C++

Ch 5 – Procédures et fonctions

I. Introduction

A. Fonctions et procédures en C++

Une fonction est un sous-programme qui retourne une valeur résultat alors qu'une procédure ne fait qu'exécuter un bloc d'instructions.
Le langage C++ définit une seule forme de sous-programme : la **fonction**.
Cependant le langage met en œuvre un type de retour particulier (**void**) pour distinguer les fonctions qui retournent une valeur (les « vraies » fonctions) de celles qui ne retournent aucune valeur (« les procédures »).

Vous avez déjà utilisé une fonction dans tous les programmes que vous avez écrits en C++ : en effet, l'algorithme principal d'une application écrite en C++ est une **fonction** qui s'appelle « **main** » : c'est le **point d'entrée du programme** (là où commence l'exécution)

```
int main () // fonction principale appelée lors du lancement
{
. . .// déclarations et instructions
  return 0 ; // retourne la valeur 0 à l'appelant (le système)
}
```

II. Fonctions

Une **FONCTION** C++ est un **SOUS-PROGRAMME** qui
- possède un **TYPE DE DONNEES DE RETOUR AUTRE QUE void**
- inclut l' **INSTRUCTION return** qui **VA RETOURNER UNE VALEUR RESULTAT** à l'appelant**.**

A. DEFINIR une fonction : ENTETE et CORPS de la fonction

Syntaxe de la DEFINITION d'une fonction:

```
type_retour fNom_fonction (par1,...,parN)
{
  // corps de la fonction : déclaration et instructions
  . . .
  return valeur_retour ;
}
```

- **type_retour :** type de données du résultat renvoyé par l'instruction **return**
  - type de données de base : int, double, char, string, bool,
  - structures (mot clef struct)
- **fNom_fonction** : identifiant de la fonction (convention : préfixé par 'f')
- **par1, . . ., parN** : déclaration des paramètres attendus par la fonction
- **return = QUITTE LA FONCTION ET RENVOIE LA VALEUR**
  - **valeur_retour :** valeur littérale ou expression ; même type que **type_retour**

Exemple :
Définition de la fonction fCalcDuree :

```
int fCalculerDuree (int PVanDeb, int PVanFin )
{
   int Vduree;
   Vduree = PVanFin – PVanDeb ;
   return Vduree ;
}
```

Utilisation de la fonction fCalculerDurée :

```
. . .
   int VanNais, VanCour ;
   cin >> VanNais >> VanCour ;
   cout << "age : " << fCalculerDuree(VanNais,VanCour)<< " ans" ;
 . . .
```

III.Procédures

Une **PROCEDURE** C++ est un **SOUS-PROGRAMME** qui
- possède un **TYPE DE DONNEES DE RETOUR void**
- et donc ne peut renvoyer de valeur de retour

A. DEFINIR une procédure : ENTETE et CORPS de la procédure

Syntaxe de la DEFINITION d'une procédure :

```
void pNom_procedure (par1,...,parN)
{
  // Corps de la procédure : déclaration et instructions
  . . .
}
```

- **pNom_procedure** : nom identifiant la procédure (convention : préfixé par 'p')
- **par1, . . ., parN** : déclaration des paramètres attendus par la procédure

Exemple de définition sans déclaration de paramètres :
Définition de la procédure pAfficherMenu (aucun paramètre)

```
void pAfficherMenu ()
{
    cout << endl << "Menu";
    cout << endl << "====" ;
    cout << endl << "choix 1 : surface d'un rectangle" ;
    cout << endl << "choix 2 : surface d'un cercle" ;
    cout << endl << "choix 0 : quitter" ;
}
```

Utilisation de la procédure :pAfficherMenu :

```
. . .
    int Vchoix ;
    pAfficherMenu() ;
    cin >> Vchoix ;
    . . .
```

Exemple de définition avec déclaration de paramètres :
Définition de la procédure pAffSomme :

```
void pAffSomme(int PVnb1, int PVnb2 )
{
    int Vsomme ;
    Vsomme = PVnb1 + PVnb2 ;
    cout << "la somme est " << Vsomme ;
}
```

Utilisation de la procédure :pAffSomme :

```
    cin >> Vn1 >> Vn2 ;
    pAffSomme(Vn1, Vn2) ;
    . . .
```

IV. Paramètres (Arguments)

A. Mode de passage des paramètres

1. Passage par valeur

Dans le mode de **PASSAGE PAR VALEUR en C++**, chaque paramètre est déclaré comme toute déclaration de variable (ou constante) et, au moment de l'appel, il est affectée de la valeur de l'argument fourni..

Exemple :

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

// Définition fonc.
int fCalcSurfCarre(int PVcote)
{
    int Vsurf ;
    Vsurf = PVcote * PVcote ;
    return Vsurf ; // retourner la valeur de Vsurf à l'appelant
}

int main ()
{
    int Vcote, Vcalc ;

    cout << "Entrez le coté :" ;
    cin >> Vcote ;
    Vcalc = fCalcSurfCarre(Vcote) ;
    cout << Vcalc ;
    return EXIT_SUCCESS; // constante fournie par C++, vaut 0
}
```

2. Passage par référence : opérateur & (C++)

Dans le mode de **PASSAGE PAR REFERENCE en C++**, l'**OPERATEUR &** : est **AJOUTE APRES LE TYPE DE DONNEES DANS LA DECLARATION DU PARAMETRE** : cela signifie : « ***ce paramètre est une forme d'alias vers la variable passée en argument*** ».

On a ainsi un accès direct à des variables définies dans une autre portée. LES VALEURS D'ORIGINE PEUVENT ETRE MODIFIEES par les instructions du sous-programme, sauf précision du mot clef **const** devant la déclaration du paramètre (référence constante)..

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

```
// Définition
void pPermuter(int & refPV1, int & refPV2)
{
int Vtemp ; // variable locale
    Vtemp = refPV1 ;
    refPV1 = refPV2 ; // modification de la variable réelle
    refPV2 = Vtemp ; // modification de la variable réelle
}

int main ()
{
int Vnb1, Vnb2 ;

    cout << "Entrez 2 nombres :" ;
    cin >> Vnb1 >> Vnb2 ;%
    pPermuter(Vnb1, Vnb2) ;
    cout << Vnb1 << "-" << Vnb2 ;
    return EXIT_SUCCESS; // constante fournie par C++, vaut 0
}
```

3. Passage par adresse : opérateur * et &

(cf. prochain chapitre sur les pointeurs).

V. DECLARATION : prototype de fonction

TOUTE FONCTION C++ DOIT ETRE CONNUE AVANT SON APPEL : ELLE DOIT DONC :

- **SOIT ETRE DEFINIE AVANT SON UTILISATION** dans le programme source (implémentation complète : entête et corps – cf. Fonctions et Procédures
- **SOIT ETRE DECLAREE AVANT SON UTILISATION** (déclaration d'une forme réduite de la fonction, le PROTOTYPE, la définition complète étant réalisée à un autre endroit du programme source ou dans un autre programme source – cf. DECLARATION : prototype de fonction et Organisation des fichiers d'un programme C++)

A. Prototype

Le prototype d'une fonction correspond à la partie de l'entête qui va permettre au compilateur d'identifier cette fonction, de connaître les types des paramètres (éventuellement leur valeur par défaut)

Syntaxe :

```
type_retour fNom_fonction (par1,...,parN) ;
```

- **nom_fonc** est le nom donné à la fonction,
- **typ1, typ2, typN** : mode de passage et type de données des arguments attendus par la fonction, avec possibilité de spécifier de valeurs par défaut
- **type_retour** : type de la donnée retournée par la fonction.

```
int fCalculerSecondes(int, int , int) ;
```

```
void pAfficherHeure(int, int , int) ;
```

Seuls les types des paramètres sont indispensables pour le prototype : **il est cependant conseillé d'indiquer leurs noms pour faciliter la compréhension** :

```
int fCalculerSecondes(int PVheures, int PVmin , int PVsec);
void pAfficherHeure(int PVheures, int PVmin , int PVsec);
```

B. Valeurs par défaut des paramètres

Il est possible de **définir des valeurs par défaut** pour les paramètres **dans le prototype** d'une fonction.

```
int fCalculerSecondes(int PVheures, int PVmin = 0, int PVsec = 0);
```

Lors de l'appel :

```
Vnbre = fCalculerSecondes(12);
```

VI. Surcharge : notion de signature

Exemple : une fonction permettant d'effectuer la somme de nombres de types différents

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

// prototypes des fonctions
int fAdditionner(int PV1, int PV2) ;
double fAdditionner(double PV1, double PV2) ;

int main ()
{
// déclaration const. Et var. locales
int VnbEnt1, VnbEnt2, VresultEnt ;
double VnbReel1, VnbReel2, VresultReel ;

   cout << "Entrez 2 nombres entiers:" ;
   cin >> VnbEnt1 >> VnbEnt2 ;
   VresultEnt = fAdditionner(VnbEnt1, VnbEnt2) ;
   cout << "le resultat est " << VresultEnt ;

   cout << "Entrez 2 nombres reels:" ;
   cin >> VnbReel1 >> VnbReel2 ;
   VresultReel = fAdditionner(VnbReel1, VnbReel2) ;
   cout << "le resultat est " << VresultReel ;

}
// Définition des fonctions
int fAdditionner(int PV1, int PV2)
{
   cout << "additioner 2 entiers";
   return (PV1 + PV2) ;
```

```
}
double fAdditionner(double PV1, double PV2)
{
   cout << "additioner 2 reels";
   return (PV1 + PV2) ;
}
```

VII. Portée des déclarations

La notion de **PORTEE** fait référence à la région d'un algorithme à l'intérieur de laquelle une donnée (variable ou constante) est connue.

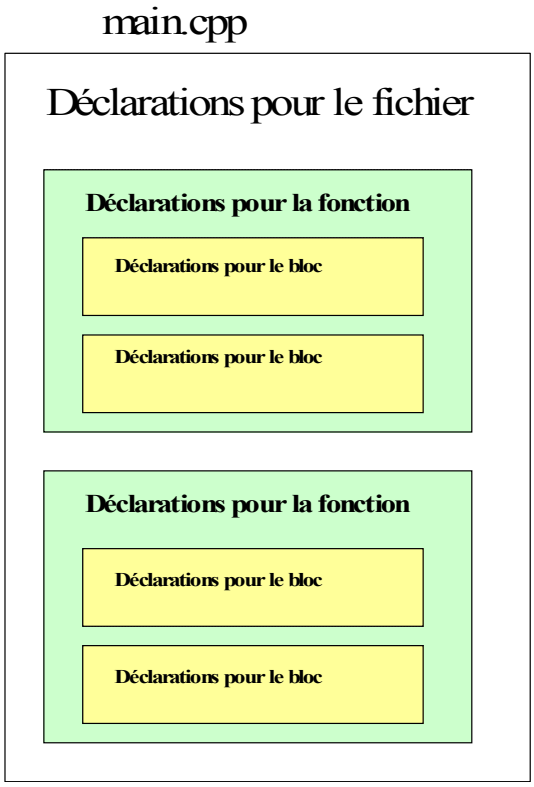
Anglais : scope (fr. étendue, portée)

En C++, on peut effectuer des déclarations :

- Au niveau **fichier**
  - En C++, **le fichier forme le bloc de niveau le plus élevé** : toute déclaration effectuée à ce niveau est connue dans toutes les fonctions du fichier et tous les blocs imbriqués (elle est globale au fichier) ;
  - Au trouve fréquemment à ce niveau les **prototypes** des fonctions utilisées dans ce fichier ;
- Au niveau **fonction**
  - Toute déclaration effectuée dans une **fonction** (locale à la fonction) est disponible à partir de sa déclaration, jusqu'à la fin de la fonction, et dans les blocs imbriqués ;
- Au niveau **bloc**
  - En C++ (comme dans la plupart des langages), une déclaration peut être réalisée à n'importe quel endroit du code. Une déclaration réalisée dans un **bloc** n'est disponible que dans ce bloc, à partir de sa déclaration jusqu'à la fin du bloc, et dans les blocs imbriqués.

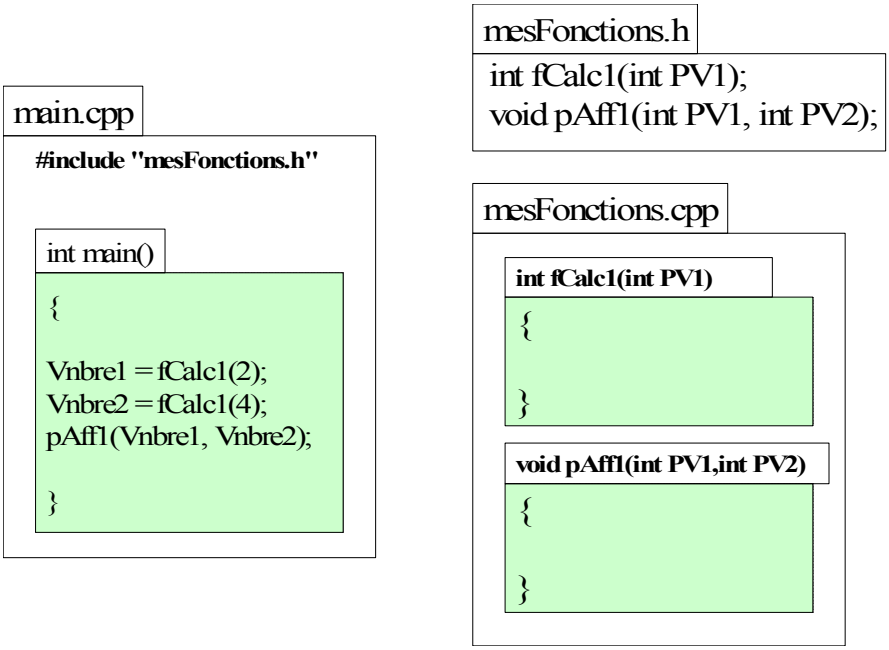
Déclaration fréquente dans un bloc : la variable de boucle dans la structure for :

```
for (int i = 0, i<MAX ;i++)
{
    // corps de la boucle = portée de la variable i
}
```



VIII. Organisation des fichiers d'un programme C++

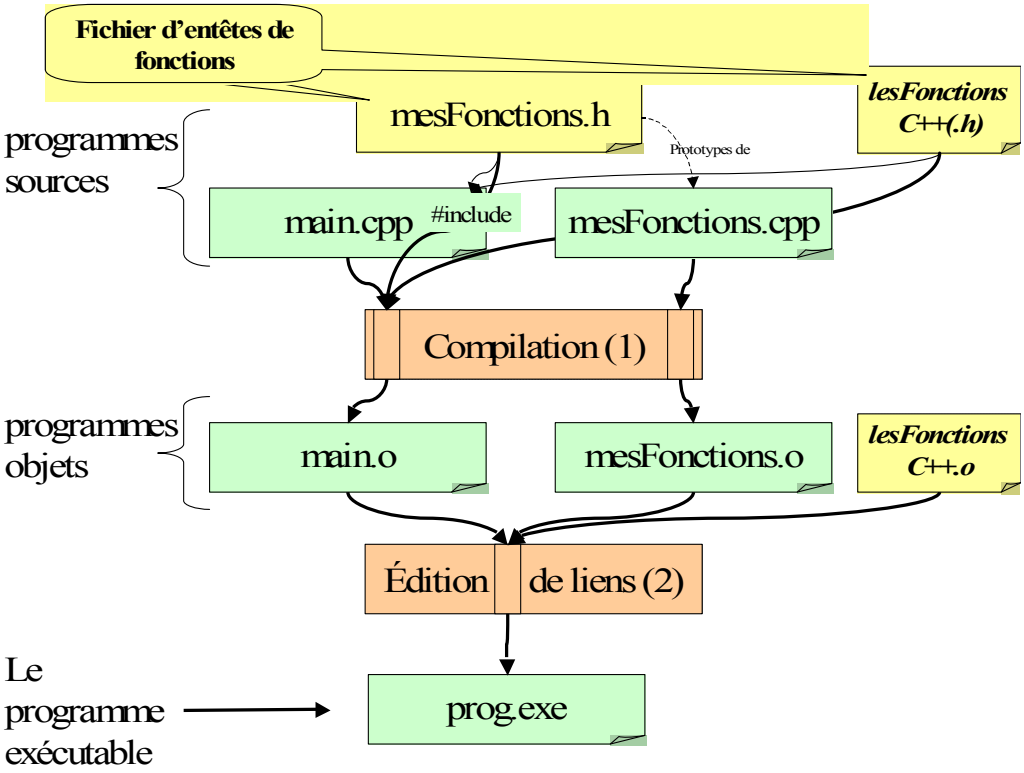
A. Organisation des fichiers sources



Exemple de fichier d'entête : (pour éviter les inclusions multiples, sources d'erreurs) :

```
#ifndef MESFONCTIONS_H
#define MESFONCTIONS _H
int fCalc1(int PV1) ; // prototype
void pAff1(int PV1, int PV2) ; // prototype
#endif
```

B. Compilation (compilation + édition de liens)



IX. Annexes

A. Tableaux et passage de paramètres

En C++, les tableaux sont passés par référence ; en effet, le nom d'un tableau passé en argument est interprété comme l'adresse de son premier élément. **Il NE faut donc PAS préciser d'opérateur de référence & dans la déclaration des paramètres**.

Exemple de syntaxe de déclaration :

```
void pNom_procedure (int RTnote[], int PVindFin)
{
  // Corps de la procédure : déclaration et instructions
  // modification de RTnote en contrôlant qu'on n'utilise
pas un indice hors des limites du tableau
}
```

- **RTnote** : va recevoir une référence vers le tableau qui sera passé en argument.
- **PVindFin** : on doit associer une variable qui va contenir l'indice du dernier élément ou bien le nombre d'éléments du tableau

Exemple de syntaxe d'appel :

```
const int NB_ELEM = 20;
const int IND_FIN = NB_ELEM - 1;
int Tnote[NB_ELEM];

pNomProcedure(Tnote, IND_FIN) ;
```

1. Passage d'un tableau et MODIFICATION DES VALEURS

```
#include <iostream>
using namespace std ;
// prototypes
void pInitialiser(int RTnote[], int PIND_DEB, int PIND_FIN);
bool fAjouter(int RTnote[], int & RVindLibre, int PIND_FIN, int
PVnote) ;
void pAfficher(const int RTnote[], int PVindDeb, int PVindLibre) ;

int main ()
{
    // Constantes et variables de gestion du tableau
    const int NB_ELEM = 20 ;
    const int IND_DEB = 0 ;
    const int IND_FIN = NB_ELEM – 1 ;
    int  Tnote[NB_ELEM];
    int VindLibre = 0 ;

    int Vnote ;
```

```
    pInitialiser(Tnote, IND_DEB, IND_FIN) ;
    Vnote = 10 ;
    if (fAjouter(Tnote,VindLibre,IND_FIN, Vnote)==true)
        cout << "OK : la note a été ajoutée" ;
        else
        cout << "ERREUR : le note n'a pas pu être ajoutée" ;
    pAfficher(Tnote, IND_DEB, VindLibre) ;
    return EXIT_SUCCESS; // constante fournie par C++, vaut 0
}
// procédure pInitialiser : initialiser le tableau
// *********************************************
// Paramètres :
//    reference : RTnote[] : entier, tableau des notes
//    valeur : PVindDeb, entier, indice premier élément
//    valeur : PVinDFin, entier, indice dernier élément
//
void pInitialiser(int RTnote[], int PVindDeb, int PVinDFin)
{
for (int i= PVindDeb; i<= PVinDFin; i++)
    {
      RTnote[i]=0;
    }
}
// fonction fAjouter : ajouter une note au tableau
// *********************************************
// Paramètres :
//    reference : RTnote[] : entier, tableau de notes
//    reference : RVindLibre : entier, position libre
//    valeur  : PVindFin : entier, indice dernier element
//    valeur  : PVnote : entier, note à ajouter au tableau
// retour :
//    logique : VRAI = ajout OK, FAUX = erreur ajout
//
bool fAjouter(int RTnote[], int & RVindLibre, int PIND_FIN, int
PVnote)
{
if (RVindLibre > PVindFin)
    {
    return false ;
    }
    else
    {
       RTnote[RVindLibre]=PVnote ;
       RVindLibre++ ;
       return true ;
    }
}
```

2. Passage d'un tableau et PROTECTION DES VALEURS

Le mot-clef **const** permet la **protection d'une variable** tableau **passée par référence** (= éviter que le sous-programme puisse le modifier).

Exemple de syntaxe de déclaration :

```
void pNom_procedure (const int RTnote[], int PVindFin)
{
  // Corps de la procédure : déclaration et instructions
  // « lecture » de RTnote en contrôlant qu'on n'utilise pas un indice hors des limites du tableau
}
```

- **RTnote** : va recevoir une référence vers le tableau qui sera passé en argument.
- **PVindFin** : on doit associer une variable qui va contenir l'indice du dernier élément ou bien le nombre d'éléments du tableau

Exemple de syntaxe d'appel (identique au précédent):

```
const int NB_ELEM = 20;
const int IND_FIN = NB_ELEM - 1;
int Tnote[NB_ELEM];

pNomProcedure(Tnote, IND_FIN) ;
```

```
// procédure pAfficher : afficher les noms
// *******************************************
// Paramètres :
//    reference constante : RTnote[] : entier, tableau des notes
//    valeur : PVindDeb : entier, indice de debut
//    valeur : PVindLibre : entier, indice position libre
//
void pAfficher(const int RTnote[], int PVindDeb, int PVindLibre)
{
   cout << endl << "==== LISTE DES NOMS ====" << endl ;
    if (PVindLibre > PVindDeb)
        {
          for(int i=PVindDeb ; i<PVindLibre ;i++)
            cout << RTnote[i] << endl ;
        }
      else
        {
          cout << "...LISTE VIDE..." << endl ;
        }
   cout << "==== FIN DE LISTE ====" << endl ;
}
```

3. Passage de tableaux multidimensionnels

Lorsqu'on doit utiliser des tableaux ayant plus d'une dimension, il faut déclarer le paramètre tableau avec un nombre de crochets, « [ ] », égal à sa dimension. **Le premier crochet peut demeurer vide, mais les suivants doivent obligatoirement contenir le nombre d'élément** correspondant à chacune des dimensions.

Exemple de syntaxe de déclaration :

```
void pNom_procedure (int RTnote[][10], int PVindFin1)
{
  // Corps de la procédure : déclaration et instructions
  // modification de RTnote en contrôlant qu'on n'utilise pas des indices hors des limites du tableau
}
```

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

// prototype
void pInitialiser(int RTnote[][10] , int PIND_DEB1, int PIND_FIN1, int PIND_DEB2, int PIND_FIN2);

int main ()
{
    // Constantes et variables de gestion du tableau
    const int NB_ELEM1 = 20 ; // dimension 1
    const int IND_DEB1 = 0 ;
    const int IND_FIN1 = NB_ELEM1 – 1 ;
    const int NB_ELEM2 = 10 ; // dimension 2
    const int IND_DEB2 = 0 ;
    const int IND_FIN2 = NB_ELEM2 – 1 ;
    int  Tnote[NB_ELEM1][NB_ELEM2];

    pInitialiser(Tnote, IND_DEB1, IND_FIN1, IND_DEB2, IND_FIN2) ;

    return EXIT_SUCCESS; // constante fournie par C++, vaut 0
}
// procédure pInitialiser : initialiser le tableau à 2 dimensions
// ***********************************************
// Paramètres :
//    reference : RTnote[][10] : entier, tableau des notes
//    valeur : PVindDeb1 : entier, indice premier élément dim1
//    valeur : PVindFin1 : entier, indice dernier élément dim1
//    valeur : PVindDeb2 : entier, indice premier élément dim2
//    valeur : PVindFin2 : entier, indice dernier élément dim2
//
void pInitialiser(int RTnote[][10] , int PVindDeb1, int PVindFin1, int PVindDeb2, int PVindFin2)
{
for (int i= PVindDeb1 ; i<= PVindFin1 ; i++)
    {
      for (int j= PVindDeb2, j<= PVindFin2 ; j++)
         {
            RTnote[i][j]=0;
            cout << i << "-" << j << endl; // pour voir...
          }
    }
} // fin pInitialiser
```

X. Exercices

cf ALGO_ch5_ssprog
Coder les algorithmes des exercices 1, 2 et 3 en c++.

XI. Exercices : Proposition de corrigé

Exercice 1 :

Programme C++

```
#include <iostream>
#include <string>
using namespace std ;
// prototypes de fonctions
void pAfficherIntro() ;
string fDemanderSaisirUnNom () ;
void pAfficherAccueil (string PVnom ) ;

int main()
{
  string Vnom ;
  int i ;
  pAfficherIntro() ;
  for(i=1 ;i<=10 ;i++)
  {
    Vnom = fDemanderSaisirUnNom () ;
    pAfficherAccueil(Vnom) ;
  }
  return 0 ;
}
```

```
// pAfficherIntro %
// Objectif : afficher le message d'intro
void pAfficherIntro()
{
  cout << "Bonjour";
  cout << "Accuellir les étudiants";
}
```

```
// fDemanderSaisirUnNom
// Objectif : demander la saisie d'un nom
// Retourne le nom saisi
string fDemanderSaisirUnNom ()
{
  string VnomSaisi ;
  cout << "Entrez un nom : " ;
  cin >> VnomSaisi ;
  return VnomSaisi ;
}
```

```
// pAfficherAccueil
// Objectif : afficher le message d'accueil pour un nom
void pAfficherAccueil (string PVnom )
{
  cout << "bonjour " << PVnom ;
  cout << "bienvenu !" ;
}
```

Exercice 2 :

Programme C++

```
#include <iostream>
#include <string>
using namespace std ;
// prototypes de fonctions
int fSaisirNombre ();
void pPermuter (int & refPVnb1, int & refPVnb2) ;
void pAfficher (int Pnb1, int Pnb2) ;


int main()
{
int Vnombre1, Vnombre2 ;
  // demander la saisie d'un nombre et l'affecter à Vnombre1 %
  Vnombre1 = fSaisirNombre() ;

  // demander la saisie d'un nombre et l'affecter à Vnombre2%
  Vnombre2 = fSaisirNombre() ;

  // Permuter 2 nombres %
  pPermuter(Vnombre1, Vnombre2) ;

  // Afficher les 2 nombres %
  pAfficher(Vnombre1, Vnombre2) ;

  return 0 ;
}
```

```
// fSaisirNombre
// Objectif : saisir un nombre
// Retourne le nombre saisi
int fSaisirNombre ()
{
  int VnbSaisi ;
  cout << "Entrez un nombre: " ;
  cin >> VnbSaisi ;
  return VnbSaisi ;
}
```

```
// pPermuter
// Objectif : permuter la valeur de 2 nombres passés en arguments
void pPermuter (int & refPVnb1, int & refPVnb2)
{
  int Vtemp ;
  Vtemp = refPVnb1 ;
  refPVnb1 = refPVnb2 ;
  refPVnb2 = Vtemp ;
}
```

```
// pAfficher
// Objectif : afficher la valeur de 2 nombres passés en arguments
void pAfficher (int Pnb1, int Pnb2)
{
  cout << "nombre 1 " << Pnb1 ;
  cout << "nombre 2 " << Pnb2 ;
}
```

Exercice 3 :

Programme C++

```
// Analyser
#include <iostream>
#include <string>
using namespace std ;

// prototypes
void pInitialiser(int & refPTnombre[], int Pdeb , int Pfin ) ;

void pSaisir1(int refPTnombre[], int PVindice) ;
void pSaisir2(int & refPVelem) ;

int fSaisir3 () ;
void pAfficher(const int refPTnombre[], int PVdeb , int PVfin ) ;

int fCalculerSomme (const int refPTnombre[], int PVdeb , int PVfin ) ;

const int NB_ELEM  = 20 ;
const int IND_DEB  = 0 ; // modif
const int IND_FIN  = NB_ELEM – 1 ; //
int Tnombre[NB_ELEM] ;
int VindLibre  = IND_DEB ;
int Vsomme, i ;
int main ()
{
  // Initialiser le tableau
  pInitialiser(Tnombre, IND_DEB, IND_FIN) ;
  for( i=IND_DEB ;i<=IND_FIN ;i++)
  {
  // demander saisie de la valeur i
  pSaisir1(Tnombre, i) ;
  //ou bien
  pSaisir2(Tnombre[i]) ;
  //ou bien
  Tnombre[i]=fSaisir3() ;
  }

  // Afficher le contenu
  pAfficher(Tnombre, IND_DEB, IND_FIN) ;
  // récupérer la somme
  Vsommme = fCalculerSomme(Tnombre, IND_DEB, IND_FIN)
  Cout << "Somme =" << Vsomme ;
}
```

En c++, les declaratopn de²param tableaux sont automatiquement passées par référence : pa de & dabs ce²cas

```
void pInitialiser(int refPTnombre[], int PVdeb , int PVfin )
{
  int i ;
  for(i=PVdeb ; i<=PVfin ;i++)
  {
    refPTnombre[i] = 0 ;
  }
}
```

```
void pSaisir1(int refPTnombre[], int PVindice)
{
  cout << "Entrez un nombre(1): " ;
  cin >> refPTnombre[PVindice ] ;
}
```

```
void pSaisir2(int & refPVelem)
{
```

```
  cout << "Entrez un nombre(2): " ;
  cin >> refPVelem ;
}
```

```
int fSaisir3 ()
{  int VnbSaisi ;
  cout << "Entrez un nombre(3): " ;
  cin >> VnbSaisi ;
  return VnbSaisi ;
}
```

```
void pAfficher(const int refPTnombre[], int PVdeb , int PVfin )
{
  int i ;
  for(i=PVdeb ; i<=PVfin ;i++)
  {
    cout << refPTnombre[i] ;
  }
}
```

```
int fCalculerSomme (const int refPTnombre[], int PVdeb , int PVfin )
{
  int Vsomme, i ;
  Vsomme = 0 ;
  for(i=PVdeb ; i<=PVfin ;i++)
  {
   Vsomme = (Vsomme + refPTnombre[i]);
  }
  return Vsomme ;
}
```