Algorithmique et programmation

I. Initiation à la programmation en C++

1. Formalisme

A. Bien écrire un code C++

On fait le programme avec l'IDE Geany et l'on enregistre en .cc

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
{
  cout << "bonjour!" << endl;
  return 0;
}
```

Bibliothèques obligatoire pour commencer à coder

Et commande obligatoire

Cout -> Afficher

<< -> indique la direction vers laquelle ça va

Endl signifie aller à la ligne, il faut toujours mettre des ;

Pour compiler un programme, il faut exécuter la commande :

g++ nomdufichierenentree.cc –o (option qui signifie en sortie) nomdufichierensortie

ou pour la version avec le Warning :

g++ -Wall nomdufichierenentree.cc –o (option qui signifie en sortie) nomdufichierensortie

Pour exécuter le programme, il faut juste donner le nom du programme tout en se placant dans le bon répertoire. ./nomdufichierdesortie

2. Variable et type

-Une variable est une entité qui contient une information, elle possède un type qui caractérise l'ensemble des valeurs que peut prendre la variable et qui restera le même tout le long du programme, et possède une valeur qui peut changer dans le temps, l'ancienne valeur attribuée est alors perdue

-Les types les plus courants sont :

- -Int pour les entiers comme l'âge d'une personne
- -Float pour les réels comme le poids d'une personne
- -Char pour écrire un caractère comme une lettre
- -Bool pour faire des questions vrai/faux

Affiche à l'écran « ce texte »

Affiche la valeur de la variable a soit ici 3.

La variable entière « a » prend la valeur 3.

```
int a=3;
cout << "la variable a vaut " << a << endl;
float b=-1.5;
cout << "la variable b vaut " << b << endl;
char c='@';
cout << "la variable c vaut " << c << endl;
bool d=true;
cout << "la variable d vaut " << d << endl;
bool e=false;
cout << "la variable e vaut " << e << endl;
```

-On retrouve les opérateurs les plus courant : +, /, -, *, pour les entiers on retrouve également le % qui nous donne le reste d'une division euclidienne.

Il affichera le résultat de l'opération

```
cout << "a+b=" << a+b << endl;
```

-On retrouve également les opérateurs logiques tel que le :

- et &&, ou ||, égale booléen ==, le non a !a, l'inégalité !=, le supérieur à >, le supérieur ou égale >=, l'inférieur à <, l'inférieur ou égal <=

-le && est prioritaire sur le || dans une opération

Cin demande à l'utilisateur la valeur d'une variable

On choisit le nom de ses variables

```
int note;
char lettre;
note=10;
lettre='A';
cout << "Un ";
cout << lettre
```

```
int note;
int noteFinale;
cout << "Veuillez saisir un entier entre 0 et 10" << endl;
cin >> note;
```

-Il faut toujours prévoir à l'avance ce qui peut arriver dans une variable pour faire attention aux réels et aux entiers car si l'on met un réel dans un entier l'on obtient uniquement la partie entière de ce réel. Pour les opérations on choisira donc le plus souvent des réels

-Pour faire des commentaires dans un programme afin de le comprendre, on le met comme ceci : //commentaire

-Pour savoir comment fonctionne un algorithme, on en fait la trace des étapes dans un tableau en suivant chaque phase de ce programme.

```
int a,b;
int Tab[3];
a =5;
b =2+a;
Tab[0] =a+b;
Tab[1] =Tab[0]-b;
Tab[2] =Tab[1]-Tab[0];
Tab[0] =Tab[0]-Tab[2];
```

a	b	Tab[]		
		Tab[0]	Tab[1]	Tab[2]
5	?	?	?	?
5	7	?	?	?
5	7	12	?	?
5	7	12	5	?
5	7	12	5	-7
5	7	19	5	-7

Imaginons que nous devions rentrer 1000 variables, ce serait long et inutile, on peut donc rassembler ces valeurs dans un tableau. La variable se déclare donc avec [un nombre de case] et son type, attention l'indice d'un tableau commence par zéro. Ensuite pour attribuer une valeur à une case du tableau il suffit de lui dire Variable[numdelacase]=valeur. Tous les éléments d'un tableau sont du même type, sa taille est fixe et son nombre de case prédéfini.

Affectation

L'affectation se fait simplement avec l'opérateur =. Un tableau de n cases indice ses cases de 0 à $n-1$

```
int monTab[4];
monTab[0] = 0;
monTab[1] =3;
monTab[2]=-1;
monTab[3]=4333 ;
```

Raccourci : a++ signifie a=a+1, a*=2 signifie a=a*2 et cela fonctionne avec tous les opérateurs.

3. Tests et boucles (séquence, choix, itérations)

A. Les conditionnelles :

Permettent de ne pas exécuter une conversion d'euros en francs si la somme est négative, de ne pas résoudre une équation du second degré si le D < 0…

Syntaxe

```
if (Expression booléenne)  Pas de ";"
    {Instructions réalisées si l'expression a été évaluée à VRAI}
else
    {Instructions réalisées si l'expression a été évaluée à FAUX}
```

else if (sinon si)

B. Les boucles :

Mais il devient vite limité d'utilisé le if tout le temps, il existe donc des commandes de répétitions :

Les déterministes où le nombre de boucle est défini à l'entrée de la boucle et les indéterministes où l'exécution de la prochaine boucle est conditionnée par une expression booléenne.

// Un commentaire simple s'écrit comme ceci

/* un commentaire plus long et sur plusieurs lignes */

C. La boucle for :

For (i=0, i<variable, i++) c'est donc la définition de la variable qui compte, sa condition et son pas.

Syntaxe

for (variable numérique =valeur de début ; tant que condition vérifiée ; modifier la variable numérique de ...)
{Instructions à réaliser dans la boucle}

On peut déclarer le type de la variable dans la boucle à chaque tour ou au tout début dans le paramètre.

D. La boucle tant que :

Syntaxe

while (expression booléenne)
{Instructions à réaliser dans la boucle}

Variante

do {Instructions à réaliser dans la boucle}
while (expression booléenne);

La variante n'est pas très conseillée.

Pour « s'arrêter dès que » on ajoute && dans la condition. Penser à l'incrémentation pour ne pas être dans une boucle infinie et penser au début et à la fin de la boucle voir si l'on ne va pas trop loin..

E. Quelques programmes types vus en TD :

```
// Ce programme de saisir un entier jusqu'à ce que 0 soit saisit:

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

int main ()
{
    int nb;
    cin >> nb;
    while (nb !=0)
    {
        cout << "Entrez un entier (0 pour arrêter)" << endl;
        cin >> nb;
    }
    Return EXIT_SUCCESS;
}
```

```cpp
/*
Ce programme de saisir un entier jusqu'à ce que 0 soit saisit
et compte le nombre d'entiers saisient */

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

int main ()
{
    int nb;
    int cpt=0;
    cin >> nb;
    while (nb !=0)
    {
        cpt++;
        cout << "Entrez un entier (0 pour arrêter)" << endl;
        cin >> nb;
    }
    cout << "Vous avez entrer "<< cpt << " entiers" <<endl;
    Return EXIT_SUCCESS;
}
```

```cpp
/*
Ce programme de saisir un entier jusqu'à ce que 0 soit saisit
et fait la somme des nombres d'entiers saisient */

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

int main ()
{
    int nb;
    int val=0;
    cin >> nb;
    while (nb !=0)
    {
        val=val+nb;
        cout << "Entrez un entier (0 pour arrêter)" << endl;
        cin >> nb;
    }
    cout << "La somme est "<< val <<endl;
    Return EXIT_SUCCESS;
}
```

```cpp
/*
Ce programme de saisir un entier jusqu'à ce que 0 soit saisit
et fait la moyenne des nombres d'entiers saisient */

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

int main ()
{
    int nb;
    int val=0;
    int nb_saisies=0;
    float avg=0.0;
    cout << "Entrez un entier (0 pour arrêter)" << endl;
    cin >> nb;
    if (nb ==0){
        cout << "Il n'y a rien à calculer " << endl;
    }
    else{
        while (nb !=0){
                nb_saisies++;
                val=val+nb;
                cout << "Entrez un entier (0 pour arrêter)" << endl;
                cin >> nb;
        }
    }
    cout << "La somme est "<< val <<endl;
    cout << "Vous avez entré " << nb_saisies << "nombres " <<endl;
    avg=1.0*val/nb_saisies; // s'il n'y avait pas le 1.0 il l'afficherait comme un entier
    cout <<"Votre moyenne est donc" << avg << endl;
    Return EXIT_SUCCESS;
}
```

```cpp
/*
Ce programme de saisir un entier jusqu'à ce que 0 soit saisit
et nous dit le plus grand saisit */

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

int main ()
{
    int nb;
    int gap=0;
    cout << "Entrez un entier (0 pour arrêter)" << endl;
    cin >> nb;
    while (nb !=0)
    {
        if (nb > gap)
        {
            gap=nb;
        }
        cout << "Entrez un entier (0 pour arrêter)" << endl;
        cin >> nb;

    }
    cout << "Le plus grand nombre saisis est "<< gap <<endl;
    Return EXIT_SUCCESS;
}
```

F. Conversion entre le C++ et le langage algorithmique papier (EXALGO)

C++	Exalgo
Int, float, bool, char, string	Entier, réel, booléen, chaine de caractères
Int a,b ; A=3 ; A==3 ;	Var a, b : entier A ←3 A=3

If (a ==3)
{…}
Else
{…}
	Si a=3
	Alors […]
	Sinon […]

Cin >> a	Lire (a)
Cout << a	Ecrire (a)
For (int i=0; i<10;i++) {….}	Pour i allant de 0 à 9 avec pas 1 Faire […]
Float MaFonction (int a, int &b) Void Monaction	Fonction : MaFonction (E a entier, S b entier) ; réel Action : Mon action …
Return a	Retourner a

4. Fonctions et procédure

L'intérêt d'une fonction est de diviser un gros problème en successions de problèmes simples.

Elles peuvent être caractérisées par un argument et contiennent des instructions. Voici une instruction qui permet d'afficher un caractère spécifique n fois.

Action : void

```
////////////////////////////
//FONCTION dessineNCaracs//
////////////////////////////
void dessineNCaracs(int n,char val)
{
for (int i=0;i<n;i++)
  cout << val;
cout << endl;
}
////////////////////////////
//FONCTION main      //
int main()
{
dessineNEtoiles(8,'a');
return 0;
}
```

A. La sortie

On utilise une fonction pour faire plus d'une fois un traitement, paramétrer un traitement qui sera réalisé à chaque appel de celle-ci, structurer le traitement, échanges clairs d'information, et réutilisation. Un paramètre de fonction est fixe, une fonction ne possède qu'un seul type qui sera celui de la valeur retourner. Une fonction ne peut retourner qu'un seul résultat. Void est une action, tandis que le reste est pour une fonction qui servira à interagie avec le programme principal.

```
int max(int a,int b)
{
if (a>=b)
  return a;
else
  return b;
}

int main(void)
{
int resultat;
resultat=max(3,5);
cout << resultat << endl;
return 0;
}
```

- void afficheNom(char tab[taille]);
- bool estPresent(int val);
- double racine(double valeur);
- void afficheDate(bool val);

```
int main(){
  afficheNom("bonjour");
  char bonjour[taille]="bonjour";
  afficheNom(bonjour);

  bool res;
  estPresent(4);
  estPresent("4");
  res=estPresent(4.0);

  double val=racine(4.0);

  afficheDate(res);
  afficheDate(TRUE);
  afficheDate(false);
  int a=afficheDate(false);
  affichedate(false);

  return 0;
}
```

Double est comme float mais en plus précis. Il n'est pas possible d'affecter une valeur de retour à un void.

B. Passage de paramètre en fonction

Il y a deux modes de transmission d'information entre fonction appelante et fonction appelée :

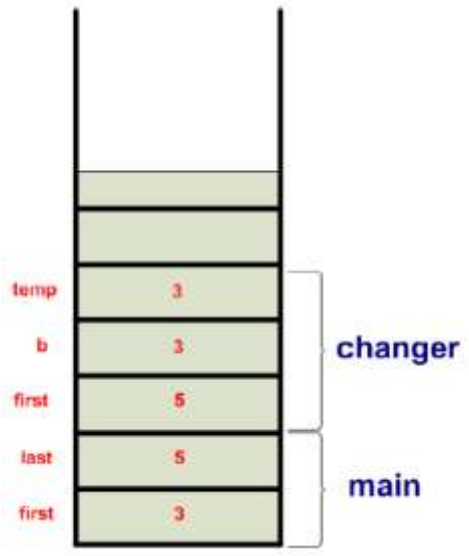
- La transmission d'un paramètre en copie (ou par valeur)

La fonction appelée travaille sur une variable accessible à elle seule ; elle peut la modifier, mais cela ne peut pas avoir de conséquence sur les variables de la fonction appelante.

```
void changer(int first, int b)
{
  int temp; temp=first;
  first=b;
  b=temp;
  cout << "Dedans : first vaut" << first << " et last vaut "<< b << endl;
}

int main()
{
  int first=3; int last=5;
  cout << "Avant : first vaut" << first << " et last vaut " <<last << endl;
  changer(first,last);
  cout << "Apres : first vaut" << first << " et last vaut " <<last << endl;
  return 0;
}
```

```
Avant : first vaut 3 et last vaut 5
Dedans : first vaut 5 et last vaut 3
Apres : first vaut 3 et last vaut 5
```



Voici une pile d'exécution

- La transmission d'un paramètre par référence :

La fonction appelée accèdera à la même variable que la fonction appelante. Conséquence : les modifications effectuées par la fonction appelée sur cette variable demeureront dans la fonction appelante après l'appel. Pour mettre en place ce mode de transmission, il suffit de mettre un & dans la fonction appelée.

```
 void changer(int & first , int & b)
{
  int temp; temp=first;
  first=b;
  b=temp;
  cout << "Dedans : first vaut" << first << " et last vaut "<< b << endl;
}

int main()
{
  int first=3; int last=5;
  cout << "Avant : first vaut" << first << " et last vaut " <<last << endl;
  changer(first , last);
  cout << "Apres : first vaut" << first << " et last vaut " <<last << endl;
  return 0;
}
```

C. Les tableaux à deux dimensions :

déclaration

On déclare un tableau (ou matrice) à deux dimensions de la manière suivante :
type Tableau[TaillePremièreDimension**][**TailleDeuxièmeDimension**]**
On accède à la i^{eme},j^{eme} valeur d'un tableau en utilisant la syntaxe suivante :
nom de la variable[i][j]

Attention

tab[1][0] est différent de tab[0][1]

Definition

Si lors de la déclaration, le tableau est initialisé (valeurs entre {}, la taille de la dimension est optionnelle).

```
int tab1[10];/*declare un tableau de 10 entiers*/
float tab2[2][5];/*matrice de reels de dimension (2,5)*/
int tab3[]={2,3,4};/*un tableau de trois entiers initialise*/
                   /*donc tab3[0]=2,tab3[1]=3 et tab3[2]=4}*/
char tab4[]="bonjour";/*declare un tableau de 7 caracteres initialises*/
int tab5[10]={3,4,5};/*initialise les 3 premiere valeurs a 3,4 et 5,
                       les autres ne sont pas initialisees*/
int tab6[2][3]={1,2,3,4,5,6};/*(1,2,3
                               4,5,6)*/
```

On accède à une valeur de tableau en suivant l'identifiant du tableau encadré par des [ ], les indices sont obligatoires de type entier, le premier indice d'une dimension est toujours 0, Il faut toujours faire attention à ne pas dépasser les limites du tableau car il n'y a pas de vérification de débordement ni à la compilation ni à l'exécution.

Rappel :

Remplir un tableau 1D

```
int tab[5];
for (int i=0;i<=4;i++)
  cin >> tab[i];
```

Afficher le contenu d'un tableau 1D

```
int tab[5];
for (int i=0;i<=4;i++)
  cout « "la valeur dans la case "
« i « " est" « tab[i] « endl;
```

Trouver la position d'un chiffre dans un tableau 1D

```
int tab[5]={-1,32,2,-1,-5};
int nombre,position,i;
cin >> nombre;
bool trouve=false;
i=0;
while (trouve==false && i<5){
  if (tab[i]==nombre)
  {
   trouve=true;
   position=i;
  }
 i++;
}
if (trouve==true)
  cout << "Element trouve a la position " << position << " du tableau" << endl;
else
  cout << "Element non present dans le tableau" << endl;
```

Remplir un tableau 2D

```
int tab[2][3];
for (int i=0;i<2;i++)
 for (int j=0;j<=2;j++)
   cin >> tab[i][j];
//Ou aussi
for (int j=0;j<=2;j++)
 for (int i=0;i<=4;i++)
   cin >> tab[i][j];
```

Afficher un tableau 2D

```
int tab[2][3];
for (int i=0;i<=1;i++)
  for (int j=0;j<=2;j++)
    cout << "la valeur dans la case ["
    << i << "][" << j << "] est "
    << tab[i][j] << endl;
```

Trouver un chiffre dans un tableau 2D

```
int tab[2][5]={-1,32,2,-1,-5,
      11,0,-4,2,50};
int nombre,positionL,positionC,i,j;
cin >> nombre;
bool trouve=false;
i=0;
j=0;
while (trouve==false && i<2){
  while (trouve==false && j<5){
    if (tab[i][j]==nombre){
     trouve=true;
     positionL=i;//position de la ligne
     positionC=j;//position de la colonne
    }
    j++;
  }
i++;
j=0;
}
if (trouve==true)
  cout << "Element trouve a la ligne " << positionL << " et colonne "
       << positionC << " du tableau" << endl;
else
  cout << "Element non present dans le tableau" << endl;
```

D. Passer un tableau en paramètre

Définition

1. Lorsque le paramètre est un tableau 1D, on ne précise pas obligatoirement la dimension de ce dernier.
2. Un tableau n'est JAMAIS passé par copie. Même si ce n'est pas indiqué, le tableau passé en paramètre n'est jamais copié
3. Lorsque le paramètre est un tableau 2D, il faut au moins préciser sa deuxième dimension

```
int FCT1(int m[][TAILLE], int tab[]);
```

```
const int TAILLE=2;

void affFirstTab(int tab1[],int tableau2[][TAILLE], int tableau3[TAILLE]){
  cout << "tab1[0]= " << tab1[0] << endl;
  cout << "tableau2[0][0]= " << tableau2[0][0]<< endl;
  cout << "tableau3[0]=" << tableau3[0]<< endl;
  tab1[0]=10;
  tableau2[0][0]=-2;
  tableau3[0]=tab1[0]+tableau2[0][0];
 }

 int main(){
  int tab1[]={7,6,-1};
  int tab2[TAILLE][TAILLE]={2,3,-1,8};
  int tab3[TAILLE]={4,32};
  affFirstTab(tab1,tab2,tab3);
  cout << "Retour de la fonction" << endl;
  cout << "tab1[0]= " << tab1[0] << endl;
  cout << "tab2[0][0]= " << tab2[0][0] << endl;
  cout << "tab3[0]= " << tab3[0] << endl;
}
```

Les questions à se poser quand on est devant un programme :

1- Fonction ou action, une fonction retourne une valeur contrairement à l'action (void en c++)

2-Paramètres d'entrée

3- Type de sortie, Si oui lequel ?

4-Appel à la fonction

5-Corps de la fonction principale.

E. La structure conditionnelle à choix multiples

(Le switch en C++)

Exemple en Exalgo :

```
Action Mention :
Début :
Lire Note :
Selon que :
Note > 16 : écrire TB
Note > 14 : Ecrire B
Note > 12 : Ecrire AB
Note >10 Passable
Sinon écrire « Ajourné »
Fin Selon Que.
```

Le C++ n'accepte que des égalités, pas de < ou > ou <= ou >=. Et il faut mettre des break si l'on veut que le programme s'arrête dès qu'une condition est vérifiée.

F. Le répéter jusqu'à :

Pour la somme :

Exalgo	C++

Action Somme
Var n, som : entiers
Début :
Som ← 0
Répéter :
Lire n
Som ←som + n
Jusqu'à n=0
Ecrire Som
Fin
	Void Somme ()
	{
	Int n, som ;
	Som = 0
	Do{
	Cin >> n ;
	Som = som +n;
	}While (n!=0);
	Cout << som << endl,
	}

G. Tableau et passage par reference

Déclarer un tableau de 100 entiers :

Const TAILLE=100

Type TTab=tableau[TAILLE] d'entiers

Var t=TTAB ;

Par convention, les constantes sont en majuscules, On le met en global avant TOUT les exercices concernant les tableaux.

Pour l'afficher après avoir saisi des entiers :

```
Var n,i entier ;
    T :TTab
Début :
Ecrire « Saisir n <TAILLE »
Lire « n » /*C'est la taille réel il faudra tester si l'utilisateur est bien dans l'intervalle */
Pour i de 0 à n -1
Lire « t[i] »
Pour i de 0 à n -1
Afficher « t[i] »
Fin
```

Ecrire une action paramétrée qui Saisir Tableau qui prend en sortie le tableau et sa taille, une action paramétrée Afficher Tableau et une action principale.

```
Const TAILLE=100
Type TTab=tableau[TAILLE] d'entiers
Var t=TTAB ;

Action AfficherTableau (E t :TTab, E : n entier)
Var i : entier
Début :
Pour i de 0 à n -1
Afficher « t[i] »
Fin

Action SaisirTableau (S t :TTab, S : n entier)
```

… Voir le programme précédent et réaliser les appels de fonction dans le main.