Introduction au langage C++

Roger D. Hersch

Laboratoire de Systèmes Périphériques (LSP)
Faculté Informatique & Communications
EPFL

Introduction au C++

Programmation orientée objets:

- Les objets comprennent données et méthodes (fonctions) agissant sur ces données.
- Les méthodes sont rendues visibles à l'extérieur (public) à travers leurs prototypes.
- L'amélioration des méthodes n'a pas d'effet sur l'extérieur.
- Les membres privés de la classe sont protégés
- Le cloisonnement entre objets rend les programmes plus sûrs et plus facile à maintenir.
- Code de création et de destruction de l'objet.

Remerciements: Jean-Cédric Chappelier, pour les nombreux exemples repris de son cours C++,

Réference: C++ par la pratique, J.C. Chappelier, F. Seydoux, PPUR, 2004

Variables et entrées-sorties en C++

Solution d'une éqution quadratique:

```
#include <iostream>                    //quadraticEq.cpp
#include <cmath>
using namespace std; // name scope, alternative std::appel

int main() {
double a(0.0);       //déclaration et initialisation de variables
double b(0.0);       // b=0.0 egalement possible
double c(0.0);
double delta(0.0);
cout << "entrez coeff. a, b, c de l'eq. quadratique :" <<endl;
cin>>a>>b>>c;        //entrée des nombres
delta=b*b-4*a*c;
if (delta < 0.0) {
cout<< "pas de solutions reelles" << endl;
} else if (delta == 0.0) {
          cout<< "une solution unique : " << -b/2.0*a << endl;
      } else { cout << "deux solutions: "
                    << (-b-sqrt(delta))/2.0*a << " et "
                    << (-b+sqrt(delta))/2.0*a << endl;
             }
}
```

Exemple d'exécution

Console:

```
> eq_quadratique
> entrez les coeff. a, b, c de l'eq. quadratique :
4
13
5
deux solutions: -44.868 et -7.13204

> eq_quadratique
> entrez les coeff. a, b, c de l'eq. quadratique :
1
4
4
une solution unique : -2
```

La surcharge de fonctions

En C++, il est possible de définir plusieurs fonctions de même nom si ces fonctions n'ont pas les mêmes listes d'arguments: le nombre ou le types d'arguments différents. Très utile pour créer des interfaces souples et étendre les programmes sans devoir multiplier les noms de fonctions.

Exemple:

```
void affiche (int x) {
cout << "entier : " << x << endl;
{
void affiche (double x) {
cout << "reel: " << x << endl;
}
void affiche (int x1, int x2) {
cout << "couple: " << x1 << x2 << endl;
}
```

`affiche(1)`, `affiche(1.0)` et `affiche(1,1)` produisent des affichages s'appliquant à différentes séquences de passages de paramètres

Un exemple complet de classe

```
#include <iostream> //rect.cpp
using namespace std;
class Rectangle{
private:              // accès privé à ce type de classe
double hauteur;       // données seulement accessibles
double largeur;       // par méthodes de la classe
                      // this: pointeur à l'objet
public:
Rectangle (double haut, double large)
   {hauteur = haut; largeur = large;}    //constructeur
~Rectangle() {};                  // destructeur ne fait rien
//définition des méthodes
double surface() const { return (hauteur * largeur); }
double getHauteur() const { return hauteur; }  // const:
double getLargeur() const { return largeur; }  // pas modif
void setHauteur(double hauteur) {this->hauteur = hauteur;}
void setLargeur(double largeur) {this->largeur = largeur;}
};
```

Explication: const protège l'objet

Utilisation de la classe

```
void main() {
Rectangle rect(10,20);
//appel au constructeur lors de l'instanciation
cout<<"surface initiale= "<<rect.surface()<<endl;
double lu;
cout << "Quelle hauteur? "; cin >> lu;
   rect.setHauteur(lu);
cout << "Quelle largeur? "; cin >> lu;
   rect.setLargeur(lu);
cout << "Surface = " << rect.surface() << endl;
}
```

Console:

```
X:\prog3\Debug> rectangle
surface initiale= 200
Quelle hauteur? 15
Quelle largeur? 15
Surface = 225
```

Définition extérieure des méthodes (1)

```
class Rectangle  // rect1.cpp
{private: // attribus privés
double hauteur; double largeur;

public: // prototypes seulement
Rectangle(double haut=0.0, double large=0.0);
~Rectangle(){}; // destructeur complet
double getHauteur() const;
double getLargeur() const;
void setHauteur(double hauteur);
void setLargeur(double largeur);
double surface() const;
};
```

Explication: haut=0.0, large=0.0 paramètres par défaut => engendrent différents constructeurs

Définition extérieure des méthodes (2)

```
Rectangle::Rectangle (double haut, double large)
{hauteur = haut; largeur = large;} //constructeur
double Rectangle::surface()
{ return (hauteur * largeur); }
double Rectangle::getHauteur() const
{ return hauteur; }
double Rectangle::getLargeur() const
{ return largeur; }
void Rectangle::setHauteur(double hauteur)
{this->hauteur = hauteur;}
void Rectangle::setLargeur(double largeur)
{this->largeur = largeur;}
```

Surcharge d'opérateurs

```
#include <iostream>     //Complex.cpp
#include <cmath>         // surcharge ( +, -, *, /, = )
using namespace std;    //pour agir sur objets.

class Complex
{ private: double r, i;
  public:
   Complex(const double rI, const double iI) //constructeur
       { r = rI; i = iI; }
   Complex() {};
   double real() const{return r;}   //retourne partie réelle
   double imaginary() const{return i;}//retourne partie im.
   double sqNorm() const{return r*r + i*i; }
   double magnitude() const {return sqrt(sqNorm()); }
   double argument() const {return atan2(i,r); } // angle
//opérateurs à une opérande
   Complex operator -()const {return Complex(-r, -i); }
   Complex conjugate() const {return Complex(r, -i); }
```

Surcharge d'opérateurs binaires

```
//opérateurs à 2 opérandes; & b: passage par réf.
  Complex operator + (const Complex & b)
  {return Complex(r+b.real(), i+b.imaginary());
  }
  Complex operator + (const double b)
  {   return Complex(r+b, i);
  }
  Complex operator - (const Complex & b)
  {   return Complex(this->real()-b.r,
                     this->imaginary()-b.i);
  }
  Complex operator - (const double b)
  {   return Complex(this->real()- b,                this-
  >imaginary());
  }
```

Explication: attributs privés de l'objet b peuvent être accédés dans la classe Complex

Surcharge d'opérateurs binaires

```
  Complex operator * (const Complex & b)
  { return Complex(r * b.real() - i * b.imaginary(),
                   r * b.imaginary() + i * b.real());
  }
  Complex operator * (const double b)
  { return Complex(r * b, i * b);
  }
     // a/b=((a.r+a.i)(b.r-b.i))/(b.r+b.i)*(b.r-b.i)
  Complex operator /(const Complex & b)
  {
  double realPart = r*b.real() + i*b.imaginary();
  double imaginaryPart = i*b.real() - r*b.imaginary();
  double divisor = b.sqNorm();
  return Complex(realPart/divisor,imaginaryPart/divisor);
  }
  Complex operator / (const double b)
  {   return Complex(r/b, i/b);
  }
};
```

Vérification classe "Complex"

```
    //affiche les nombres complexes dans la console
void dispC(const Complex & a )
{   printf("(%f, %f)\n", a.real(), a.imaginary());
}
void main()
{
  Complex comp1(10,20); Complex comp2(1, 4); Complex res(0, 0);
  cout << "partie reelle = " << comp1.real() << endl;
  cout << "partie imaginaire = " << comp1.imaginary() << endl; printf("\n");
  res = comp1 + comp2; dispC(comp1); printf(" + "); dispC(comp2);
                                    printf(" = "); dispC(res); printf("\n");
  res = comp1 - comp2; dispC(comp1); printf(" - "); dispC(comp2);
                                   printf(" = "); dispC(res); printf("\n");
  res = comp1 * comp2; dispC(comp1);          printf(" * "); dispC(comp2);
                                   printf(" = "); dispC(res); printf("\n");
  res = comp1 / comp2; dispC(comp1); printf(" / "); dispC(comp2);
                                  printf(" = );  dispC(res); printf("\n");
}
```

Soustraction “double” – “Complex” ?

(a) Soustraction Complex moins double:

```
Complex operator - (const double b)

Complex b(20.0,10.0); double a=10;
res=b-a; dispC(res); printf("\n");
```

(b) Comment effectuer "double" - "Complex" ? Constructeur:

```
  Complex(const double rI, const double iI=0)
void main()
{Complex comp1(10,20);  double nbReel=10;
 Complex res0 = Complex(nbReel)-comp1;//que se passe-t-il ?
 cout << "nbReel - nbComplex = "<< nbReel << " - " ;
 dispC(comp1); printf(" = "); dispC(res0); printf("\n");
}
```

Console:

```
nbReel - nbComplex = 10 - (10.000000, 20.000000) = (0.000000, -20.000000)
```

Soustraction “double” – “Complex” ?

(c) opérateur en dehors de la classe

dans classe Complex: friend nécessaire !

```
friend Complex operator - (double a, const Complex & b);
```

en dehors de la classe : opérateur à deux opérandes explicites

```
Complex operator - (double a, const Complex & b)
{ return Complex(a-b.r,-b.i);}
```

```
void main() // Complex1.cpp
{Complex comp1(10,20);  double nbReel=10;
 Complex res1 = nbReel- comp1;//que se passe-t'il ?
 cout << "nbReel - nbComplex = "<< nbReel << " - " ;
 dispC(comp1); printf(" = "); dispC(res0); printf("\n");
}
```

Console:

```
nbReel - nbComplex = 10 - (10.000000, 20.000000) = (0.000000, -20.000000)
```

Copie d'objets

C++ vous permet de copier un objet à l'aide du constructeur de copie.

Syntaxe du constructeur:

```
NomClasse(const NomClasse & obj){…}
```

Par exemple pour la classe Rectangle :

```
Rectangle(const Rectangle & obj):
  hauteur(obj.hauteur), largeur(obj.largeur){}
```

Utilisation:

```
Rectangle r1( 12.3, 24.5 );
Rectangle r2( r1 );
     //crée un rectangle r2 identique à r1.
```

Copie d'objet Rectangle

```
#include <iostream>    //rect2.cpp
using namespace std;
class Rectangle{
private:
double hauteur;// données seulement accessibles
double largeur;// par méthodes de la classe
                       // this: pointeur à l'objet
public:
Rectangle (double haut=1, double large=1):  //parametres défaut
   hauteur(haut),largeur(large) {} //constructeur, const

Rectangle Rectangle(const Rectangle & obj): //constr. copie
   hauteur(obj.hauteur), largeur(obj.largeur){}
// alternative:       {*this=obj;} , voir rect2alt.cpp

~Rectangle() {};                   // destructeur ne fait rien
//définition des méthodes
double surface() const { return (hauteur * largeur); }
double getHauteur() const { return hauteur; }
double getLargeur() const { return largeur; }
void setHauteur(double hauteur) {this->hauteur = hauteur;}
void setLargeur(double largeur) {this->largeur = largeur;}
};
```

Vérification copie d'objets

```
int main()
{
Rectangle rect(10,20);
Rectangle rect1;
Rectangle rect2(rect);
//appel au constructeur lors de l'instanciation
cout<<"surface initiale=" <<rect1.surface()<<endl;
cout << "Surface copie = " << rect2.surface() << endl;
}
```

Console

```
surface initiale= 1
Surface copie = 200
```

Pile de chaînes de caractères (i)

```
class Stack
{ private :
   EnclString* top;  // top of stack
   EnclString* base; // base address of stack
   int stacksize;
  public:
     Stack(int s):stacksize(s) { top=base=new EnclString[s];}
     ~Stack() {delete[] base;}
     void push(EnclString & item) {*top++ = item;} // attention:
                                                   //no error check
      EnclString pop(){return *--top; }
     int nbItems() {return top-base;}
};
```

Quel type peut on utiliser pour EnclString ? Une chaîne de caractères ?

```
#define STRLENGTH 80
typedef char EnclString[STRLENGTH];
// utilisation
EnclString myString; //equivalent a: char myString[STRLENGTH]
```

Alternative ?

Pile de chaînes de caractères (ii)

```
#include <stdio.h>  // RDH 22Nov05 stackString.cpp
#include <string>
#define STRLENGTH 80
typedef char myStringT[STRLENGTH];

class EnclString // encloses an array of chars into a class
{  // makes copy possible by default = operator definition
private:
   myStringT string;
public:
   EnclString(){string[0]=0;}              //default constructor
   EnclString(const myStringT inString) // string constr.;
   {    int i=0;
        while (inString[i]!= 0) { string[i]=inString[i]; i++; }
        string[i]=0;
   }

   void getString(myStringT outString)
   {    for (int i=0;i<STRLENGTH;i++) outString[i]=string[i];
   }
};
```

Pile de chaînes de caractères (iii)

```
int main(void)          // stack of strings
{
 char seps[]   = " ,\t\n";  // separateurs: espace, tab, retour chariot
 char * ptrToken;
 Stack stackOfStrings(64);
 myStringT myString, outString;
 printf("Entrez chaines de caracteres a mettre sur la pile \n");
 fgets(myString, 80, stdin);          // reads lines of strings
 ptrToken = strtok( myString, seps ); // pointer to first strimg
 while( ptrToken != NULL )
 { EnclString myEnclStr(ptrToken);  // encloses string into class
   stackOfStrings.push(myEnclStr);  // push the string on the stack
   ptrToken = strtok( NULL, seps ); // pointer to next string
 }
 printf("\n affiche contenu pile: \n\n"); // print content of stack
 while (stackOfStrings.nbItems()>0)
 { EnclString myEnclStr(stackOfStrings.pop()); // uses default constructor
   myEnclStr.getString(outString);
   printf(outString); printf("\n");
 }
 return 0;
}
```

Membre statique d'une classe

Il est possible de définir un élément commun à toutes les instances d'une classe :

```
#include <iostream> // refCount.cpp RDH 26.9.05
using namespace std;
class MyList{
private:
   int myVar;
   static int refCount;
public:
   MyList * ptrNext;
   MyList(int currVar):myVar(currVar){MyList::refCount++;}
   ~MyList(){refCount--;}
   addList(const MyList* & ptrNewEl){ptrNext=ptrNewEl;}
   int getRefCount()  {return refCount;}
};
// variable statiques à déclarer hors de la classe
int MyList::refCount; //espace alloué pour cette variable
```

Membre statique d'une classe (suite)

```
void main(){    // créer 3 éléments de la liste numérotés 1,2,3
                // détruire les éléments 1 à 1 et afficher elCount
MyList* myHeadPtr = new MyList(1);
MyList* currPtr = myHeadPtr; // current pointer is myHeadPtr

for(int i=2;i<4;++i) { // construction de la liste
  MyList * elPtr= new MyList(i);
  currPtr->addList(elPtr);
  currPtr=elPtr;
  cout<< "num elem: "<< i<<"refCount: "<< \
                          myHeadPtr->getRefCount()<< endl;
 }
currPtr = myHeadPtr;
MyList * tempPtr;
for(int j=1;j<4;++j) { // destruction de la liste
   tempPtr=currPtr;
   currPtr=tempPtr->ptrNext;
   delete tempPtr;
   cout<<"num detruit: "<<j<<" refCount: " << \
                          myHeadPtr->getRefCount()<<endl;
  }
}
```

Exemple d'exécution

Console:

```
> statMember
num elem: 2 refCount: 2
num elem: 3 refCount: 3
num detruit: 1 refCount: 2
num detruit: 2 refCount: 1
num detruit: 3 refCount: 0
```

Exercice

Est-ce qu'une classe est passée par copie ou par référence à une fonction?

Exemple:

```
void affVar( MyClass mc)
```

Créer un programme qui donne la réponse !

Tentative de solution

```
#include <iostream>     //paramClassDir.cpp :
using namespace std;    // tester le passage d'une classe
class MyClass{          // comme paramètre à une fonction
public:
   static int newCount, delCount;
   int myVar;
   MyClass(){newCount++;}
   MyClass(int initVar):myVar(initVar) { newCount++; }
   ~MyClass(){delCount++;}
   static int getNewCount() {return newCount;};
   static int getDelCount() {return delCount;};
};
int MyClass::newCount=0; int MyClass::delCount=0;
void affVar(MyClass  mc) {
cout << "affDeb newCount " << MyClass::getNewCount()
   << " delCount " << MyClass::getDelCount()
   << " mc.myVar = " << mc.myVar << endl << endl;
}
```

```
void main(){      // affiche newCount et delCount
MyClass mcl(5);
cout<<"start newCount "<< MyClass::getNewCount()
  <<“delCount “ << MyClass::getDelCount()
  << " mc.myVar = " << mcl.myVar << endl;
affVar(mcl);
cout<<"end newCount " << MyClass::getNewCount()
  <<" delCount “  << MyClass::getDelCount()<< endl;
}
```

Console:

```
Start  newCount 1 delCount 0 mc.myVar = 5
affDeb newCount 1 delCount 0 mc.myVar = 5
end    newCount 1 delCount 1
```

Question: que se passe-t-il ? Pourquoi y a-il une instanciation et une désinstanciation de l’objet, alors qu’il est présent à la fin du programme ?

Exemple d'exécution

Explication

```
void affVar(MyClass  mc)
```

L'appel à la fonction `affVar` crée un nouvel object et copie `mc` dans le nouvel object. Vu que le constructeur de copie manque, il n'y a pas de mise à jour de `newCount` mais la copie de l'objet est détruite à la sortie de `affVar`.

Rajouter constructeur

```
// MyClass(const MyClass & mc):
            myVar(mc.myVar){newCount++;}
```

Assignation: exercice

Lors d'une copie d'une variable instance de classe,

par ncl=mcl, est-ce que :

(a) les variables internes à la classe sont copiées?

(b) les variables pointées par des pointeurs qui sont dans la classe sont copiées (copie profonde) ?

Créer un programme qui donne la réponse !

Essai de solution

```
#include <iostream> // passParamPtr.cpp
// tester la copie de classe avec = et par passage
// de parametre, ce programme plante, pourquoi ?

class MyClass
{ public:
int * varPtr;    // init. variable pointée par varPtr
MyClass(int initVar):varPtr(new int(initVar)){}
MyClass(const MyClass & mc)):
  varPtr(new int(*(mc.varPtr))){} // initialise int
                                  //  avec *(mc.varPtr)
~MyClass(){delete varPtr;}
};

void affVar(MyClass mc)
{ cout << "aff: *mc.varPtr=" << *mc.varPtr << "
  mc.varPtr=" << mc.varPtr <<endl;
}
```

```
void main() {
// but: copie d'instances de classes
MyClass mcl(5);
MyClass ncl(3);
cout <<"avant: *mcl.varPtr=" << *mcl.varPtr << " *ncl.varPtr=" << *ncl.varPtr
    << " mcl.varPtr=" << mcl.varPtr <<" ncl.varPtr=" << ncl.varPtr <<endl;

affVar(ncl);

ncl=mcl; cout <<" ncl=mcl " << endl;

cout << "apres *mcl.varPtr=" <<*mcl.varPtr <<" *ncl.varPtr=" << *ncl.varPtr \
     << " mcl.varPtr=" << mcl.varPtr <<" ncl.varPtr=" << ncl.varPtr <<endl;
}
```

Console: Est-ce que ce programme est correct ? Comment l'améliorer ?

```
avant: *mcl.varPtr=5          *ncl.varPtr=3
        mcl.varPtr=0x00322270 ncl.varPtr=0x003222A8
aff:   *mc.varPtr=3 mc.varPtr=0x00322B70
 ncl=mcl
apres *mcl.varPtr=5          *ncl.varPtr=5
       mcl.varPtr=0x00322270   ncl.varPtr=0x00322270
```

Solution: surcharger operator=

```
#include <iostream>  //passParamPtrEqu.cpp
// tester la copie de classe avec = et par passasge de
   parametre

class MyClass{
public:
static int newCount, delCount;
int * varPtr;
MyClass(int initVar):varPtr(new int(initVar)){}
MyClass(const MyClass & mc)):
   varPtr(new int(*(mc.varPtr))){} // initialise int
                                   //avec *(mc.varPtr)
~MyClass(){ delete varPtr;}
MyClass operator=(const MyClass & mc) //surcharge de =
   {*varPtr=*(mc.varPtr); return *this;} //copie profonde
};
```

Exemple d'exécution

Console:

```
>passParamPtrEq

avant: *mcl.varPtr=5              *ncl.varPtr=3
        mcl.varPtr=0x00322270   ncl.varPtr=0x003222A8

aff:   *mc.varPtr=3
        mc.varPtr=0x00322B70

ncl=mcl

apres *mcl.varPtr=5             *ncl.varPtr=5
      mcl.varPtr=0x00322270    ncl.varPtr=0x003222A8
```

Héritage

L'héritage permet d'étendre une classe existante, c'est à dire rajouter des membres et des fonctions, tout en bénéficiant de la classe de base.
Les méthodes de la classe dérivée masquent les méthodes de la classe de base.

```
#include <iostream>
using namespace std;      // rect3D.cpp

class Rectangle {         //classe de base
protected: // permettre accès que par les
           // classes dérivées
double largeur, hauteur;
// le reste de la classe et constr. Ici ..
public:
double surface() {return (largeur * hauteur);}
};
```

Héritage (2)

```
class Rectangle3D : public Rectangle {
protected:
double profondeur;
public:
// les constructeurs seraient ici...
double surface() {
return (2.0*(largeur*hauteur) + 2.0*(largeur*profondeur) +
                                        2.0*(hauteur*profondeur));}
// le reste de la classe...
};
```

Pour appliquer soit la méthode de la classe de base, soit celle de la classe dérivée.

```
class Rectangle3D : public Rectangle {
// ... constructeurs, attributs comme avant
double surface() {
if (profondeur == 0)
   return Rectangle::surface();
else
   return (2.0*(largeur*hauteur) + 2.0*(largeur*profondeur) +
           2.0*(hauteur*profondeur));
}
};
```

Constructeur de la classe dérivée

```
class Rectangle {
protected: // accessible par classes dérivées
double largeur, hauteur;
public:
Rectangle(double l, double h) : largeur(l), hauteur(h){ }
// le reste de la classe ...
};

class Rectangle3D : public Rectangle {
protected:
double profondeur;
public:
Rectangle3D(double l, double h, double p) :
                Rectangle(h, l), profondeur(p) { }
// le reste de la classe...
};
```

Si pas nécessaire, on peut ne pas invoquer le constructeur de la classe de base.

Test

```
int main() {
int const largeur = 10, hauteur = 20, profondeur = 5;
Rectangle3D rect3D(largeur,hauteur,profondeur);
//appel au constructeur lors de l'instanciation
cout<<"taille rectangle 3D: largeur = "<<largeur << "
  hauteur=" << hauteur \
  << " profondeur=" << profondeur << endl;
cout << "surface totale = " << rect3D.surface()<<endl;
return 0;
}
```

Que se passe-t-il lors de

```
rect3D(largeur,hauteur,profondeur);
```

```
rect3D.surface()
```

Console

```
>taille rectangle 3D: largeur = 10 hauteur=20 profondeur=5
>surface totale = 700
```

Classes abstraites

Les classes abstraites sont des classes qui déclarent des méthodes sans les implémenter. Ces méthodes sont implémentées par les sous-classes. Cette technique est généralement utilisée pour définir une interface qui devra être implémentée par toutes les sous-classes, et ainsi rendre possible le polymorphisme. Exemple:

```
class Forme { public:
   virtual double surface() = 0; //fonction virtuelle pure
};                                //virtual = : define method in subclass

class Rectangle : public Forme {
private: double hauteur, longueur;
public:
   Rectangle(double h,double l) { hauteur=h; longueur=l; }
   double surface() { return hauteur*longueur; }
};

class Cercle : public Forme {
private: double rayon;
public:
   Cercle(double r) { rayon=r; }
   double surface() { return 3.14*rayon*rayon; }
};
```

Classes abstraites (2)

```
int main()
{
   // Forme tab[5]; impossible car Forme est abstraite
   Forme *tab[5];
   tab[0] = new Rectangle(3,4);
   tab[1] = new Cercle(1);
   int i;
   for(i=0;i<2;++i)
       cout << i << ": " << tab[i]->surface() << endl;
}
```

```
Console:
X:\prog3\Debug> polymorphism
0: 12
1: 3.14
```

Méthodes virtuelles

Lorsqu'on appelle une méthode sur un objet, c'est l'implémentation du type actuel de l'objet qui est utilisée, par exemple la super-classe.
En spécifiant une méthode comme *virtuelle*, la méthode correspondante de la classe dérivée est appelée. Les méthodes d' une classe abstraite dont l'implémentation n'existe pas doivent être virtuelles.

```
class Cercle {
public: double rayon;
   Cercle(double r) { rayon=r; }
   void affiche() { cout << "Rayon " << rayon << endl; }
};

class CercleNomme : public Cercle {
public: char *nom;
   CercleNomme(double r,char *nom) : Cercle(r) { this->nom=nom; }
   void affiche() { cout << "Cercle " << nom << endl; }
};

int main() {
   Cercle *c = new CercleNomme(3,"c1");
   c->affiche(); // Affiche "Rayon 3":
   delete c;     // méthode provient de la classe de l'objet
}
```

Méthodes virtuelles (2)

On peut appeler la méthode de la super-classe en spécifiant explicitement le nom de la classe

```
class Cercle {
public: double rayon;
   Cercle(double r) { rayon=r; }
   virtual void affiche() { cout << "Rayon " << rayon << endl; }
};

class CercleNomme : public Cercle {
public: char *nom;
   CercleNomme(double r,char *nom) : Cercle(r) { this->nom=nom; }
   void affiche() {
        cout << "Cercle " << nom << ": ";
        Cercle::affiche();
   }
};

int main() {
   Cercle *c = new CercleNomme(3,"c1");
   c->affiche(); // Affiche "Cercle c1: Rayon 3"
   delete c;
}
```

Méthodes virtuelles (3)

Il est fortement recommandé de rendre virtuels tous les destructeurs

```
#include <iostream>  // rectMethVirt2.cpp
using namespace std;
class Cercle {
public: double rayon;
   Cercle(double r) { rayon=r; }
};

class CercleNomme : public Cercle {
public: char *nom; // Stocke une copie du nom
   CercleNomme(double r,char *nom) : Cercle(r) {
        this->nom = (char *) malloc((strlen(nom)+1)*sizeof(char));
        strncpy(this->nom,nom,strlen(nom)+1); }
   ~CercleNomme() { free(nom); printf("memoire liberee \n"); }
};

int main() {
   Cercle *c = new CercleNomme(3,"c1");
   delete c;  // ce programme n'affiche rien !Pourquoi ?
}
```

Correction: doter Cercle d'un destructeur virtuel `// rectMethVirt3.cpp`

```
    virtual ~Cercle() { }
```

Récursivité

Plus grand commun dénominateur (PGCD) de deux nombres

```
#include <stdio.h>  // pgcd.c
#include <stdlib.h>

void main(){
  int t, u, v;
  printf("Entrez deux nombres séparés par un espace:
  \n");
  scanf("%d %d",&u, &v);
  if (u < v) t = u; else t = v;

  // tant que les 2 restes ne sont pas zero
  while ((u % t) != 0 || (v % t) != 0)     t = t-1;

     printf("Le PGCD de %d et %d est %d", u, v, t);
}
```

Algorithme d'Euclide

a > b **// GCD: greatest commun denominator**

GCD (a, b) = GCD (a, a-b)

GCD (a, a-2b)

...

g= CD(a,b)
a/g – b/g -> entier
⇒ (a-b)/g -> entier
⇒ (a-2b)/g -> entier
⇒ g = CD(a,a mod b)

donc:

GCD (a, b) = GCD (a, a mod b)

Si (a mod b = 0)

Alors b = GCD (a, b);

Sinon poursuivre avec

GCD (b, a mod b);

Récursivité

```
#include <stdio.h>  //pgcdEuclide.c
#include <stdlib.h>

int gcd(int u, int v){
   int result;
   if ( v == 0 )                    //reste précédent zero
       result = u;
   else
       result = gcd(v, u % v); //appel récursif
   return result;
}

int main(){
   int u, v;
   printf("Entrez deux nombres séparés par un espace  \n");
   scanf("%d %d",&u, &v);
   printf("Le PGCD de %d et %d est %d \n", u, v, gcd(u,v));
}
```

Algorithme d'Euclide sans récursivité

```
#include <stdio.h>                    //pgcdNonRecursif.c
#include <stdlib.h>

int gcd(int u, int v){
   int t;
   while(v != 0){
       t = u % v;   u = v;   v = t;
   }
   return u;


int main(){
   int u, v;
   printf("Entrez deux nombres séparés par un espace \n");
   scanf("%d %d",&u, &v);
   printf("Le PGCD de %d et %d est %d \n", u, v, gcd(u,v));
}
```

Exemples d'exécutions

Console: `> pgcd`

```
Entrez un premier nombre:
2541
Entrez un deuxième nombre:
282
Le PGCD de 2541 et 282 est 3

> pgcd
Entrez un premier nombre:
10535
Entrez un deuxième nombre:
5250
Le PGCD de 10535 et 5250 est 35
```

Structures de données (i)

But: Stocker des données de manière à y accéder efficacement : ajout, suppression, recherche.

Exemple: Dictionnaire de mots

Fonctions:

1. **Lire le dictionnaire déjà trié qui se trouve sur fichier**
2. **Effectuer des recherches de mots : extraire tous les mots qui correspondent au pattern t . m e , où "." est un "place holder"**

Recherche dans un dictionnaire: chaîne de caractères

Le programme linearDictF stocke les mots dans une longue chaîne de caractères.

```
#include <iostream> // search in linearDictF.cpp  BS,RDH 16.10.05
                //command line: linearDictF worldlist.txt myPattern
#include <stdio.h>
#define MAX_WORD_LENGTH 64
#define NB_WORDS 100000
#define NB_CHARS 3

void patternMatch(const char *pattern,const char *word,
                                                size_t patternLength)
// Matches a string against a given pattern, and
// prints the string if a match is found, dot '.' is single
// wild char, i.e. 't.me' matches 'time' of 'tome'

// Return if lengths or content do not match
{  if(patternLength!=strlen(word)-2) return;// word: length+CR+LF
   for(size_t i=0;i<patternLength;++i)
       if(pattern[i]!='.' && pattern[i]!=word[i]) return;
   // If we get here, we found a match
   std::cout << word << " ";
}
```

Recherche dans un dictionnaire (ii)

```
int main(int argc, char *argv[]) {
  if(argc!=3) {
   std::cout << "Please specify name of dictionary \
                              file and pattern to look for" << std::endl;
   return 1;
  }

  FILE * inputFile; //  Open read only file, C style

  if (!(inputFile=fopen(argv[1], "rb"))) { // opening error
   std::cout << "Could not open file " << argv[1] << std::endl; return 1;
  }

  // Words are stored in a large array 100000 words of 64 characters)
  char *words=(char*)malloc(MAX_WORD_LENGTH*NB_WORDS*sizeof(char));
  int wordCount=0; int i;

  //  Loop until the stream cannot be read any more
  while(fgets(words+wordCount*MAX_WORD_LENGTH,
                MAX_WORD_LENGTH, inputFile)!=NULL) wordCount++;

/* // verify that words correctly stored
  for (i=0;i<(wordCount);i++) {
   int length= strlen(words+MAX_WORD_LENGTH*i);
   printf(" length=%d \t%s\n",length,words+MAX_WORD_LENGTH*i);
  }  */
```

Recherche dans un dictionnaire (iii)

```
  std::cout<< "Stored " << wordCount << " words in vector" <<std::endl;

  int count3=0; // Count number of words with NB_CHARS characters

  for(i=0;i<wordCount*MAX_WORD_LENGTH;i+=MAX_WORD_LENGTH)
   if(strlen(words+i)==NB_CHARS+2) count3++; //(+CR+LF)

  std::cout << "Found " << count3 << " 3-character words" << std::endl;

  //  Match pattern
  size_t len=strlen(argv[2]); // pattern length
  printf("Pattern length = %d \n",len);
  for(i=0;i<wordCount*MAX_WORD_LENGTH;
      i+=MAX_WORD_LENGTH)
   patternMatch(argv[2],words+i,len);
}
```

Console:
>linearDict wordlist.txt mi.e
Stored 75273 words in dictionary
Found 636 3-character words
Pattern length = 4
mice mike mile mime mine mire mite mice

Questions:

1. **Comment fonctionne ce programme ?**
2. **Pourquoi est-il utile de vérifier la longueur des mots lus du fichier ?**
3. **Comment faudrait-il modifier ce programme pour rechercher des mots qui ont une certaine racine (par exemple "… jour…") ?**
4. **Peut-on prédire la durée de recherche d'un mot dans le dictionnaire ?**

Structures de données (ii)

But: Stocker des données de manière à y accéder efficacement : ajout, suppression, recherche.

Exemple:

Liste de noms (+No de téléphone) par ordre alphabétique (livre de téléphone électronique)

Fonctions:

1. **établir la liste ordonnée,**
2. **rajouter un nom,**
3. **rechercher un nom,**
4. **enlever un nom.**

Tableau statique

Enumérez les avantages et inconvénients.

Est-il possible de rendre ce tableau plus dynamique?

Nom	No de tel
Abecassis	3045
Alfonso	2814
Archimède	9736
⋮	⋮
Christin	2540
Cléopâtre	3366
⋮	⋮
Zappacosta	9987
Zeppelin	3572

Tableau de taille fixe

```
#include <stdlib.h>  //fixedArrayF.cpp
#include <stdio.h>

class FixedArray
{
private:
   int capacity;              // capacity of array
   int elementCount;          // current number of element in array
   int *data;                 // array storing the elements

public:
   FixedArray(int capacity){              // Constructor
        this->capacity = capacity;
        elementCount = 0;
        data = (int*)malloc(capacity*sizeof(int));
   }

   virtual ~FixedArray() { free(data);}  // Destructor
```

Tableau de taille fixe (2)

```
  void pushBack(int el)    {       // Add element at the end of
      if(elementCount==capacity) // the array if space left
            return;                // array full: donothing
      data[elementCount]=el; elementCount++;
  }
  void pushFront(int el)   {       //Add element at the beginning
      if (elementCount==capacity) //of the array if space left
            return;                // array full: donothing
      for(int i = elementCount-1; i >= 0; i--)
                                   data[i+1]=data[i];
      data[0]=el; elementCount++;
  }
  int getSize() { return elementCount;   }
  int elementAt(int pos) { return data[pos]; }
  void clear() { elementCount = 0; }
  void print() {                   // print content of array
      for(int i=0; i<elementCount; i++) printf("%d ", data[i]);
      printf("\n");        }
};
```

Utilisation de la classe FixedArray

```
void main(int argc, char *argv[])
{
   if(argc!=2){
        printf("Please specify the size of the array\n");
        exit(1);}
   int arraySize = atoi(argv[1]);
   FixedArray fa(arraySize);

   // Add elements at the end of the array
   for(int i=0;i<arraySize;i++)   fa.pushBack(i);
   fa.print(); fa.clear();

   // Add elements at the beginning of the array
   for(int j=0;j<arraySize;j++)   fa.pushFront(j);
   fa.print();
}
```

Console:

```
> FixedArray 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

Questions

1. Carnet d'adresses
   Peut-on créer un carnet d'adresses, avec comme clef d'accès le "Nom" et comme champ l'adresse ? Comment ?

2. Serait-il possible de créer un tableau de taille qui puisse être augmentée pendant l'exécution du programme ? Comment ?

Solutions

1. Créer un tableau de classes, chaque classe comprenant le nom et l'adresse.

2. Créer un tableau à une taille initiale, puis une fois remplit, réallouer un nouveau tableau et recopier l'ancien dans le nouveau.

   Temps d'exécution ?

   Autres solutions plus économiques en temps d'exécution ?

Classe Tableau redimensionnable

```
#include <stdlib.h>  // resizableArrayF.cpp  BS, RDH
#include <stdio.h>
#include <memory.h>

// This class defines a resizable array
class ResizableArray
{
private:
   int capacity;                // Current capacity of array
   int capacityIncrement;       // resizing size increment
   int elementCount;            // Current number of elements in array
   int *data;                   // Array storing the elements
public:
   ResizableArray(int capacityIncrement) {    // Constructor
        this->capacityIncrement=capacityIncrement;
        capacity=0;
        elementCount=0;
        data=NULL;
   }

   virtual ~ResizableArray() { if(data!=NULL) delete data; }
                                              // Destructor
```

Classe Tableau redimensionnable (2)

```
void pushBack(int el) { // Add an element at end
  if(elementCount==capacity) { // Reallocate if necessary
  data=(int*)realloc(data,(capacity+capacityIncrement)*sizeof(int));
       capacity+=capacityIncrement; // realloc: pseudo agrandissement
  }  // Store new element at end of array                  // de taille
  data[elementCount]=el;
  elementCount++;
}

void pushFront(int el) { // Add an element at beginning
  if(elementCount==capacity) { // Reallocate if necessary
// Initialize new array, and copy elements from old to new array
     int *tmp=(int*)malloc((capacity+capacityIncrement)*sizeof(int));
     memcpy(tmp+1,data,elementCount*sizeof(int));
     if(data!=NULL)
             free(data);
     data=tmp;
     capacity+=capacityIncrement;
  }
  else {              // Move every elements to its next position
     memmove(data+1,data,elementCount*sizeof(int));
  }
  data[0]=el; elementCount++;
                    // Store new element at first position in array
}
```

Classe Tableau redimensionnable (3)

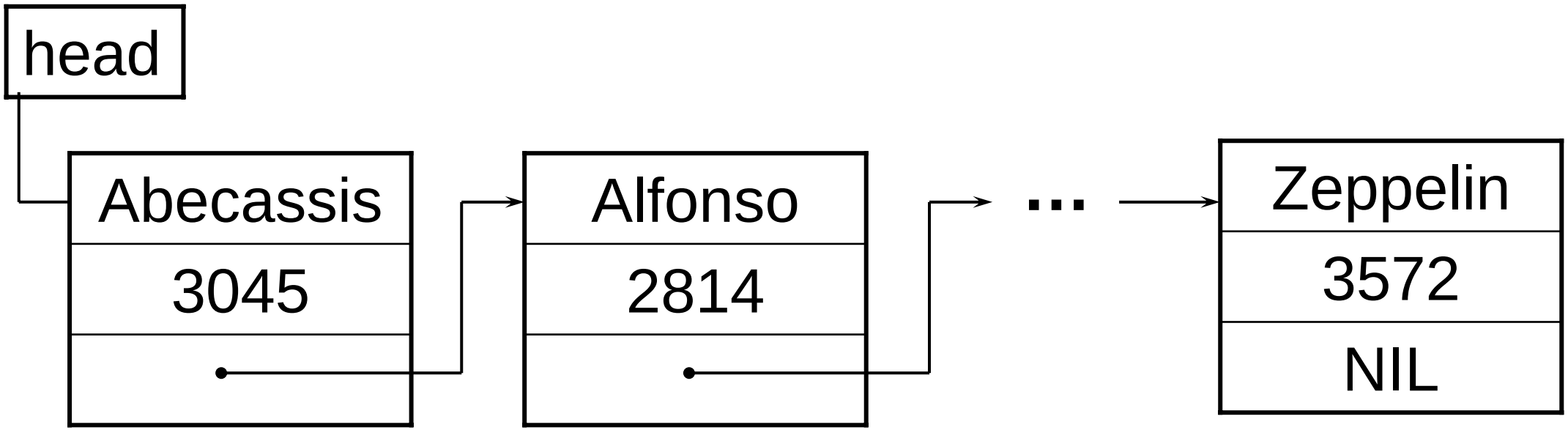
```
    // Returns the number of elements in the vector
    int getSize() { return elementCount; }

    // Retrieve an element at fixed position
    int elementAt(int pos) { return data[pos]; }

    void clear(){// Remove elements in array and deallocate memory
        if(data!=NULL)
                free(data);
        data=NULL;
        capacity=0;
        elementCount=0;
    }

    void print() { // Print array content
        for(int i=0;i<elementCount;i++)
                printf("%d\n",data[i]);
    }
};
```

Classe Tableau redimensionnable (4)

```
int main(int argc,char *argv[])
{  if(argc!=3) {
       printf("Please specify number of
              elements to insert and resize increment\n");
       exit(1);
   }

   int elementsToAdd=atoi(argv[1]);
   int capacityIncrement=atoi(argv[2]);

   ResizableArray ra(capacityIncrement);

   // Add elements at end of array
   for(int i=0;i<elementsToAdd;i++) ra.pushBack(i);
   ra.print();
   ra.clear();

   // Add elements at front of array
   for(int j=0;j<elementsToAdd;j++) ra.pushFront(j);
   ra.print();
}
```

Liste chaînée

Enumérez les avantages et inconvénients.



Il y a-t-il un moyen d'accéder rapidement à un élément qui se trouve dans la liste?

Liste chaînée simple (i)

```
#include <stdlib.h>// SimpleLinkedList.cpp BS, RDH, 28.9.05
#include <stdio.h>// Simple linked list to store positive integers

// We assume that only positive integers are stored
#define ILLEGAL_POSITION -1

class LinkedListNode {
private:
   int el; // Value of local element
   LinkedListNode *next;          // Pointer to next list element
public:
   // Constructor
   LinkedListNode(int element) { el=element; next=NULL; }

   virtual ~LinkedListNode() { }  // Destructor

   void pushBack(int el) {        // Add an element at the end
        LinkedListNode *lln=this;
        while(lln->next!=NULL)    // Find last element
              lln=lln->next;
        lln->next=new LinkedListNode(el); // Hook new element
   }
```

Classe liste chaînée simple (ii)

```
LinkedListNode *pushFront(int el) // Add an element at the front
{                          // and return a pointer to the new element
     LinkedListNode *lln=new LinkedListNode(el);
     lln->next=this;
     return lln;
}
// Retrieve an element at a specified position
int elementAt(int pos) {
     LinkedListNode *lln=this;
     for(int i=0;i<pos;i++)
             if(lln->next==NULL) return ILLEGAL_POSITION;
             else lln=lln->next;
     return lln->el;
}

int getSize() {                        // Return the number of elements
     int size=1;
     LinkedListNode *lln=this;
     while(lln->next!=NULL) {
             size++;
             lln=lln->next;
     }
     return size;
}
```

Liste chaînée simple (iii)

```
   void clear()                        // Delete all nodes
   {   LinkedListNode *lln=next;
       while(lln!=NULL) {
              LinkedListNode *tmp=lln->next;
              delete lln;
              lln=tmp;
       }
       next=NULL;
   }

   void print() {              // Print all elements
       LinkedListNode *lln=this;
       do {
              printf(" %d ",lln->el);
              lln=lln->next;
       } while(lln!=NULL);
       printf("\n");
   }
};
```

Liste chaînée simple (iv)

```
int main(int argc, char *argv[])
{
   if(argc!=2) {
        printf("Specify nb elements of list\n"); exit(1); }
   int nbElements =atoi(argv[1]);

   // Add elements at end of list
   LinkedListNode *listHead=new LinkedListNode(0);
   for(int i=1;i<nbElements ;i++)
        listHead->pushBack(i);
   listHead->print();
   listHead->clear();

   // Add elements at front of list
   for(int j=1;j<nbElements;j++)
        listHead=listHead->pushFront(j);
   listHead->print();

   // Clean up
   listHead->clear();
   delete listHead;
}
```

Exemple d'exécution

Console:

```
D:\Prog3\prog\Debug> simpleLinkedListF 15
0  1  2  3  4  5  6  7  8  9  10  11  12  13 14
14  13  12  11  10  9  8  7  6  5  4  3  2  1 0
```

Liste chaînée (i)

```
#include <stdlib.h> //linkedListF.cpp, BS
#include <stdio.h>

// Linked list node
class LinkedListNode
{
public:    // can we make it private ?
   int el; // Value of local element
   LinkedListNode *next; // Pointer to next list element

public:
   LinkedListNode(int element) { // Constructor
        el = element;
        next = NULL;
   }
   virtual ~LinkedListNode() { } // Destructor
};
```

Liste chaînée (ii)

```
class LinkedList
{
private:
   LinkedListNode *head, *tail; // Pointers to first and last nodes
   int size; // Size of list
public:
   LinkedList() { // Constructor
        head = NULL;
        tail = NULL;
        size = 0;
   }

   virtual ~LinkedList() { clear(); } // Destructor

   void pushBack(int el) { // Add an element at the end
        LinkedListNode *lln = new LinkedListNode(el);
        if(head == NULL) // If list is empty
                head = lln;
        else
                tail->next = lln;
        tail = lln;
        size++;
   }
```

Liste chaînée (iii)

```
void pushFront(int el) { // Add an element at the beginning
     LinkedListNode *lln = new LinkedListNode(el);
     if(head == NULL) tail=lln;  // If list is empty
     lln->next = head;
     head = lln;
     size++;
}

int elementAt(int pos) // Retrieve an element at a specified position
{                      // no check is done that position is valid
     LinkedListNode *current = head;
     for(int i=0;i<pos;i++)
              current=current->next;
     return current->el;
}

int getSize() { return size; } // Return the number of elements

void clear() { // Delete all nodes in the list
     LinkedListNode *current = head;
     while(current != NULL) {
              LinkedListNode *tmp = current;
              current = current->next;
              delete tmp;
     }
     head = NULL; tail = NULL; size = 0;
}
```

Liste chaînée (iv)

```
   void print() { // Print all elements
        LinkedListNode *current = head;
        while(current != NULL) {
                printf("%d ",current->el);
                current=current->next;
        } printf("\n");
   }
};

int main(int argc,char *argv[]) {
   if(argc!=2) { printf("Please specify nb elements of list\n");
                                                exit(1); }

   int nbElements = atoi(argv[1]);
   LinkedList ll;
   // Add elements at end of list
   for(int i=0;i< nbElements;i++)     ll.pushBack(i);
   ll.print();
   ll.clear();

   // Add elements at front of list
   for(int i=0;i< nbElements;i++)     ll.pushFront(i);
   ll.print();
}
```

Comment protéger attributs de la classe LinkedListNode ?

Solution 1. friend permet d'accéder aux membres de LinkedList :

```
//linkedListFfriend.cpp
class LinkedListNode
{
private:
     friend class LinkedList; // enables access from LinkedList
     int el;                  // Value of local element
     LinkedListNode *next;     // Pointer to next list element
public:
     LinkedListNode(int element) { // Constructor
       el=element;
       next=NULL;
     }
     virtual ~LinkedListNode() { } // Destructor
};
```

Comment protéger attributs de la classe LinkedListNode ? (suite)

Solution 2. Insérer la class `LinkedListNode` dans la classe `LinkedList`

```
// linkedListFimbricate.cpp
class LinkedList
{
  class LinkedListNode
  {
   public:  // public since protected by LinkedListNode
     int el; // Value of local element
     LinkedListNode *next; // Pointer to next list element
     LinkedListNode(int element) { // Constructor
        el=element;
        next=NULL;
     }
     virtual ~LinkedListNode() { } // Destructor
  };
  private:
     LinkedListNode *head, *tail; // Ptr to 1st and last nodes
     int size; // Size of list
    // continuation
```

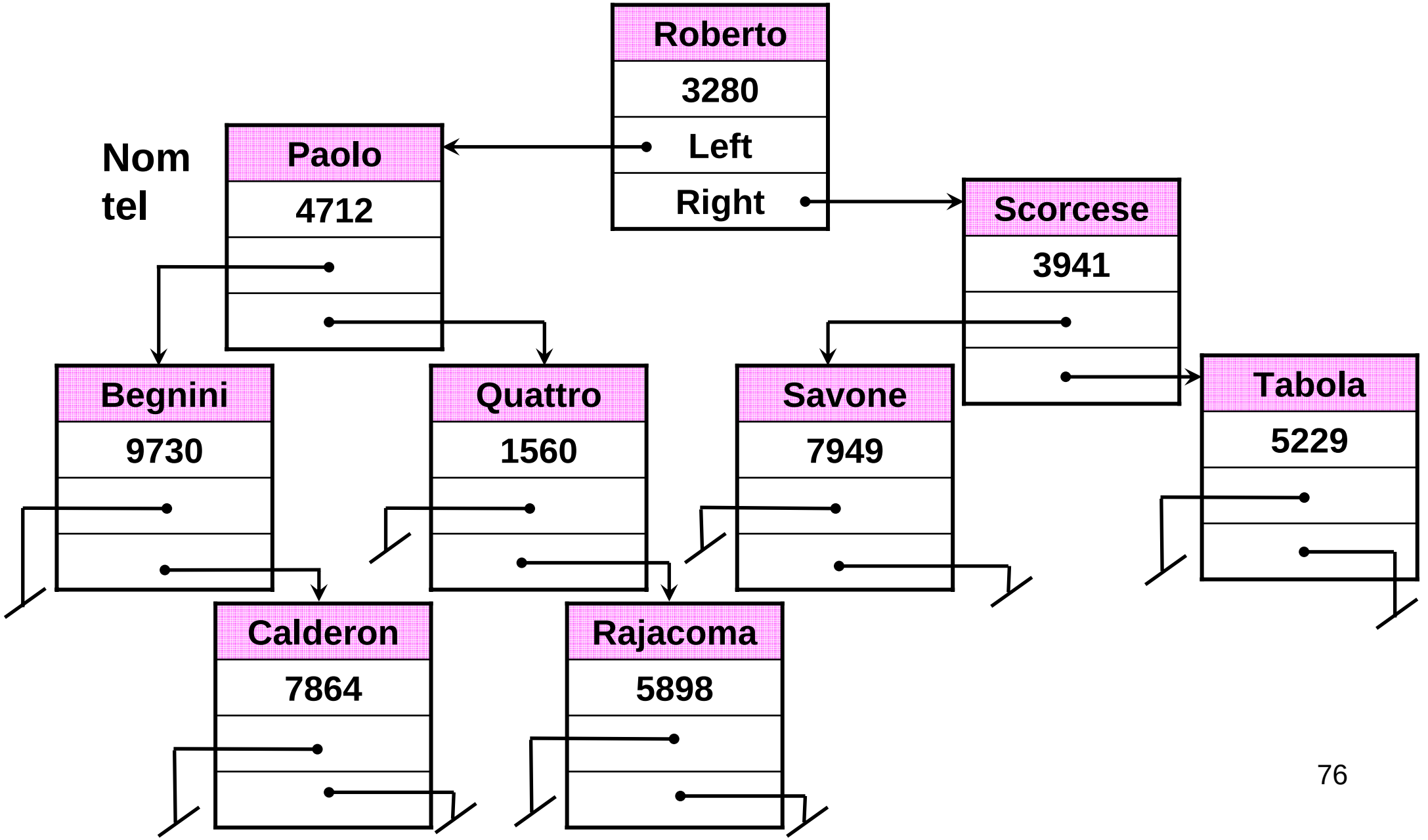
Questions

Admettons que l'on cherche à créer un carnet d'adresses avec comme clef : *Nom* et comme information *prénom* et *adresse*. Nous désirons un carnet avec *n* noms.

1. Comment adapter le programme linkedListF.cpp ?
2. Comment faire en sorte que le temps d'accès à un nom soit réduit ?
3. Au pire des cas, quel est le temps de recherche d'un nom ?
4. Si les accès aux noms sont uniformément répartis, quel sera le temps moyen d'accès à un nom ?

Arbre binaire

Enumérez les avantages et les inconvénients.



Arbres binaires (i)

```
#include <stdlib.h>    //binarySearchTreeF.cpp, BS, RDH, 15 oct 05
#include <stdio.h>
// We assume all the values in the tree are positive
#define NO_VALUE -1

// Defines a node in a binary search tree
class BSTNode
{
private:
  int el;               // Value of local element
  BSTNode *left, *right; // Left and right sons
public:
  BSTNode() {           // Constructor
        el=NO_VALUE;
        left=NULL; right=NULL;
  }

  BSTNode(int el) {     // Constructor
        this->el=el;
        left=NULL; right=NULL;
  }

  ~BSTNode() { // Destructor (delete all leafs)
        if(left!=NULL) delete left;
        if(right!=NULL) delete right;
  }
```

Arbres binaires (ii)

```
void insert(int newEl) // Insert an element in the tree
{                      // if it does not already exist
if(el==NO_VALUE) {    // If tree is empty
      el = newEl; return;
}

int found = 0; // Position of new element is found (boolean)
BSTNode *current = this; // Current node in the tree
BSTNode **leaf = NULL;   // Leaf that should be updated

while(!found)           // Visit the tree until correct leaf is found
{
  if(newEl == current->el) return;     // If element already exists

  if(newEl>current->el) {              // If we should go right
     if(current->right == NULL) {      // If right son does not exist
                                leaf = &(current->right);
                                found = 1;
     } else current = current->right;
  }
  else { if(current->left == NULL) {   // If left son does not exist
                                leaf = &(current->left);
                                found = 1;
                           } else current = current->left;
               }
}
(*leaf) = new BSTNode(newEl); // Create new node and hang as leaf
}
```

Arbres binaires (iii)

```
void insertR(int newEl)  // Insert an element in the tree if
 {                       // it does not already exist (recursive)
  if(el==NO_VALUE) {    // If tree is empty
        el=newEl; return;
  }

  if(newEl>el) {        // If we should go right
        if(right==NULL) // If right son does not exist
                  right = new BSTNode(newEl);
        else right->insertR(newEl);
  }
  else if(newEl<el) {   // If we should go left
        if(left==NULL)  // If left son does not exist
                 left=new BSTNode(newEl);
        else left->insertR(newEl);
  }
 }

 int getSize() {        // Compute the size of the tree (recursive)
  if(el==NO_VALUE)      return 0; // If tree is empty

  int size=1;
  if(left!=NULL)        // Add contribution from left branch
                size+=left->getSize();
  if(right!=NULL)       // Add contribution from right branch
                size+=right->getSize();
  return size;
 }
```

Arbres binaires (iv)

```
void print() { // Print tree elements in a sorted manner
  if(el==NO_VALUE) return;   // If tree is empty
  if(left!=NULL) left->print();
  printf("%d ",el);
  if(right!=NULL) right->print();
 }

 void printAsTree() {         // Print elements as a tree
  if(el==NO_VALUE) return;    // If tree is empty

  bool foundNode = false; int targetDepth = 0;

  do { // Run while there are nodes at specified depth,
        // then incr targetDepth
       char *currentLocation = new char[targetDepth+1];
       currentLocation[targetDepth] = '\0';
       foundNode = printNodesAtDepth(targetDepth,0,currentLocation);
       delete currentLocation;
       printf("\n");
       targetDepth++;
  } while(foundNode);
 }
```

Arbres binaires (v)

```
private:
  // Prints the current node if targetDepth==currentDepth, otherwise moves down recursively
    bool printNodesAtDepth(int targetDepth, int currentDepth, char *currentLocation){
    if(currentDepth==targetDepth)      // If we reached the requested depth
    {     printf("%s|%d  ",currentLocation,el); return true; }
                                       // Tell caller that a node was printed
    bool printedNodesLeft=false; bool printedNodesRight=false;

    if(left!=NULL) {                   // Move down left path
          currentLocation[currentDepth]='l';
          printedNodesLeft=left->printNodesAtDepth(targetDepth, currentDepth+1,currentLocation);
    }
    if(right!=NULL) {                  // Move down right path
          currentLocation[currentDepth]='r';
          printedNodesRight=right->printNodesAtDepth(targetDepth, currentDepth+1,currentLocation);
    }
    return printedNodesLeft||printedNodesRight;
  }
}; // end class BSTNode

int main(int argc,char *argv[]) {
    if(argc!=2) { printf("Please specify nb elements to add\n"); exit(1); }
    int elementsToAdd=atoi(argv[1]);
    BSTNode b;
    srand(14); // Initialize pseudo random number generator
    for(int i=0;i<elementsToAdd;i++) { // Fill in tree
          int r=rand();
          printf("%d ",r%elementsToAdd);
          b.insert(r%elementsToAdd);
    }
    printf("\n\n"); b.print(); printf("\n"); b.printAsTree(); return 0;
}
```

Exécution & questions

Console :

```
>binTreeN 35
14 0 22 23 32 10 26 26 21 30 2 17 5 33 12 18 8 9 32 0 13 33 2 12
   6 14 19 24 5 7
1 17 10 6 9

0 1 2 5 6 7 8 9 10 12 13 14 17 18 19 21 22 23 24 26 30 32 33
|14
l|0  r|22
lr|10  rl|21  rr|23
lrl|2  lrr|12  rll|17  rrr|32
lrll|1  lrlr|5  lrrr|13  rllr|18  rrrl|26  rrrr|33
lrlrr|8  rllrr|19  rrrll|24  rrrlr|30
lrlrrl|6  lrlrrr|9
lrlrrlr|7
```

Questions:

1. **Comment se fait l'insertion d'un élément dans l'arbre (insert) ?**
2. **Comment se fait l'insertion, selon la fonction récursive (insertR) ?**
3. **Comment se calcule la taille de l'arbre ?**
4. **Comment s'affiche l'arbre ?**

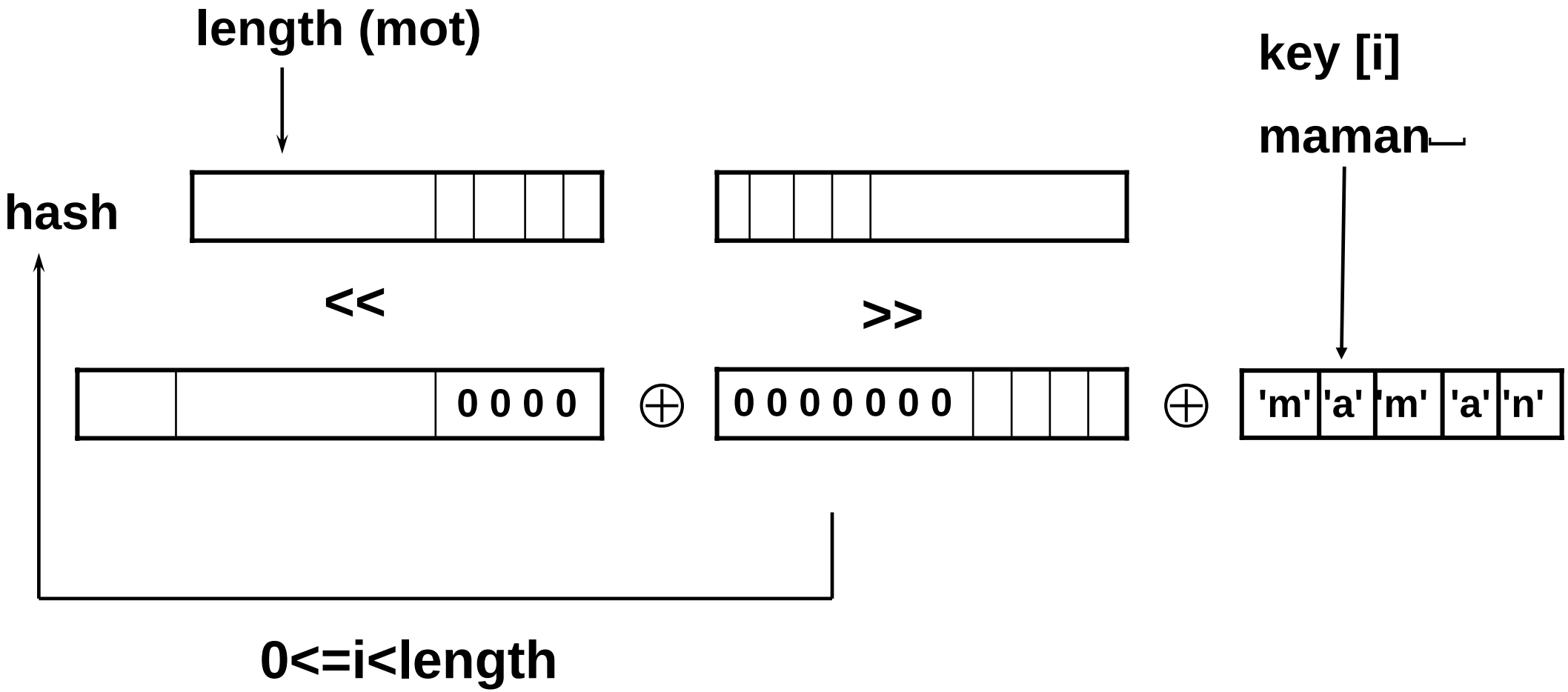
Accès par table de hachage

La fonction de hachage permet de distribuer de manière uniforme les mots sur les entrées de la table, indépendamment de leur longueur. Il y a risque de collision. A chaque mot (clef), une entrée est calculée. Le signe ^ est le ou-exclusif bit à bit.

```
// hash function
unsigned int hash(const char *key)
{                   //(uses a 4-bit circular shift)
  size_t length = strlen(key);
  unsigned int hash, i;
  for (hash=length, i=0; i<length; ++i)
     hash = (hash<<4)^(hash>>28)^key[i];
                   // ^ou-exclusif bit a bit
     return (hash%NB_WORDS);
  }
```

Fonction de hachage

Rotation des bits et ou-exlusif avec octets successifs de la clef



Accès par table de hachage

```
#include <stdio.h>  // BS, RDH, 17.10.05 hashTableDictF.cpp
#include <string>
#include "time.h"

// NB_WORDS must be prime
#define NB_WORDS 100003
#define LENGTH_MAX 64

// Uncomment to measure performance of pattern matching
// #define MEASURE_PERF

class HashtableEntry { //hashtable entry
public:
   char *word;
   HashtableEntry *next;

public:
   HashtableEntry(const char* word)
   {    this->word=(char*)malloc((strlen(word)+1)*sizeof(char));
        strcpy(this->word,word);
        next=NULL;
   }
   ~HashtableEntry(){ free(word);     }
};
```

Accès par table de hachage

```
class HashtableDictionary { // stores hashtable entries
private:
  HashtableEntry **entries; // The hash table

  unsigned int hash(const char *key)      // hash function
  {size_t length=strlen(key);             //(uses a 4-bit circular shift)
   unsigned int hash, i;
   for (hash=length, i=0; i<length; ++i)
        hash = (hash<<4)^(hash>>28)^key[i];//^ou-exclusif bit a bit
        return (hash%NB_WORDS);
  }

public:
  HashtableDictionary() { // Constructor, Initialise memory
   entries=(HashtableEntry**)malloc(NB_WORDS*sizeof(HashtableEntry*));
   memset(entries,NULL,NB_WORDS*sizeof(HashtableEntry*));
  }
  ~HashtableDictionary() // Destructor (delete all entries)
  {  for(int i=0;i<NB_WORDS;++i) {
        HashtableEntry *current=entries[i];
        while(current!=NULL){
                HashtableEntry *tmp=current;
                current=current->next;
                delete tmp;
        }
    }
    free(entries);
  }
```

Accès par table de hachage

```
void addWord(const char *word)  //Add word into dictionary
 {     // Compute position of word
  int pos=hash(word);
  // Check that entry does not exist yet
  HashtableEntry *current=entries[pos];
  bool exists=false;
  while(current!=NULL && !exists) {
    if(strcmp(word,current->word)==0)
       exists=true;
       current=current->next;
  }
  // If not, push it in front of other entries
  if(!exists) {
       HashtableEntry *newEntry=new HashtableEntry(word);
       newEntry->next=entries[pos];
       entries[pos]=newEntry;
  }
 }
```

Accès par table de hachage

```
void patternMatch(const char *pattern)
 { // Find words in the dictionary that match the specified pattern
  size_t patternLength=strlen(pattern);
  for(int i=0;i<NB_WORDS;++i){
    // For each entry in the table, go through list of words
       HashtableEntry *current=entries[i];
       while(current!=NULL) {
              if(patternLength==strlen(current->word)) {
                bool mismatch=false;
                for(size_t j=0;j<patternLength && !mismatch;++j)
                  if(pattern[j]!='.' && pattern[j]!=current->word[j])
                              mismatch=true;
                              #ifndef MEASURE_PERF
                                      // If we measure performance
                              if(!mismatch)
                                      printf("%s\n",current->word);
                              #endif
              }
              current=current->next;
       }
 }
}
```

Accès par table de hachage

```
int countWords(int wordLength)
 { //Count the number of words that contain the specified number of characters
  int count=0; HashtableEntry *current;
  for(int i=0;i<NB_WORDS;++i) {
      current=entries[i];
       while(current!=NULL) {
               if(strlen(current->word)==wordLength) count++;
               current=current->next;
       }
  }
  return count;
 }

void print() // Print content of dictionary
{  HashtableEntry *current;
   for(int i=0;i<NB_WORDS;++i) {
       current=entries[i];
       while(current!=NULL) {
               printf("%s\n",current->word);
               current=current->next;
       }
  }

 }
}; // end class
```

Accès par table de hachage

```
// Opens a list of words, loads it into the hash table, and
// extracts  information from the dictionary

int main(int argc, char *argv[])
{ int j;
  if(argc!=3) {
   printf("Please specify name of dictionary file and \
           word to look for\n"); return 1;
  }

  FILE *fptr=fopen(argv[1],"r"); //  Open file, C style
  if(fptr==NULL) {
   printf("Could not open file %s \n",argv[1]);
   return 1;
  }

  //  Fill data structure
  HashtableDictionary dict;
  char buf[64]; // Assume no word is longer than 64 characters
  while(fscanf(fptr,"%s",buf)!=EOF) dict.addWord(buf);
  dict.print();
```

Accès par table de hachage

```
   //  Count number of words with three characters
  double st=Time::get();
  int count;
  for(j=0;j<1000;++j)
        count=dict.countWords(3);
  printf("time counting 3 char words: %f\n",Time::get()-st);
  printf("%d 3 letter words\n",count);

//  Match pattern
#ifndef MEASURE_PERF
   printf("PatternMatch:\n");
   dict.patternMatch(argv[2]);
#else
   st=Time::get();
   for(j=0;j<1000;++j) dict.patternMatch(argv[2]);
   printf("%f\n",Time::get()-st);
#endif
   return 0;
}
```

Questions:

1. **Comment vérifier que la fonction de hachage fonctionne bien ?**
2. **Dans quelle tâches la table de hachage permet-elle d'accélérer l'accès aux données ?**
3. **Pour des tâches de recherche de mots incomplets, est-ce que la table de hachage ou l'arbre binaire est plus efficace ?**
4. **Pourquoi la longueur des mots lus ne prend pas en compte les CR et LF ?**

Mesure du temps (i)

```
#ifndef TIME_H
#define TIME_H

// Helper class to measure time
// Two versions for UNIX and Windows platforms
#ifndef WIN32

// UNIX code
#include <sys/time.h>
class Time {
public:
//   Returns the number of seconds since the Epoch
//   (00:00:00 UTC, January 1, 1970)

  static double get()
  {// Use 'man gettimeofday' for info on timeval structure
   // and gettimeofday functions
   struct timeval t;
   gettimeofday(&t,NULL);
   return t.tv_sec+((double)t.tv_usec)/1000000;
  }
};
```

Mesure du temps (ii)

```
#else

#include <windows.h> // Windows code
class Time
{ public:
// Returns the number of seconds since the system was
// started
// Time wraps around to zero every 49.7 days
  static double get()
  {
   // DWORD is an unsigned int
   // GetTickCount returns a number of seconds
   DWORD t=GetTickCount();
   return ((double)t)/1000; }
};

#endif
#endif
```

Les patrons de fonctions

S'applique à n'importe quel type –> programmation de plus haut niveau

```
// functionTemplate.cpp, inspiré par J. H u b b a r d, Programmer en C++,
#include <iostream>
using namespace std;
#include <string>  // utilise pour stocker noms et faire des comparaisons
#define NB_NOMS 7
#define NB_CHIFFRES 10

template <typename T> void swap(T* x, T* y)  // patron de fonction
{ T temp = *x;
  *x = *y;
  *y = temp;
}
// bubble sort
template < typename T> void sort(T* v, int n) // patron de fonction
{ for (int i=1; i<n; i++)
    for (int j=0; j<n-i; j++)
      if (v[j] > v[j+1]) swap(v[j], v[j+1]);
}
template < typename T> void print(T* v, int n) //patron de fonction
{ for (int i=0; i<n; i++) cout << " " << v[i];
  cout <<  endl;
}
```

Les patrons de fonctions (ii)

```
int main()
{ int a[NB_CHIFFRES] = {55, 33, 88, 11, 44, 99, 77, 22,
  66, 28};
  print(a,NB_CHIFFRES);
  sort(a,NB_CHIFFRES);
  print(a,NB_CHIFFRES);

  string s[NB_NOMS] = {"Jean", "Albert", "Efraim",
  "Catherine", "Marelyse", "Berthe", "Gustave"};
  print(s,NB_NOMS);
  sort(s,NB_NOMS);
  print(s,NB_NOMS);
}
```

Console

```
 55 33 88 11 44 99 77 22 66 28
 11 22 28 33 44 55 66 77 88 99
 Jean Albert Efraim Catherine Marelyse Berthe Gustave
 Albert Berthe Catherine Efraim Gustave Jean Marelyse
```

Les patrons de classe (templates)

```
#include <stdio.h>  // RDH 20Nov05 stackTemplate.cpp
#include <string>
// gestion de pile avec patron
template <typename T>
class Stack
{ private :
   T* top;           // top of stack
   T* base;   // base address of stack
   int stacksize;
  public:
     Stack(int s):stacksize(s) { top=base=new T[s];}
     ~Stack() {delete[] base;}
     void push(T & item) {*top++ = item;}; //postincrement
       // attention: no error check
      T pop(){return *--top; } // predecrement
     int nbItems() {return top-base;}
};
```

Les patrons: pile de points

```
class PointT  // points géométriques (x,y)
{
private:
  int x, y;
public:
  PointT(){x=0;y=0;};
  PointT (const PointT & obj):
     x(obj.x), y(obj.y){}  //constr. copie
  PointT(int xIn, int yIn):x(xIn),y(yIn){};
  int gx() {return x;}
  int gy() {return y;}
};
```

Les patrons: pile de points (ii)

```
int main(void)
{
  const int nbPoints=10;
  Stack<PointT> stackOfPoints(nbPoints);
  for (int i=0;i<nbPoints;i++)
  {   // stocker sur la pile les points successifs
      PointT myPoint(i,i);
      stackOfPoints.push(myPoint);
  }
  while (stackOfPoints.nbItems()>0) // reprendre points
  { PointT myPoint=stackOfPoints.pop();
    printf(" %d %d  ", myPoint.gx(), myPoint.gy());
  } return 0;
}
```

Console

```
> 99 88 77 66 55 44 33 22 11 00
```

Les patrons: pile de chaînes de caractères

Comment faire pour créer une pile de chaînes de caractères ?

```
typedef char myStringT[STRLENGTH];
char seps[]   = " ,\t\n";  // space, tab, CR separators
char * ptrToken;

Stack<myStringT> stackOfStrings(10);
myStringT myString = {voici une phrase};
ptrToken = strtok( myString, seps );
stackOfStrings.push(myString);
```

Qu’est-ce qui est mis sur la pile ?

Par quel mécanisme ?

Les patrons: pile de chaînes de car.

Solution: **créer une classe qui englobe une chaîne de caractères**

```
// RDH 21Nov05 stackTemplateString.cpp
class EnclString    // encloses an array of chars into a class
{      // makes copy possible by default = operator definition
private:
   myStringT string;
public:
   EnclString(){string[0]=0;} //default constructor
   EnclString(const myStringT inString) // string constructor;
   {    int i=0;
        while (inString[i]!= 0)
        { string[i]=inString[i]; i++; } string[i]=0;
   }
   void getString(myStringT outString)
   {    for (int i=0;i<STRLENGTH;i++) outString[i]=string[i];
   }
};
```

Les patrons: pile de chaînes de car.

```
int main(void)          // stack of strings
{
char seps[]   = " ,\t\n";  // separateurs: espace, tab, retour chariot
char * ptrToken;           // pointer to string
Stack<EnclString> stackOfStrings(10);
myStringT myString, outString;
printf("Entrez chaines de caracteres a mettre sur la pile \n");
fgets( myString, 80, stdin);// reads lines of strings
ptrToken = strtok( myString, seps ); // pointer to first strimg
while( ptrToken != NULL )
{  EnclString myEnclStr(ptrToken);    // encloses string into class
   stackOfStrings.push(myEnclStr);  // push the string on the stack
   ptrToken = strtok( NULL, seps ); // pointer to next string

}
   printf("\naffiche contenu pile: \n\n"); // print content of stack
   while (stackOfStrings.nbItems()>0)
   {  EnclString myEnclStr(stackOfStrings.pop()); // uses default constr
     myEnclStr.getString(outString);
     printf(outString); printf("\n");
   } return 0;
}
```

Les patrons: classe Vector

Mission:

A. créer une classe patron *tableau d'objets* similaire à `fixedArrayF.cpp` avec:

1. **Constructeur initialise le tableau selon la taille spécifiée**
2. **Constructeur de copie**
3. **Opérateur d'assignation**
4. **Accès aux éléments du tableau à travers indices: `[i]`**

B. créer une classe patron dérivée avec itérateur débutant à une valeur spécifiée

Les patrons: classe Vector (ii)

```
//  Dérivé de programmation en C++ de J. H u b b a r d
//  vectorTemplate.cpp: Vector is an array with indices starting with 0
//  Array starts with a user-defined starting index
//  compiled with: g++ vectorTemplate.cpp -o main

#include <iostream>
using namespace std;
template<typename T> class Vector
{
protected:
  T* ptrT;
  unsigned vSize;
  void copy(const Vector<T>&);  //define externally
public:
  Vector(unsigned n=8) : vSize(n), ptrT(new T[n]) { } //default size : 8
  Vector(const Vector<T>& v) : vSize(v.vSize), ptrT(new T[vSize])
  { copy(v);
  }
  ~Vector() { delete [] ptrT; }
  Vector<T>& operator=(const Vector<T>&);
  T& operator[](unsigned i) const { return ptrT[i]; }
  unsigned getSize() { return vSize; }
};
```

Les patrons: classe Vector (iii)

```
template<typename T>  // external definition of = operator
Vector<T>& Vector<T>::operator=(const Vector<T>& v)
{ vSize = v.vSize;
  ptrT = new T[vSize];
  copy(v);
  return *this;
}

template<typename T>  // external definition of copy() function
void Vector<T>::copy(const Vector<T>& v)
{ unsigned minvSize = (vSize < v.vSize ? vSize : v.vSize);
  for (int i = 0; i < minvSize; i++) ptrT[i] = v.ptrT[i];
}

template <typename T>
class Array : public Vector<T>  // same as vector, but with
{                               // externally defined iterator range
protected:
  int iStart;
public:
  Array(int i, int j) : iStart(i), Vector<T>(j-i+1) { }
  Array(const Array<T>& v) : iStart(v.iStart), Vector<T>(v) { }
  T& operator[](int i) const { return Vector<T>::operator[](i-iStart); }
  int firstSubscript() const { return iStart; }
  int lastSubscript() const { return iStart+ this->vSize-1; }
};
```

Les patrons: classe Vector (iv)

```
int main()
{ Array<float> mx(1,3);
  mx[1] = 3.14159;
  mx[2] = 0.08516;
  mx[3] = 5041.92;
  cout << "mx.getSize() = " <<  mx.getSize() << endl;
  cout << "mx.firstSubscript() = " << mx.firstSubscript()<<endl;
  cout << "mx.lastSubscript() = " << mx.lastSubscript() << endl;
  for (int i=1; i<=3; i++)
                    cout <<"mx[" << i << "] = " << mx[i] << endl;
}
```

```
Console:
$ main
mx.size() = 3
mx.firstSubscript() = 1
mx.lastSubscript() = 3
mx[1] = 3.14159
mx[2] = 0.08516
mx[3] = 5041.92
```

Matrix: Vector of Vectors

```
//  matrixTemplate.cpp
//  Un patron de classe Matrix, derivé de J. H u b ba r d,
//  Programmation en C++,

#include <iostream>
using namespace std;
#include "Vector.h“  // same as vectorTemplate.cpp, without main()
template<typename T>
class Matrix
{
protected:
  Vector<Vector<T>*> row; // row is a vector of pointers to vectors
public:
  Matrix(unsigned r=1, unsigned c=1) : row(r)
  { for (int i = 0; i < r; i++)  // create one vector for each row
      row[i] = new Vector<T>(c); // each vector c components
  }
  ~Matrix() { for (int i=0; i < row.getSize(); i++) delete row[i]; }
  Vector<T>& operator[](unsigned i) const { return *row[i]; }
  unsigned getRows() { return row.getSize(); }
  unsigned getColumns() { return row[0]->getSize(); }
};
```

Matrix (ii)

```
int main()
{
  Matrix<int> a(2,3); //
  a[0][0] = 00; a[0][1] = 01; a[0][2] = 02;
  a[1][0] = 10; a[1][1] = 11; a[1][2] = 12;
  cout << "The matrix a has " << a.getRows()
       << " rows and " << a.getColumns() << " columns.\n";
  for (int i=0; i<2; i++)
    {
    for (int j=0; j<3; j++) cout << a[i][j] << " ";
    cout << endl;
    }
}
```

Console

```
$ main
The matrix a has 2 rows and 3 columns.
0 1 2
10 11 12
```

Patron de classe: List

```
// listTemplate.cpp, dérivé de J. H u b b a r d,
//                                Programmation en C++
// Un patron pour une classe List

#include <iostream>
#include <string>  // STL: type prédéfinit
using namespace std;

template<typename T> class List; // reference avant

template <typename T>
class ListNode
{ friend class List<T>;  // permettre acces depuis List
  // placer ici autres friend tels que: friend class ListIter<T>;
  protected:
    T data;           // champs de données
    ListNode* next;  // prochain element de liste
  public: // constructeur: nouveau noeud contient t, pointe sur p
    ListNode(T& t, ListNode<T>* p) : data(t), next(p) { }
};
```

Patron de classe: List (ii)

```
template<typename T> class List
{protected:
    ListNode<T>* first; //1er élément de liste
    ListNode<T>* newNode(T& t, ListNode<T>* p)  // interne
    { ListNode<T>*q = new ListNode<T>(t,p); return q; }
 public:
    List() : first(0) { }  // liste vide
    ~List()
    { ListNode<T>* temp;
      for (ListNode<T>* p = first; p; ) // traverse la liste jusqu'à NULL
           { temp = p; p = p->next; delete temp; }
    }
    void insert(T t)                    // insère t en debut de liste
    { ListNode<T>* p = newNode(t, first); first = p; }
    int remove (T& t)                  // ote 1er element de la liste, rend t par ref
    { if (isEmpty()) return 0;    // liste vide
      t = first->data;            // contenu de l'élément
      ListNode<T>* p = first;     // 1er élément à détruire
      first = first->next;     // avancer premier élément au prochain el.
       delete p; return 1;          // succes
    }
    bool isEmpty() { return (first==0); }
    void print()
    { for (ListNode<T>* p=first; p; p=p->next) cout << p->data << " -> ";
      cout << "*\n";
    }
};
```

Patron de classe: List (iii)

```
int main()
{ List<string> friends;
  friends.insert("Dupres, Jean");
  friends.insert("Novello, Alfred");
  friends.insert("Dubois, Jacques");
  friends.insert("Alberti, Solange");
  friends.print();
  string name = "Novello, Alfred";
   friends.remove(name);  //oter le 1er nom
  cout << "Removed: " << name << endl;
  friends.print();
}
```

```
Console
$ main
Alberti, Solange -> Dubois, Jacques -> Novello, Alfred ->
Dupres, Jean -> *
Removed: Alberti, Solange
Dubois, Jacques -> Novello, Alfred -> Dupres, Jean -> *
```

Patron d'itérateur: classe abstraite

```
//  Iterator.h : Classe abstraite Iterator
template<typename T>
class Iterator
{
public:

  virtual void reset() =0;        // mise à zero
  virtual T operator() () =0;     // rend élément
  virtual void operator=(T t) =0;// initialise élément
  virtual int operator!() =0;     // true si
                                  // iterateur existe
  virtual int operator++() =0;    // avance itérateur
};
```

Patron d'itérateur

```
//  iteratorList.c, derivé de J. H u b b a r d, Programming in C++
//  Classe patron itérateur de liste (patron List)
#include <iostream>
#include <string>
using namespace std;
#include "List.h"  // identique à listTemplate.cpp, sans main()
#include "Iterator.h"  // classe abstraite
template<typename T>

class ListIter : public Iterator<T>  // iterateur pour liste
{protected:
  ListNode<T>* current; // pointeur noeud courant
  ListNode<T>* previous; // pointeur noeud précédent
  List<T> & list;         // liste traversée;  List<T> & list: par reference
public:                   // list initialisée lors de sa création
  ListIter(List<T>& myList) : list(myList) { reset(); } //constr.
  virtual void reset() { previous = NULL; current = list.first; }
  virtual T operator()() { return current->data; }
  virtual void operator=(T t) { current->data = t; }
  virtual int operator!() // determine whether current exists
  { if (current == NULL) // reset current pointer
        if (previous == NULL) current = list.first;
        else current = previous->next;
    return (current != NULL); // returns TRUE if current exists
  }
```

Patron d'itérateur (ii)

```
 virtual int operator++() // advance iterator to next item
 {  if (current == NULL)   // reset current pointer
      if (previous == NULL) current = list.first;
      else current = previous->next;
    else // advance current pointer
   {previous = current; current = current->next;
   } return (current != NULL);
 }

 void insert(T t) // insert t after current item
 { ListNode<T>* p = list.newNode(t,0);
    if (list.isEmpty()) list.first = p;
    else { p->next = current -> next; current -> next = p; }
 }

 void remove() // remove current item
 { if (current == list.first) list.first = current->next;
    else previous->next = current->next;
    delete current; current = 0;
 }
};
```

Patron d'itérateur (iii)

```
int main()
{ List<string> friends;
  ListIter<string> iter(friends);
  iter.insert("Dupres, Jean");      ++iter;
  iter.insert("Novello, Alfred");   ++iter;
  iter.insert("Dubois, Jacques");   ++iter;
  iter.insert("Alberti, Solange");  ++iter;
  friends.print();
  iter.reset();
  ++iter; iter = "PremEnPlus, Xenon";
  iter.insert("SecEnPlus, Rosa"); ++iter;
  friends.print();
  ++iter; cout<< iter()<< endl; iter.remove();
  friends.print();
}
```

Console

```
$ main
Dupres, Jean -> Novello, Alfred -> Dubois, Jacques -> Alberti, Solange -> *
Dupres, Jean -> PremEnPlus, Xenon -> SecEnPlus, Rosa -> Dubois, Jacques -> Alber
ti, Solange -> *
Dubois, Jacques
Dupres, Jean -> PremEnPlus, Xenon -> SecEnPlus, Rosa -> Alberti, Solange -> *
```

Standard Template Library (STL)

Classes conteneurs générales:

vector: tableau généralisé à accès indexé

list: permet insertions et suppressions de manière efficace

map: structure d'accès "clef - donnée"

dequeue: insérer et extraire éléments aux deux bouts, accès indexé

Type dérivés

stack : dernier entré -premier sorti

queue: insérer par back() et extraire par front()

STL: accès patrons standard

p, i, j: itérateurs, t: type passé comme paramètre

a.insert(p, t) : insère copie de *t* avant *p*

a.insert(p, i, j) :copie élém. *i* à *j*-1 devant *p*

a.erase(i): supprime élément *i*

a.erase(i, j) :supprime élem *i* à *j*-1

i=a.begin() : *i* pour accéder au premier élément

j=a.end() : *j*-1 pour accéder au dernier élément

t=a.front() : affecte à *t* premier élément de *a*

t=a.back() : affecte à *t* dernier élément de *a*

n=a.size() : affecte à *n* la taille de *a*

a.empty() : rend 1 si *a* est vide

a.swap(b) : Echange contenus de *a* et *b*

STL: vector<int> : indexe, itérateur

```
//stlVector.cpp
#include <vector>
#include <string>
#include <stdio.h>
void exNumber()
{  std::vector<int> v;
   for(int i=0;i<10;++i)v.push_back(i);
   // Print vector size (cast from size_t to int)
    printf("Vector contains %d elements: ",(int)v.size());
   // Use position in vector to obtain content
   for(int i=0;i<10;++i) printf("%d ",v[i]);
   printf("\n");
   // Use an iterator to go through the vector
   // Iterators must be dereferenced to obtain the value (as with
       pointers)
   for(std::vector<int>::iterator it=v.begin();it!=v.end();++it)
       printf("%d ",*it); printf("\n");
}
Console
Vector contains 10 elements: 0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```

STL: vector<string> : itérateur

```
void exString()
{ std::vector<std::string> v;
  v.push_back(std::string(“premier"));
  v.push_back(“deuxieme"); // Cast of char* to std::string has
       been overloaded
  v.push_back(“troisieme");
  // Remove the second element
  // erase() takes an iterator as a parameter
  v.erase(v.begin()+1);
  // Print last element
  // A char* form of a string is obtained using c_str()
  printf("Last element: %s\n",v.back().c_str());
  // Remove last element
  v.pop_back();
  // Print last element
  printf("Last element: %s\n",v.back().c_str());
}
```

Console

```
Last element: troisieme
Last element: premier
```

STL: vector<string> : itérateur

```
void exStringBis()
{  std::vector<std::string> v;
  v.push_back("premier");
  v.push_back("deuxieme");
  v.push_back("troisieme");
  // Print all elements and remove "deuxieme"
  for(std::vector<std::string>::iterator it=v.begin();
                                        it!=v.end(); ++it)
  { if((*it)=="deuxieme")
    { it--; // Decrement iterator to prevent it from being
            // invalidated
      v.erase(it+1);
    }
    else printf("%s\n",it->c_str());
  }
}
```

Console

```
premier
troisieme
```

std::vector< std::vector<int>>

```
void exMatrix()
{ std::vector<std::vector<int> > vv; // 2D vector (matrix)
  for(int i=0;i<2;++i)
  { vv.push_back(std::vector<int>());
    vv[i].push_back(1+10*(i+1));
    vv[i].push_back(2+10*(i+1));
  }
  printf("(0,0)= %d ",vv[0][0]); // Print element (0,0)
  // Print element (1,1), other form
  printf(" (1,1)=%d\n",vv.back().back());
  // Print all elements using iterators
  for(std::vector<std::vector<int> >::iterator
      it=vv.begin();it!=vv.end();++it)
  {  for(std::vector<int>::iterator it2=it->begin();
           it2!=it->end();++it2 ) printf("%d ",*it2);
     printf("\n");
  }
}
```

Console

```
(0,0)= 11  (1,1)=22
11 12
21 22
```

STL std::list<T> (option)

```
//stlList.cpp
#include <list>
#include <stdio.h>
class Pair
{private: int x,y;
 public:
  Pair(int x,int y) { this->x=x; this->y=y; }
  void print() { printf("(%d,%d)",x,y); }
  void swap() { int tmp=x; x=y; y=tmp; }
};

void ex_Pairs_Pointers()
{ std::list<Pair> pairs;
  pairs.push_back(Pair(3,4));
  pairs.push_front(Pair(1,2)); // There is no indexing possibility
  for(std::list<Pair>::iterator it=pairs.begin();it!=pairs.end();++it)
    it->print(); printf("\n"); // Do the same with a list of pointers
  std::list<Pair *> pairStars;
  pairStars.push_back(new Pair(3,4));
  pairStars.push_front(new Pair(1,2));
  for(std::list<Pair *>::iterator it=pairStars.begin();
                                  it!=pairStars.end(); ++it)
    (*it)->print(); printf("\n");
}
```

Console

```
(1,2)(3,4)
(1,2)(3,4)
```

STL: std::list<T> (option)

```
void ex_Swap_Pointers()
{ // Elements are stored by value
  std::list<Pair> pairs;
  std::list<Pair *> pairStars;
  Pair p1(1,2); Pair *p2=new Pair(3,4);
  pairs.push_back(p1); // A copy of p1 is stored
  pairStars.push_back(p2); // Copy of pointer stored
  // Front returns a reference to the first element
  pairs.front().swap();
  pairStars.front()->swap();
  printf("p1="); p1.print();// Pas swap, car copie de
                            // l'objet dans list
  printf(" pairs.front ="); pairs.front().print();
       // p2: Swapped: car copie du pointeur dans list
  printf(" p2=");p2->print();
}
Console
p1=(1,2) pairs.front =(2,1) p2=(4,3)
```

std::map<int,std::string>

```
//stlMap.cpp
#include <map>
#include <iostream>
void accessByKey()
{
 std::map<int,std::string> codes;
 // Map maintains a tree with all values already
 // sorted. It is therefore not possible to add
 // elements at the front or at the back.
 codes.insert(std::pair<int,std::string>(1211,"Genève"));
 codes.insert(std::pair<int,std::string>(1004,"Lausanne"));
 codes.insert(std::pair<int,std::string>(8020,"Zürich"));
 // Print map content (it is sorted)
 for(std::map<int,std::string>::iterator
       it=codes.begin();it!=codes.end();++it)
   printf("%d %s\n",it->first,it->second.c_str());
 // A search for a particular key returns an iterator
 // it->first yields the key and it->second yields the string
 std::map<int,std::string>::iterator it=codes.find(1211);
 if(it!=codes.end())
  std::cout << it->second << std::endl; // No need for c_str()
}
```

Console

```
1004 Lausanne
1211 Genève
8020 Zürich
Genève
```

std::map<int,std::string>

```
typedef std::pair<int,std::string> codeCityPair;
typedef std::map<int,std::string>::iterator mapIterator;

void simplAccessByKey()
{// Typedefs make life simpler
 std::map<int,std::string> codes;
 codes.insert(codeCityPair(1211,"Genève"));
 codes.insert(codeCityPair(1004,"Lausanne"));
 // Remove city with code 1211
 mapIterator it=codes.find(1211);
 codes.erase(it);
 for(mapIterator it=codes.begin();it!=codes.end();++it)
  std::cout << it->first << " " << it->second <<
      std::endl;
}
```

Console

```
1004 Lausanne
```