment de la déclaration, s'il y a initialisation :

```
char ville[] = "annecy";
```

mot [0]	mot [1]	mot [2]	mot [3]	mot [4]	mot [5]	mot [6]
a	n	n	e	c	y	\0

Ex 1

Ecrire les lignes de code permettant de remplir un tableau de 100 éléments, avec 100 entiers compris entre 0 et 100, choisis aléatoirement.

Ex 2

On souhaite déclarer et initialiser un tableau dont l'indice correspond au numéro du mois, et le contenu au nombre de jour du dit mois. Donner la ligne de code nécessaire.

Ex 3

On souhaite maintenant entrer au clavier un numéro compris entre 1 et 12, et afficher le nombre de jour du mois correspondant. Proposer le code.

Ex 4

Soit un tableau `tab[50]` déjà rempli, quelles lignes de code permettent d'en afficher le contenu, à raison d'une valeur par ligne ?

Ex 5

Rappeler la démarche pour trouver le maximum d'un ensemble d'entiers positifs. Même question pour le minimum. Quelles sont les lignes de code correspondantes ? On supposera l'existence d'un tableau `tab[50]`, rempli de nombres compris entre 0 et 20.

Amélioration : comment résoudre le même problème si on ne connaît pas les bornes des nombres contenus dans le tableau ?

Ex 6

On souhaite simuler 1000 jets de dés et traiter statistiquement les faces obtenues.

- Rappeler comment obtenir un entier compris entre 1 et 6, pour simuler les 6 faces du dé.
- Proposer les lignes de code qui permettent de créer un tableau de N entiers (N sera fixé en constante, à 1000 par exemple, moins en phase de test), et de le remplir de chiffres aléatoirement pris entre 1 et 6.
- A l'aide d'une variable intermédiaire (compte), compter le nombre de "1" présents dans le tableau et afficher ce résultat. Recommencer pour compter le nombre de "2".
- Comment effectuer ce comptage pour les 6 valeurs possibles, en utilisant une boucle ? On pourra réfléchir en supposant l'existence de trois variables :
 - cherche** : qui est le nombre que l'on recherche dans le tableau (1, puis 2...)
 - compte**, qui est le nombre d'occurrence du nombre recherché.
 - stat** : qui est un tableau dont les indices sont les nombres recherchés (de 1 à 6) et le contenu, les différents comptes obtenus.

Par exemple, si on trouve 15 "1" dans un tableau de 100 entiers, on aura `stat[1] = 15`.

Remarque : à chaque fois que l'exercice s'y prêtera, on respectera l'ordre ci-dessous :

- entrée des données ou initialisation, puis
- affichage des données, puis
- traitement des données, puis
- affichage des résultats.

1. EXERCICES DE BASE

Ex1 mois.cpp

Ecrire un programme de test des exercices 2 et 3 du TD.

Indication pour les exercices 2 et 3...et de nombreux exercices travaillant sur les tableaux :

Ces exercices sont fondamentaux.

On conseille de détailler les programmes comme suit :

- Remplir le tableau, puis
- En afficher le contenu, puis
- Effectuer le traitement des données (calculs, tri...) puis,
- Afficher les résultats.

Il y a donc plusieurs boucles for, indépendantes.

Ex2 tableau.cpp

Ecrire un programme qui affiche les entiers multiples de 6 jusqu'à 100. (soit 6×1 , 6×2 , $6 \times 3 \dots$) à l'aide d'une boucle while.

Modifier le programme pour stocker ces valeurs dans un tableau (on déclarera un tableau de taille 50),

Déterminer le nombre d'éléments affichés grâce à un compteur, puis, à l'aide de boucle "for", modifier le programme pour afficher, après stockage, le contenu du tableau.

Ex3 notes.cpp

Ecrire un programme qui remplit un tableau de 10 nombres aléatoires compris entre 0 et 20, puis calcule le maximum et le minimum de ces nombres.

Ex4 stat.cpp

Déclarer et **initialiser** un tableau de 10 notes (comprises entre 0 et 20), calculer, à la calculatrice, la moyenne et l'écart-type de ces 10 nombres.

Les valeurs de test :

- Les 10 nombres.....
- La moyenne calculée :
- L'écart-type obtenu :

Rappeler les formules littérales de calcul de la moyenne et de l'écart-type :

-
-

Quelles sont les sommes à calculer par itération ? Les nommer :

-
-

Définir les variables nécessaires : nom et type.

-
-

Compléter le programme pour qu'il calcule la moyenne, les extrémums et l'écart-type de ces notes. Vérifier les résultats.

Complément

On inclut en début de programme le fichier iomanip pour utiliser les manipulateurs ci-dessous :

#include <iomanip>

setw (int n) permet de fixer la largeur du champ

setfill(char c) permet de fixer un caractère de remplissage

setprecision (int p) permet de fixer un nombre maximum de chiffre significatifs.

```
int n = 13;
```

```
double x = 3.14159;
```

```
cout<<setw(4)<<3<<endl; //affiche 3
```

```
cout<< setw(6)<<setprecision(2)<<x<<endl; //affiche 3.1
```

```
cout<<setw(8)<<setfill(' * ')<<n<<endl; //affiche *****13
```

Application

La console comporte 80 colonnes et 25 lignes.

Application :

Ecrire un programme qui affiche sur une même ligne les 20 premiers entiers ($N = 1$ à 20), chacun sur un champ de largeur 4. On mettra une seule instruction de fin de ligne après la dernière écriture à l'écran.

Ajouter le code permettant d'afficher également les 20 premiers réels inverses des précédents (de $1/1$ à $1/20$), chacun sur un champ de largeur 8 avec 3 chiffres significatifs.

Si on veut afficher 10 nombres sur une ligne, régulièrement répartis, quelle instruction de formatage faut-il utiliser ?

Ajouter le code permettant de saisir un entier positif n et d'afficher tous les entiers compris entre n et $2n$, à raison de 10 par ligne.

CE QU'IL FAUT SAVOIR FAIRE

Vérifier que vous savez :

- Déclarer un tableau, le remplir au moment de la déclaration ou ensuite via le clavier.
- Déclarer un tableau, puis le remplir automatiquement avec des nombres aléatoires.
- Rechercher les extremums d'un tableau
- Mettre en forme des résultats pour une présentation agréable

2. EXERCICES AVANCES

Ex6 table.cpp

Ecrire un programme qui calcule et affiche à l'écran les entiers de 0 à 10, ainsi que leur carré et leur cube, suivant le format suivant :

```
0    0    0
1    1    1
2    4    8
...  ...  ...
9    81   729
10   100  1000
```

Ex7 dés.cpp

Créer et afficher convenablement un tableau de N entiers compris entre 1 et 6. N est une constante qui vaudra 10 pendant la mise au point du programme, puis 100, 1000 ou plus...

Modifier le programme pour, dans une nouvelle boucle, compter et afficher le nombre de "1" du tableau. **FAIRE VERIFIER.**

Réaliser le programme final et afficher convenablement, en pourcentage, les résultats statistiques.

Ex8 tasser.cpp

Déclarer et initialiser un tableau contenant les 10 entiers suivants : 0 1 0 0 8 0 7 6 0 0.

L'afficher. Effacer toutes les valeurs "0" du tableau et tasser les éléments restants. Afficher le tableau résultant.

On utilisera un second tableau dans un premier temps... puis on recommencera l'exercice avec un seul tableau.

3. PROJET

P4

La valeur de π peut être calculée par la somme infinie suivante :

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} \dots \right)$$

Ecrire un programme qui calcule la somme des 100 premiers termes.

Encore mieux...

Modifier pour obtenir un programme qui permet de calculer le nombre de termes de la série nécessaire pour « stabiliser » la Nième décimale, N compris entre 0 et 5. (la vraie valeur de π sera obtenue par la méthode de l'exercice 1)

Encore plus fort...

Le modifier pour que l'utilisateur entre le nombre de décimales souhaitées et qu'il obtienne le nombre de termes de la série utilisé.

1. PRESENTATION DES FONCTIONS

Pour permettre de réutiliser une portion de programme sans réécrire les lignes, on utilise les fonctions.

Lorsqu'on appelle une fonction, une portion de code est exécutée.

Une fonction

- reçoit (en entrée) des informations : les arguments : données
- renvoie (en sortie) une information, la valeur de retour.

1.1 Exemple

```
double Moyenne(int x1, int x2)
{
    return (x1+x2)/2.;
}
int main()
{
    int nMath=15, nPhys=11;
    cout<<Moyenne(nMath, nPhys)<<endl;
    return 0;
}
```

1.2 Description

- Moyenne est le nom de la fonction
- double est le type de la valeur retournée
- x1 et x2 sont les variables dans lesquelles seront placés les arguments formels
- (x1+x2)/2 est la valeur retournée.
-
- On aurait pu également écrire :

```
double moy;
moy = (x1+x2)/2.;
return moy;
```

- Moyenne(nMath, nPhys) est l'appel de la fonction.

Les valeurs numériques des variables nMath et nPhys de main sont recopiées dans les variables x1 et x2 de la fonction.

Représenter le transfert des valeurs sur le programme ci-dessous :

```
int main()
{
    int nMath=15, nPhys=11;
    cout<<Moyenne(nMath, nPhys)<<endl;

    return 0;
}
```

```
double Moyenne(int x1, int x2)
{
    double moy;
    moy = (x1+x2)/2.;
    return moy;
}
```

1.3 Généralisation

A chaque utilisation d'une fonction, on retrouve :

- l'appel de la fonction
- la définition de la fonction en elle même : type de la valeur retournée, nom, type et nom des arguments (Règle de style : on utilisera une majuscule comme première lettre du nom d'une fonction... sauf pour main.)
- le corps de la fonction : les lignes de code entre accolades.

Une fonction renvoie une valeur, cette valeur est

- Soit stockée dans une variable : `var = Fonction(...);`
- Soit directement affichée à l'écran :

```
cout<<Fonction(...)
```

1.4 Exercice

Pour l'exemple précédent, quels sont :

- L'appel de la fonction
- Le type de valeur retournée
- Le nom de la fonction,
- Les arguments de la fonction,
- Le corps de la fonction.

1.5 Prototypage (nous utiliserons par la suite le cas 1)

Une fonction qui apparaît dans `main` doit avoir été déclarée : soit elle est placée avant la fonction `main` (cas 1), soit on écrit avant la fonction `main` son prototype (cas 2)

Cas 1	Cas2
<pre>double Moyenne(int x1, int x2) { return (x1+x2)/2.; } int main() { int nMath=15, nPhys=11; cout<<Moyenne(nMath, nPhys)<<endl; return 0; }</pre>	<pre>double Moyenne(int, int); void main() { int nMath=15, nPhys=11; cout<<Moyenne(nMath, nPhys)<<endl; return 0; } double Moyenne(int x1, int x2) { return (x1+x2)/2.; }</pre>

Le prototype ressemble à la première ligne de déclaration de la fonction, mais comporte un point virgule en fin de ligne et ne spécifie pas le nom des arguments.

Lorsqu'on utilise des fonctions du système, il faut également donner leur prototype. En fait, il suffit d'indiquer au préprocesseur le fichier dans lequel est écrit le prototype. Ces prototypes sont regroupés dans des fichiers en-tête d'extension `.h`.

Ainsi, écrire : `#include <stdio.h>` c'est inclure dans le programme le prototype des fonctions d'entrées / sorties.

Les bibliothèques spécifiques au C++ ne comportent pas d'extension `.h` : `#include <iostream>`

2. CAS PARTICULIERS

2.1 Les fonctions qui ne renvoient rien : les procédures

Une fonction sans valeur de retour est aussi appelée procédure. (Terme non employé dans le polycopié). On le signale en remplaçant le type de la valeur de retour par `void`.

Dans ce cas, on ne trouve pas de `return` dans le corps de la fonction, et l'appel de la fonction apparaît seul dans `main` (pas de traitement de la valeur retournée puisque pas de valeur retournée).

exemple

```
void AfficheEntiers(int n)
{
    int i;
    for(i=0; i<n; i++)
    {
        cout<<i<<endl;
    }
}
```

```
int main()
{
    int fois = 10;
    AfficheEntiers(fois);
    return 0;
}
```

2.2 Les fonctions sans argument

De même, il est possible de définir des fonctions sans argument : la liste des arguments est vide.

exemple

```
int SaisirEntier()
{
    int entier;
    cout<<"entrer un entier"<<endl;
    cin>>entier;
    return entier;
}
int main()
{
    int val;
    val = SaisirEntier();
    cout<<val<<endl;
    return 0;
}
```

Ainsi, la fonction main ne reçoit pas d'argument, mais renvoie un entier : 0.

C'est une manière de savoir que le programme est arrivé à la fin.

Ex 1

Prévoir le résultat du programme du paragraphe 2.1 du cours.

Ex 2

Pour chacune des fonctions ci-dessous, remplir le tableau

```
void Carre(int i)
{
    int j;
    j=i*i;
    cout<<"carré de "<<i<<":"<<j<<endl;
}
```

```
double Pi()
{
    return 3.14159265;
}
```

```
int Multiplie (int a, int b)
{
    return(a*b);
}
```

```
void Message()
{
    cout<<"Au revoir"<<endl;
}
```

	Carre	Pi	Multiplie	Message
Argument(s) : nom et type				
Valeur de retour : type				
Que fait la fonction ?				
Exemple d'appel de la fonction				

Ex 3

Pour les fonctions décrites ci-dessous, prévoir le type de fonction en définissant ses arguments, sa valeur retournée (éventuelle), son rôle et son appel :

- **Parabole** : la fonction reçoit un réel x et renvoie $3x^2 + 1$, sans affichage à l'écran.
- **Maxim** : la fonction reçoit deux entiers et renvoie le plus grand des deux, sans affichage à l'écran.
- **Bonjour** : la fonction affiche "Bonjour" à l'écran.
- **Nbonjour** : la fonction reçoit un nombre entier « nFois » et affiche "Bonjour" un nombre de fois égal à cette valeur.

	Parabole	Maxim	Bonjour	Nbonjour
Argument : nom et type				
Valeur de retour : type				
Que fait la fonction ?				
Exemple d'appel de la fonction				

Ex 4

Même exercice

- Fcube : la fonction reçoit un entier et renvoie son cube ($n^3=n*n*n$).
- SommeCube : la fonction reçoit un entier compris entre 100 et 999, calcule ces unités, ces dizaines et ces centaines, effectue la somme $u^3 + c^3 + d^3$ et l'affiche.
- TestSomCube : La fonction reçoit un entier compris entre 100 et 999, calcule la somme des cubes des unités, dizaines et centaines, n'affiche rien, mais retourne 0 ou 1 selon que cette somme est égale ou non à l'entier reçu.

	Fcube	SommeCube	TestSomCube
Argument : nom et type			
Valeur de retour : type			
Que fait la fonction ?			
Exemple d'appel de la fonction			

Ex 5 : Passage d'arguments... pour information

Seule la valeur numérique (contenu) de l'argument est transmise.

```
void Echange(int a, int b)
{
    int c;
    c=a;
    a=b;
    b=c;
}
int main()
{
    int n=10, p=20;
    cout<<"main: Avant appel de Echange n="<<n<<" p="<<p<<endl;
    Echange(n,p);
    cout<<"main: Apres appel de Echange n="<<n<<" p="<<p<<endl;
    return 0;
}
```

Déterminer le contenu des variables lors du déroulement en pas à pas de ce programme.

Pourquoi n'avoir pas appelé la variable a « n » et la variable b « p » ?

Rq : Cas des tableaux

L'identificateur du tableau, c'est à dire son nom, représente l'adresse du premier élément : `tab = &tab[0]`.

Passer en argument à une fonction le nom d'un tableau revient à lui passer l'adresse mémoire du tableau. On transmet donc le tableau lui même et non une copie de son contenu :

```
void Fonction (double tableau[])
{
    ...
}
int main()
{
    ...
    double reel[10];
    Fonction (reel);
    ...
}
```

exemple :

```
#include <iostream.h>
#include <iomanip.h>

void LireTableau(int N, double tab[])
{
    int i;
    for (i=0; i<N; i++)
    {
        cout<<"tab["<<i<<" ] : "<<endl;
        cin>>tab[i];
    }
}

int main()
{
    const int NMAX = 50;
    int i, nbMes;
    double resultats[NMAX];
    cout<<"Nombre de mesures ? "<<endl;
    cin>>nbMes;
    LireTableau(nbMes, resultats);
    for(i=0; i<nbMes; i++) cout<<setw(10)<<resultats[i];
    cout<<endl;
    return 0;
}
```

1. EXERCICES DE BASE

Ex1 parabole.cpp

Ecrire une fonction qui reçoit un réel x et renvoie $3x^2 + 1$, sans affichage à l'écran. Tester cette fonction en l'appelant dans le programme principal, le réel sera entré au clavier. La fonction principale réalise l'affichage du résultat.

Ex2 maxim.cpp

Ecrire une fonction qui reçoit deux entiers et renvoie le plus grand des deux, sans affichage à l'écran. Tester cette fonction en l'appelant dans le programme principal, les deux entiers seront entrés au clavier. La fonction principale réalise l'affichage du résultat.

Ex3 bonjour.cpp

Ecrire une fonction qui affiche "Bonjour" à l'écran. Tester cette fonction en l'appelant dans le programme principal.

Ex4 Nbonjour.cpp

Ecrire une fonction qui reçoit un nombre entier « nFois » et affiche "Bonjour" un nombre de fois égal à cette valeur. Tester cette fonction en l'appelant dans le programme principal, le nombre d'itérations sera entré au clavier.

Ex5 surface.cpp

Ecrire une fonction qui retourne la valeur numérique de π : $PI()$.
Ecrire une fonction qui reçoit le rayon d'un disque et renvoie sa surface, cette fonction utilise la précédente.
Ecrire la fonction principale permettant de mettre en œuvre ces fonctions.

Ex6 cube.cpp

On cherche les nombres qui comme 153 vérifient $153 = 1^3 + 5^3 + 3^3$

- Ecrire une fonction « Fcube » qui reçoit un entier et renvoie son cube ($n^3 = n * n * n$). Ecrire une fonction principale qui appelle Fcube, et tester cette fonction.
- Ecrire une fonction « SommeCube » qui reçoit un entier compris entre 100 et 999, calcule ces unités, ces dizaines et ces centaines et affiche la somme des cubes des unités, dizaines et centaines. On utilisera Fcube. Utiliser la fonction principale précédente modifiée pour tester cette fonction.
- En partant de la fonction précédente, et en la modifiant, créer une fonction "TestSomCube" qui reçoit un entier compris entre 100 et 999, calcule ses unités, dizaine et centaine, puis la somme des cubes des unités, dizaines et centaines à l'aide de Fcube, n'affiche rien, mais retourne 0 ou 1 selon que cette somme est égale ou non au nombre. Utiliser la fonction principale précédente modifiée pour tester cette fonction.
- Modifier la fonction principale pour déterminer et afficher tous les nombres cdu compris entre 100 et 999 qui vérifient $c^3 + d^3 + u^3 = cdu$.

(153, 370, 371, 407)

Vérifier que vous savez :

- Ecrire une fonction, avec ou sans argument, qui renvoie ou non une valeur : 4 cas.
- Tester avec une fonction principale chacune de ces 4 fonctions
- Extraire d'un entier ses différents digits (unités, dizaines...)

2. EXERCICES AVANCES

Ex7 disparfaits.cpp

Les fonctions seront testées dans un programme principal.

- Ecrire une fonction « Diviseurs » qui reçoit un entier et affiche tous ses diviseurs.
- Ecrire une fonction « SommeDiv » qui reçoit un entier et renvoie la somme de ses diviseurs, l'entier lui-même compris.

Ex : $\sigma(6) = 1 + 2 + 3 + 6 = 12$

- Ecrire une fonction « Premier » qui reçoit un nombre et l'affiche s'il est premier (alors $\sigma(n) = n+1$, sauf pour 1);
- Ecrire une fonction « Disparfait » qui reçoit un nombre et l'affiche s'il est disparfait, (il vérifie $\sigma(n) = 2n$).
- Ecrire une fonction « Triparfait » qui reçoit un nombre et l'affiche s'il est triparfait.

Chercher et afficher les nombres premiers compris entre 1 et 100, les nombres disparfaits compris entre 1 et 1000, les nombres triparfaits compris dans le même intervalle.

(0 6 28 496 puis 0 120 672)

Ex8 factorielle.cpp

Ecrire une fonction qui reçoit un entier et renvoie sa factorielle : $n! = 1.2.3. \dots .n$.

Tester cette fonction avec une fonction principale adéquate.

COMPLEMENT : VISIBILITE ET PASSAGE DE PARAMETRES

bloc d'instructions

Un bloc d'instructions est encadré d'accolades et composé de deux parties : les déclarations locales et les instructions.

Exemples : une fonction, un bloc de condition...

Le C permet la déclaration de variable dans n'importe quel bloc d'instruction (pour des programmes simples, on peut continuer à tout déclarer en tête de fonction, pour une bonne lisibilité).

```
if (N>0)
{
    int i;
    for (i=0; i<N; i++)...
}
```

variables locales

Les variables déclarées dans un bloc d'instructions sont *uniquement visibles à l'intérieur de ce bloc* : on dit que ce sont des **variables locales** à ce bloc.

```
int Fonction(int a);
{
    int x;
    x = 100;
    ...
    while (a>10)
    {
        double x;
        ...
        x = x*a;
    }
}
```

variables globales

Les variables déclarées au début du fichier, à l'extérieur de toutes les fonctions, *sont disponibles pour toutes les fonctions du programme*. Ce sont alors des **variables globales**.

Il faut faire attention à ne pas cacher involontairement des variables globales par des variables locales du même nom.

En général, les variables globales sont déclarées immédiatement derrière les instructions **#include** au début du programme.

```
#include <stdio.h>
int status;
void A(...)
{
    if (status>0) status--;
    else...
}
void B(...)
{
    ...
    status++;
    ...
}
```

Les variables globales sont à utiliser avec précaution : pour nous uniquement lorsque

- plusieurs fonctions qui ne s'appellent pas ont besoin des même variables, ou
- plusieurs fonctions d'un programme ont besoin du même ensemble de variables (un tableau par exemple)

TD - TP 6 : BILAN

Les exercices sont à préparer en TD et à tester en TP.

Ex 1 Somme

L'utilisateur doit entrer un entier N, le programme doit afficher la somme des N premiers entiers : $1 + 2 + 3 + \dots + N-1 + N$.

Ex 2 PGCD

Pour éviter de calculer le PGCD de deux nombres (plus grand commun diviseur), on utilise l'astuce suivante : Soient deux entiers A et B tels que $A > B$. Le PGCD de A et B est le même que celui de A-B et B.

Exemple : pour calculer le PGCD de 30 et 18

PGCD de 30 et 18 = PGCD de (30-18), et 18 = PGCD (12 et 18) = PGCD (18-12) et 12 = PGCD (6 et 12)= PGCD (12-6) et 6 = PGCD de 6 et 6 = 6.

- Calculer à votre tour, à la main, le PGCD de 36 et 16.
- Elaborer un algorithme permettant d'effectuer ce calcul automatiquement.
- Ecrire une fonction qui reçoit deux nombres entiers positifs et renvoie leur PGCD, dans un premier temps, cette fonction affichera **tous les résultats intermédiaires**. Ecrire une fonction principale qui teste la fonction précédente.

Ex 3 dichotomie

Problème

Prévoir la structure d'un programme déterminant, par dichotomie, le zéro d'une fonction f(x) continue sur un intervalle considéré. On suppose connu un intervalle [a, b] sur lequel la fonction change de signe : $f(a).f(b) < 0$.

Exemple d'application :

- Une fonction simple pour commencer : $f(x) = 3x - 2$ dans l'intervalle [0 ; 1]... on doit trouver $x = 2/3$
- Une fonction plus complexe pour tester... $f(x) = \text{atan } x + \text{atan}(x+1) + \text{atan}(x-1) - \pi/2 = 0$, [a, b] = [0 ; 1] ($x \approx 0,82$)

Analyse

On suppose que f(x) ne s'annule qu'une fois sur l'intervalle. Soit m le milieu de [a, b]. Si $f(a).f(m) < 0$, le zéro est entre a et m, sinon, il est entre b et m. On remplace l'intervalle [a,b] par l'intervalle [a,m] ou [b,m], selon la place du zéro, et on divise le nouvel intervalle en deux.

Le traitement est interrompu lorsque l'intervalle [a,b] est suffisamment réduit, c'est à dire lorsque $|b - a|$ est inférieur à la précision souhaitée.

Ecrire la démarche pour résoudre le problème (liste d'action, organigramme...)

Ex 4 tri par minimum

Soit un tableau initial :

	tab[0]	tab[1]				tab[5]
Tableau initial	6	4	3	5	1	2

Premier programme

Ecrire un programme qui permet de repérer l'indice du plus petit élément, afficher cet indice.

Modifier ce programme pour permuter cette plus petite valeur avec celle contenue dans tab[0].

Second programme

Il s'agit maintenant de chercher le plus petit élément du tableau (dimension N), et de le placer en tête (échange avec le premier élément), puis de recommencer sur les N-1 éléments restants, etc...

Effectuer le tri dans le tableau suivant

Tableau initial	6	4	3	5	2	1
Itération 1						
Itération 2						
Itération 3						
Itération 4						
Itération 5						

Proposer un organigramme ou un algorithme.

Définir les variables utilisées, en spécifiant les variables de comptage (boucle) et leurs bornes.

Ecrire le code.

Lors du test, on utilisera

- dans un premiers temps le tableau proposé dans l'exemple, entré à la déclaration, chaque tableau intermédiaire sera affiché,
- puis un tableau de 13 entiers aléatoires générés par la fonction rand(), chaque tableau intermédiaire sera inscrit sur une ligne. (chaque entier sera affiché sur un champ de 6 à l'aide de **stew**, puisqu'il comporte au plus 5 digits, $6*13 = 78 < 80$ colonnes de la console).

Ex 5 tri bulle

Il s'agit de parcourir le tableau du début à la fin, en permutant deux éléments successifs s'ils ne sont pas bien classés. Ce ci permet de placer au fond du tableau le plus grand élément et de pré-trier les autres.

On recommence le parcours jusqu'à ce qu'il ne comporte plus de permutation.

Tableau initial	6	4	3	5	2	1
	4	6				
		3	6			
			5	6		
				2	6	
					1	6

Tableau n°2	4	3	5	2	1	6
	3	4				
		4	2	5		
				1	5	6

Tableau n°3	3	4	2	1	5	6

Tableau n°4						

Tableau n° 5						

Même questions qu'à l'exercice précédent.

Durée 1 h 45

Document autorisé : cahier de TP

Ex 1 : structure conditionnelle 1

Ecrire un programme qui teste si un nombre entier est compris dans l'intervalle fermé [5 ; 10] et affiche le résultat comme suit :

```

Entrez un entier
8
8 est compris entre 5 et 10
Ou
Entrez un entier
12
12 n'est pas compris entre 5 et 10
    
```

Ex 2 : structure conditionnelle 2

Un fournisseur effectue une remise de 3 % sur le montant brut des achats si ce montant dépasse 100 € ou une remise de 5 % si le montant dépasse 300 €.

Ecrire un programme qui demande à l'utilisateur d'entrer le montant brut de ses achats en euros et centimes puis qui calcule et affiche le montant de la réduction et la somme à payer.

```

Saisir le montant des achats
125
Reduction en euros : 3.75
Somme a payer : 121.25
    
```

(45/0/45 348/17.40/330.6)

Ex 3 : structure itérative 1

On cherche le plus grand diviseur d'un nombre, différent du nombre. Exemples : le plus grand diviseur de 16 est 8 ; le plus grand diviseur de 15 est 5. Ecrire un programme qui recherche puis affiche le plus grand diviseur d'un entier entré au clavier par l'utilisateur.

```

Entrez un entier
2431
Le plus grand diviseur de 2431 est 221
    
```

Ex 4 : structure itérative 2

Ecrire un programme qui demande à l'utilisateur d'entrer un entier à trois chiffres (c'est-à-dire compris entre 100 et 999) dont la somme des chiffres vaut 9, et qui repose la question tant que l'utilisateur se trompe.

Modifier l'exercice pour que la question soit également posée si le nombre entré ne possède pas trois chiffres.

```

Entrez un entier a trois chiffre dont la somme des chiffres vaut 9
123
Entrez un entier a trois chiffre dont la somme des chiffres vaut 9
9
Entrez un entier a trois chiffre dont la somme des chiffres vaut 9
9000
Entrez un entier a trois chiffre dont la somme des chiffres vaut 9
423
    
```

Ex 5 : Fonction 1

Soit f la fonction mathématique définie par $f(x) = \sin(x) + x^2 - \sqrt{x}$.

Ecrire une fonction nommée Equation qui reçoit un réel (naturellement exprimé en radian) et renvoie la valeur de la fonction f pour ce réel.

Utiliser cette fonction dans une fonction « main » appropriée pour afficher la table des valeurs de f pour les entiers compris entre 1 et 9, avec 5 chiffres significatifs, sous la forme donnée ci-contre.

```

f(1) = 0.84147
f(2) = 3.4951
f(3) = 7.4091
f(4) = 13.243
f(5) = 21.805
f(6) = 33.271
f(7) = 47.011
f(8) = 62.161
f(9) = 78.412
    
```

Ex 6 : Tableaux

Réaliser un programme qui remplit un tableau de 10 éléments avec des valeurs entières comprises entre 0 et 5, choisies aléatoirement.

Le programme doit

- afficher le contenu du tableau à raison d'un élément par ligne,
- calculer la somme des éléments du tableau et l'afficher
- déterminer l'indice des éléments nuls et les afficher.

Ex 7 : Fonction 2

Ecrire une fonction **Jeu** qui génère trois nombres aléatoires compris entre 1 et 6 (simulant trois jets de dé), **qui affiche** ces trois nombres, qui calcule la somme des trois nombres **et qui renvoie 1** si la somme est supérieure à 13, qui renvoie **0 sinon**.

Créer une fonction principale permettant de tester cette fonction, on affichera « gagné » si la somme dépasse 13, « perdu » sinon.

```
4
1
1
1
perdu    ou    6
5
6
gagne
```

Ex 8 : Bilan

Décomposer un nombre entier **n** en facteurs premiers consiste à trouver tous les diviseurs de ce nombre qui sont des nombres premiers.

$$792 = 2 \times 2 \times 2 \times 3 \times 3 \times 11$$

En fait, on essaie souvent d'aller plus loin en trouvant pour chaque diviseur son nombre d'apparitions dans la décomposition, ainsi on peut écrire :

$$792 = 2^3 \times 3^2 \times 11.$$

Ecrire un programme qui

- demande à l'utilisateur d'entrer un entier
- affiche les fréquences des diviseurs premiers de ce nombre.

Indication :

Dès qu'un diviseur est trouvé, diviser le nombre par ce diviseur jusqu'à ce que le reste soit non nul :

On teste 2 :

792 est divisible par 2, résultat 396 donc 2 divise 1 fois 792

396 est divisible par 2, résultat 198 donc 2 divise 2 fois 792

198 est divisible par 2, résultat 99 donc 2 divise 3 fois 792

99 n'est pas divisible par 2

On teste 3...

```
Entrer un nombre au clavier
792
est divisible
3 fois par 2
2 fois par 3
1 fois par 11
```

ANNEXE 1

UTILISER CODE::BLOCKS

On utilise un environnement de développement intégré (IDE ou EDI) pour

- éditer le code, c'est-à-dire écrire un programme en langage évolué (ici le C, avec quelques instructions volées au C++, donc extension .cpp)
- le compiler pour réaliser un fichier exécutable par le processeur
- le déboguer

L'IDE utilisé ici est Code::Blocks, il est libre et très en vogue en ce moment.

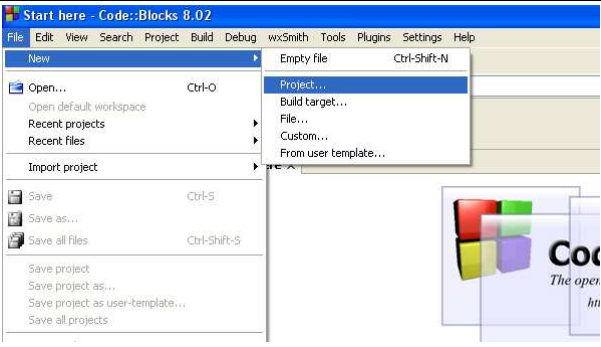
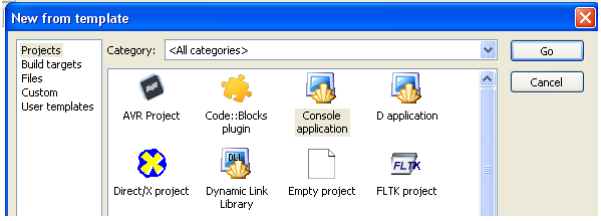
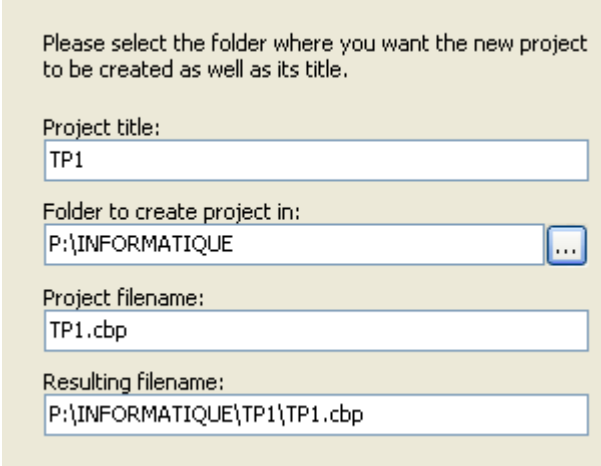
Le compilateur associé est GNU GCC.

Pour installer ces logiciels, voir l'annexe 2

1. DEMARRER SOUS CODE::BOCKS

- Sur votre partition, créer un dossier INFORMATIQUE

1.1 Créer un projet

<ul style="list-style-type: none"> • Ouvrir un nouveau projet.... 	
<ul style="list-style-type: none"> • Utiliser la console (fenêtre "DOS" noire) comme interface d'entrée-sortie. 	
<p>Choisir le langage C++, puis Renseigner la fenêtre :</p> <ul style="list-style-type: none"> • le nom du projet : TP1 • l'endroit où il sera stocké sur le serveur : P:\INFORMATIQUE <p>Next</p>	

<p>Garder les choix proposés</p>	
----------------------------------	--

1.2 Créer un premier fichier dans le projet

<p>Dans la fenêtre Management/Project :</p> <ul style="list-style-type: none"> • Ouvrir le projet TP1, • Ouvrir le répertoire des sources : un fichier main.cpp apparaît. • Avant d'ouvrir ce fichier, le renommer : Ex1Bonjour.cpp 	
---	--

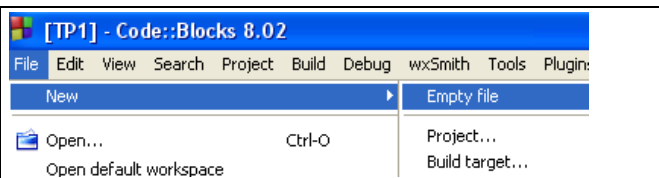
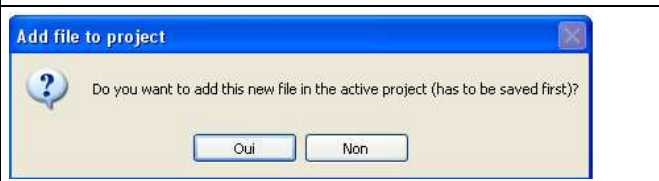
Le code de votre programme remplacera la ligne

`cout << "Hello world!" << endl;`

Tout le reste sera présent dans tous vos programmes.



1.3 Créer un second fichier dans un projet

<ul style="list-style-type: none"> • Quand on ne veut plus travailler sur un fichier, il suffit de le retirer du projet (il reste dans le répertoire de travail, TP1 ici) 	
<ul style="list-style-type: none"> • Ajouter alors un nouveau fichier au projet après avoir sélectionné le projet, 	

<ul style="list-style-type: none"> • File/New/Empty File 	
<ul style="list-style-type: none"> • Accepter d'inclure ce nouveau fichier dans le projet.... 	
<ul style="list-style-type: none"> • Le nommer (xxx.cpp), et cocher au moins "Debug" dans la fenêtre suivante. 	

1.4 Faire "tourner" un programme dans un projet

Barre d'outil Compiler au menu View/Toolbars/Compiler

<p>Une fois un programme écrit : Sauver :</p>	<p>Ctrl S, ou file Save, ou  de la barre de menu.</p>
<ul style="list-style-type: none"> • La roue dentée effectue la « construction » du projet, c'est-à-dire la compilation du fichier et la création du fichier exécutable (code binaire compris par l'ordinateur). • Le bouton "play" vert exécute le programme. • Le troisième bouton construit puis exécute. 	
<div style="text-align: center;">  </div> <div style="display: flex; justify-content: space-between; margin-top: 20px;"> <div style="width: 45%;"> <p><i>Build</i> / Construit le projet / Ctrl F9</p> <p><i>Run</i> / Exécute du projet / Ctrl F10</p> <p><i>Build and Run</i> / Construit et exécute / F9</p> <p><i>Rebuild</i> / reconstruit tout</p> <p><i>Abort</i> / Termine l'exécution en cours</p> </div> <div style="width: 45%; text-align: right;"> <p>Choix du type de cible : version pour chercher les erreurs (<i>Debug</i>) ou version finale (<i>Release</i>)</p> </div> </div>	
<p>Lors de la compilation, si tout se passe bien, on obtient le message ci-contre dans la fenêtre Build log</p>	<pre>Process terminated with status 0 (0 minutes, XX seconds) 0 errors, 0 warnings</pre>

2. SUIVRE PAS A PAS L'EXECUTION D'UN PROGRAMME

2.1 L'environnement

Barre d'outil *Debugger* au menu *View/Toolbars/Debugger*

La barre d'outils du débogueur comporte 7 boutons et un menu, qui permettent de suivre pas à pas l'exécution du programme, ce qui est très utile lorsque le programme fonctionne....mais mal !

<p><i>Debug/Continue</i> Exécute le programme jusqu'au point d'arrêt suivant ou jusqu'à la prochaine erreur. F8</p> <p><i>Run to cursor</i> / Exécute le programme jusqu'à la ligne du code où se trouve le curseur. F4</p> <p><i>Next line</i> / Exécute une ligne sans entrer dans les fonctions F7</p> <p><i>Next Instruction</i> / Exécute une instruction en assembleur</p> <p><i>Step into</i> / Exécute une ligne en entrant dans les fonctions</p> <p><i>Step out</i> / Exécute jusqu'à la fin du bloc en cours</p> <p><i>Stop Debugger</i> / Termine l'exécution en cours</p> <p>Menu d'ouverture des fenêtres de débogage</p>	
--	--

<p>L'ouverture de la fenêtre de débogage <i>Watches</i></p> <p>Glisser cette fenêtre sous la fenêtre de <i>Management</i>.</p>	
--	--

2.2 Pour déboguer un programme :

<p>On place le curseur sur la première ligne intéressante :</p>	<pre>int main() { int i, j, k; i=3; j=3*i; k=i+3*j; cout<<i<<endl<<j<<endl<<k<<endl; return 0; }</pre>
<p>On exécute jusque là :</p>	<p>F4 / Run to Curseur</p>
<p>La flèche jaune indique la prochaine ligne qui sera exécutée :</p>	<pre>5 int main() 6 { 7 int i, j, k; 8 i=3; 9 j=3*i; 10 k=i+3*j; 11 cout<<i<<endl<<j<<endl<<k<<endl; 12 return 0; 13 }</pre>
<p>On exécute les lignes une par une en regardant les variables (<i>Watches</i>)</p>	<p>F7 / Next Line</p>
<p>Plus tard, quand on voudra entrer dans le code d'une fonction ou d'une structure, on utilisera</p>	<p>Ctrl F7 / Step into</p>
<p>Si on doit entrer une donnée dans la fenêtre noire, il faut l'ouvrir en cliquant dans la barre en bas de l'écran :</p>	
<p>On sort du débogueur si on le souhaite, avant la fin du programme :</p>	<p>Stop debugger</p>

ANNEXE 2

INSTALLER CODE::BLOCKS

A ce jour (décembre 2009), voilà ce qu'il faut faire pour télécharger la version 10.05 de Code ::blocks :

- Aller sur le site officiel : www.codeblocks.org
- Download
- **Download the binary release**
- Choisir la version avec MinGw : codeblocks-10.05mingw-setup.exe, télécharger depuis une source quelconque (Download from Sourceforge par exemple)
- Une fois l'exécutable chargé sur votre machine...
- Lancer le...
- Next, Next, Next, Install...
- Lancer Code ::Blocks
- Choisir le compilateur GNU GCC par défaut
- Suivre les conseils de l'annexe 1 !

ANNEXE 3

COMPLEMENTS DE LANGAGE C

Les types de variable

Type		Taille	dynamique
char	caractère (entier sur 8 bits affiché ascii)	entier sur 8 bits-1octet	-128 à 127
unsigned char	caractère non-signé		0 à 255
int	entier	2 octets (sur processeur 16 bits) 4 octets (sur processeur 32 bits)	
unsigned int	entier non-signé		
short	entier	entier sur 16 bits-2octets	-32 768 à 32 767
unsigned short	entier non-signé		0 à 65535
long	entier long	entier sur 32 bits-4octets	-2 147 483 648 à 2 147 483 647
unsigned long	entier long non-signé		0 à 4 294 967 295
float	réel simple précision : 7 chiffres signif.	réel sur 32 bits-4octets	$\pm 1,2 \cdot 10^{-38}$ à $\pm 3,4 \cdot 10^{+38}$
double	réel double précision : 16 chiffres signif.	réel sur 64 bits-8octets	$\pm 2,3 \cdot 10^{-308}$ à $\pm 1,7 \cdot 10^{+308}$
long double	réel max précision : 19 chiffres signif.	réel sur 80 bits-10 octets	$\pm 3,3 \cdot 10^{-4932}$ à $\pm 1,0 \cdot 10^{+4932}$

Les opérateurs logiques entre bits

Opérateur	Nom	Exemples			
~	Inversion (complément à 1)				
<<	décalage à gauche				
>>	décalage à droite				
&	ET Logique (AND)	0&0	0&1	1&0	1&1
^	OU exclusif (XOR)	0^0	0^1	1^0	1^1
	OU Inclusif (OR)	0 0	0 1	1 0	1 1

Les opérateurs de décrémentation et d'incrémententation

Opération	Nom	Exemple	Résultat pour i=3
i++	post incrémententation	j =i++ équivaut à j=i puis i=i+1	
i--	post décrémentation	j =i-- équivaut à j=i puis i=i-1	
++i	pré incrémententation	j=++i équivaut à i=i+1 puis j=i	
--i	pré décrémentation	j=--i équivaut à i=i-1 puis j=i	

• Inclusion des en-têtes	<code>#include <iostream> #include <xxxx.h></code>
• Définition des espaces de noms à utiliser (noms standard du C)	<code>using namespace std;</code>
• Déclaration des fonctions	
• Point d'entrée du <i>programme</i>	<code>int main() {</code>
• Déclaration des constantes et des variables	
• Entrée des données	
• Traitement des données, calculs	
• Affichage des résultats	
• Fin du programme	<code>return 0; }</code>

Il faut prendre l'habitude d'ajouter des commentaires aux programmes, même si ceci peut paraître un peu artificiel lors de l'écriture des premiers programmes.

Pourquoi ajouter des commentaires ?

(Le contenu de ce paragraphe est extrait de <http://fr.wikibooks.org/wiki/Programmation/Commentaires>)

Ajouter des commentaires à un programme permet de décrire ce que fait le code, en utilisant un langage naturel. Il est important d'ajouter des commentaires pour les raisons suivantes :

- Expliquer le rôle d'une fonction
Utiliser une fonction est compliqué si on ne sait pas ce qu'elle fait, ce qu'elle retourne. Une explication permet de savoir comment l'utiliser.
- Expliquer le fonctionnement d'un algorithme complexe
Le code source des algorithmes les plus complexes ne suffit pas à en comprendre le fonctionnement. Une description synthétique permet de le modifier pour l'adapter à un autre projet, ou pour le corriger en cas de problèmes.
- Justifier certains choix techniques
Un code source est souvent repris ultérieurement (correction, réutilisation). Si certains choix ne sont pas justifiés par un commentaire explicatif, ils pourront être remis en cause par le développeur.

Comment ajouter des commentaires ?

- Pour écrire une ligne de commentaires, faire précéder la ligne de `//`.
- Pour écrire plusieurs lignes de commentaires, encadrer le paragraphe par `/*` (début) et `*/` (fin).

```
// ceci est un commentaire sur une ligne

/* ceci
   est
   un
   commentaire
   sur
   plusieurs lignes */
```