IFT159
Analyse et programmation
Thème 11 — Conclusion

Gabriel Girard

Département d'informatique
UNIVERSITÉ DE SHERBROOKE

3 décembre 2015

Thème 11 — Conclusion

Conclusion

Généralité

- Composants d'un ordinateur
- Résolution de problème
- Abstraction

Conclusion

Analyse et Conception

- Analyse et conception fonctionnelle (décomposition selon le traitement)
- Analyse et conception objet
- Abstraction procédurale
- Abstraction de données
- Analyse et conception descendante
- Analyse et conception ascendante
- Raffinement successif

Analyse

- Données en entrée
- Données en sortie
- Formules
- Constantes
- Identification des types de données utiles à la résolution du problème
- Identifications des contraintes

Conception

- Diagramme structurel pour exprimer d'abord la décomposition
- Algorithme par pseudo-code très abstraits ne s'attardant pas à l'implantation.
- UML
- Évaluation d'algorithme (complexité algorithmique – O)

Conclusion

Concepts généraux de programmation

- Programmation structurée
  - séquence
  - sélection
  - itération
- Programmation modulaire
  - un programme est composé de petits modules courts, avec un degré élevé de cohésion et un seul point d'entrée et de sortie.
  - chaque module est conçu pour être codé et testé séparément.

Mise au point

La mise au point consiste à s'assurer que le programme répond aux spécifications. On s'assure qu'il est fiable et correct. On s'assure qu'il s'exécute dans les délais et l'espace prescrits.

- Outils (debugger interactif ou trace) ;
- Mise au point ascendante ;
- Mise au point descendante.
- *Preuve de programme* (pas vu)

Conclusion

Langage C++

- Structure d'un programme C++
  (Main, fonctions, blocs)
- Types de base (int, float, char) : définition et utilisation
- Expressions arithmétiques et logiques
- Les fonctions
  - ligne prototype (déclaration)
  - définition (implantation)
  - passage de paramètres

Langage C++

- Énoncés
  - affectation ;
  - sélection : `if` et `switch` ;
  - itération : `while`, `for`, `do-while` et récursivité ;
- Concept de tableau (définition, utilisation)

Langage C++

- Déclaration de types
  - type enuméré (`enum`) ;
  - enregistrement (`struct`) ;
  - types abstraits de données (`class`) :
    - encapsulation (private, public) ;
    - définition et implantation ;
    - fonctions et données membres ;
    - constructeurs et destructeurs ;
    - surcharge d'opérateurs ;
    - ...

Reste à voir

- Spécification (génie logiciel) ;
- Méthodes d'analyse et de conception pour des systèmes de grande taille (génie logiciel) ;
- Notions plus avancées d'algorithmiques.
- Structures de données de bases ;
- Autres langages de même catégorie pour bien comprendre les concepts (Pascal, Modula, Assembleur ...) ;
- Analyse, conception et implantation avec des langages d'une autre catégorie (fonctionnel et éventuellement logique).

Reste à voir

- Notions avancées de programmation et meilleure maîtrise du langage.
  - programmation système et les opérateurs binaires ;
  - les pointeurs et l'allocation dynamique (new, delete)
  - l'héritage ;
  - le polymorphisme ;
  - programmation générique
  - le traitement des erreurs (try, throw, catch, assert)
  - les union
- La programmation par événements
- La programmation orientée objet.

Programmation système et opérateurs binaires

- &, |, ^ , $\ll, \gg, \sim$

```
....
x =  4 << 2; // x = 16
x =  7  & 3; // x = 4
x =  4  | 3; // x = 7
x = 64 >> 4; // x = 4
x = ~4 ;     // x = -5
...
```

Pointeurs et allocation dynamique

Les opérateurs pertinents sont :

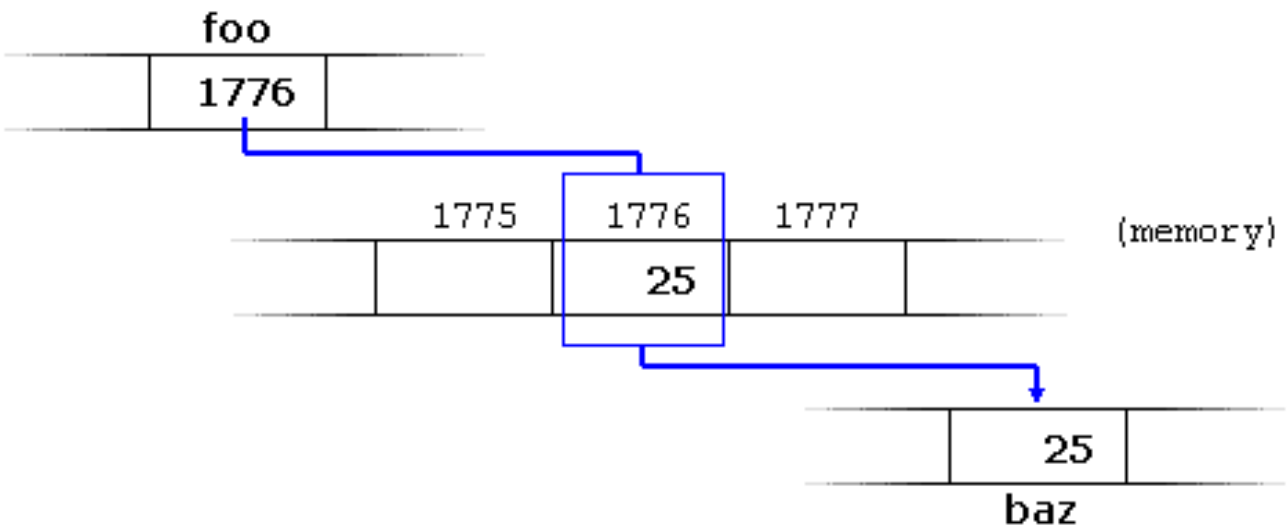
- `&` : "adresse de"
- `*` : "valeur pointée par"

```
int variable1 = 25;
int *ptr1 = &variable1;
cout << *ptr1 << endl;    // imprime 25
--------

double* valeur = new double;
*valeur = 29494.99;
cout << "Value of pvalue : " << *valeur << endl;
--------

int * tab = new int[10];
p[5] = 33;
```

Pointeurs et allocation dynamique

```
myvar = 25;
foo = &myvar;
baz = *foo;
```

```
myvar == 25
&myvar == 1776
foo == 1776
*foo == 25
```



L'héritage ;

```
class Forme
{
   public:
      void setLargeur(int w)
      {
         largeur = w;
      }
      void setHauteur(int h)
      {
         hauteur = h;
      }
   protected:
      int largeur;
      int hauteur;
};
```

L'héritage ;

```
// Classe derivee
class Rectangle: public Forme
{
   public:
      int calculerSurface()
      {
         return (largeur * hauteur);
      }
};
```

L'héritage ;

```
int main(void)
{
   Rectangle Rect;

   Rect.setLargeur(5);
   Rect.sethauteur(7);

   cout << "Surface du rectangle =  " << Rect.calculerSurface()
        << endl;

   return 0;
}
```

Le polymorphisme

- Unifier les fonctions disponibles pour un type général
- Utilise l'héritage
- Spécialisation de la fonction au niveau de l'enfant

Le polymorphisme

```
class Forme
{
   public:
      // methode abstaite
      virtual int calculerAire() = 0;
};
```

Le polymorphisme

```
class Rectangle: public Forme
{
   public:
      int calculerAire() override
      {
         return (largeur * hauteur);
      }

      void setLargeur(int w)
      {
         largeur = w;
      }
      void setHauteur(int h)
      {
         hauteur = h;
      }
  protected:
      int largeur;
      int hauteur;

};
```

Le polymorphisme

```
class Cercle: public Forme
{
   public:
      int calculerAire() override
      {
         return 3.14 * pow(rayon,2);
      }
   private:
      int rayon;
};
```

Le polymorphisme

```
int main(void)
{
   Rectangle Rect;
   Cercle cercle;

   cout << "Surface du rectangle =  " << Rect.calculerAire()
        << endl;
   cout << "Surface du cercle    =  " << cercle.calculerAire()
        << endl;

   return 0;
}
```

Le polymorphisme

```
void afficherAire(Forme & forme)
{
   cout << "Surface =  " << forme.calculerAire()   << endl;
}

int main(void)
{
   Rectangle Rect;
   Cercle cercle;

   afficherAire(Rect);
   afficherAire(cercle);

   return 0;
}
```

Programmation Générique C++

```
int min(int entier1, int entier2)
{
  if (entier1 < entier2)
      return entier1;
  return entier2;
}
```

Programmation Générique C++

```
int main(void)
{
    cout << min(2,3) << endl; // affiche 2

    // demotion de double en int
    cout << min(2.3, 2.1) << endl; // affiche 2
}
```

Programmation Générique C++

- Pour chaque type, on définit une fonction
- Il peut y avoir une infinité de types
- Haut coût de maintenance

Programmation Générique C++

```
template <typename type>
type min(type element1, type element2)
{
  if (element1 < element2)
      return element1;
  return element2;
}
```

Programmation Générique C++

```
int main(void)
{
    cout << min(2,3) << endl; // affiche 2
    cout << min(2.3, 2.1) << endl; // affiche 2.1
}
```

Librairie Standard - SL

- Collection d'outils pour améliorer le temps de développement
- Implémenter par chaque compilateur
- Aucun coût de maintenance

Librairie Standard - SL

- Conteneurs
  - vector, list, array
  - map, unordered_map
- Algorithmes
  - find, find_if, find_if_not
  - swap
  - sort, is_sorted, ...
- Outils
  - chrono (temps)
  - memory (allocation dynamique)
  - thread (gestion standardisée des fils d'exécution)
- ...

Librairie Standard - std : :array

- Un contenant séquentiel
- Définit des itérateurs (begin, end, cbegin, cend)
- Remplace les tableaux primitifs

Librairie Standard - std : :array

```
int main(void)
{
    array<int, 5> tableau = {17, 42, 13, 44, 25};
    // equivalent de int tableau[] = {17, 42, 13, 44, 25}

    cout << tableau[1] << endl;
}
```

Librairie Standard - std : :array

```
//
using TableauEntier = array < int, 5 > ;
int main(void)
{
    TableauEntier tableau = {17, 42, 13, 44, 25};

    cout << tableau.size() << endl; // affiche 5
}
```

Librairie Standard - find

- Algorithme de recherche
- Utilise des itérateurs
- Retourne l'itérateur de l'élément trouvé ou la fin
- Au pire, ordre linéaire – O(n)

Librairie Standard - std : :array et find

```
int main(void)
{
   array<int, 5> tableau = {1, 2, 3, 4, 5};

   // array definit des iterateurs

   cout << boolalpha
        << std::find(tableau.begin(), tableau.end(), 3) !=
            tableau.end()
        << endl; // affiche -- true --
}
```

Librairie Standard - map

- Conteneur associatif – associe une clé à une valeur
- Pour insérer un élément, il faut associer une paire clé-valeur
- Cette association peut se faire de différentes façons
  - Par accès aux indices
  - Par le type «Pair»
  - Par la fonction make_pair

Librairie Standard - std : :map

```
#include <map>
#include <utility>
int main()
{
    map<int, string> employes;
    // Affectation avec les indices
    employes[5234] = "Mike C.";
    employes[3374] = "Charlie M.";
    employes[1923] = "David D.";
    employes[7582] = "John A.";
    employes[5328] = "Peter Q.";

    cout << "employes[3374]=" << employes[3374] << endl << endl;
    cout << "Map size: " << employes.size() << endl;
    for( map<int,string>::iterator ii=employes.begin();
         ii!=employes.end(); ++ii)
    {
        cout << (*ii).first << ": " << (*ii).second << endl;
    }
}
```

Librairie Standard - std : :map

```
...
int main()
{
    map<string, int> employes;

    // Affectation avec des indices
    employes["Mike C."] = 5234;
    employes["Charlie M."] = 3374;

    // Affectation avec le insert() et le type STL pair
    employes.insert(std::pair<string,int>("David D.",1923));

    // Affectation avec le insert() et la fonction "make_pair()"
    employes.insert(std::make_pair("Peter Q.",5328));

    .....

}
```

Librairie Standard - std : :map et make_pair

```
int main(void)
{
    // la cle est une chaine de caracteres
    // la valeur est un entier
    map<string, int> carte;

    carte.insert(make_pair("hello", 4));
    carte.insert(make_pair("world", 2));

    // on recherche sur la cle
    cout << carte["hello"] << carte["world"] << endl;
}
```

Librairie Standard - chrono

- Algorithmes reliés au temps (secondes, minutes, heures, ...)
- Définit des opérations de bases
- Utile pour calculer le temps (exemple : tp1)

Librairie Standard - chrono

```
int main(void)
{
    // deduction automatique de type
    auto temps1 = chrono::seconds(10);
    auto temps2 = chrono::minutes(10);

    auto tempsTotal = temps1 + temps2;
    cout <<
        chrono::duration_cast<chrono::seconds>(tempsTotal).count()
         << endl;
}
```

Librairie Standard - thread

- Algorithmes à un fil d'exécution
- Facilite la multiprogrammation
- Accès à this_thread qui représente le fil courant

Librairie Standard - thread

```
int main(void)
{
    auto aDormir = chrono::seconds(10);
    this_thread::sleep_for(aDormir);
}
```

Quelques langages

- Python
- Java
- C, C#, Objective-C
- Matlab
- PhP, Ruby, Javascript, Nodejs
- Go
- Haskell, Lisp, D, R

Python : interpréteur

```
> python
.....

>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5.0*6) / 4
5.0
>>> 8 / 5.0
1.6
>>> 5 ** 2  # 5 squared
25
>>> 2 ** 7  # 2 to the power of 7
128
```

Python : interpréteur

```
>>> the_world_is_flat = 1
>>> if the_world_is_flat:
...     print "Be careful not to fall off!"
...
Be careful not to fall off!
```

Python : programme

```
# Fibonacci series:
# the sum of two elements defines the next
a, b = 0, 1
while b < 10:
      print b
      a, b = b, a+b
...
1
1
2
3
5
8
```

Python : programme

```
x = int(raw_input("Please enter an integer: "))
if x < 0:
     x = 0
     print 'Negative changed to zero'
 elif x == 0:
     print 'Zero'
 elif x == 1:
     print 'Single'
 else:
     print 'More'
...
Please enter an integer: 42
More
```

Python : programme

```
# Measure some strings:
words = ['cat', 'window', 'defenestrate']
for w in words:
    print w, len(w)
...
cat 3
window 6
defenestrate 12
```


I
LOVE
PARIS IN THE
THE SPRINGTIME