Chapitre 2

Alternatives

Introduction

Les instructions sont exécutées les unes à la suite des autres. À la fin de chaque instruction exécutée, on passe à la suivante, et ainsi de suite sans possibilité de revenir en arrière (au moins de prime abord). Considérons, par exemple, les instructions suivantes :

```
1 int b = 5;
2 int a = b;
```

Nous pouvons dessiner le graphe de séquence suivant :

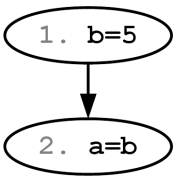

FIGURE 2.1 – Rappels sur la séquentialité

Dans l'exemple de la Figure 2.1, on affecte la valeur `5` à la variable `b`, puis on affecte le contenu de la variable `b` à la variable `a`.
Remarque : L'affectation de la ligne 2 **écrase le contenu** de la variable `a` mais ne modifie pas le contenu de la variable `b`.
En réalité, il est possible de quitter une séquence pour emprunter une autre séquence (un peu comme un aiguillage de train). Pour cela, il faut utiliser les **tests** et les **alternatives**.

2.1 Les tests

Qu'est ce qu'un test ?

Un test est une expression dont l'évaluation vaut *vrai* ou *faux*. En C++ ces deux valeurs possibles sont les deux constantes :

- `true` signifiant *vrai*,
- `false` signifiant *faux*.

Attention : Il faut bien respecter la casse (minuscule/majuscule) de ces constantes !
Exemples :

- L'évaluation du test "La terre est plate" vaut *false*,
- L'évaluation du test 15>2 ("*est ce que la valeur 15 est strictement supérieure à la valeur 2 ?*") vaut *true*.

Variables booléennes

Une variable booléenne est un emplacement mémoire (de type `bool` – c'est-à-dire booléen) qui permet de stocker soit la valeur `true`, soit la valeur `false`. Comme cela a été mentionné dans le cours sur la séquentialité, cet emplacement est appelé *variable* et dispose d'un nom appelé *nom de variable*, son type est booléen (`bool`).

Exemple :

```
1 bool a,b,c;   /* On déclare trois variables a,b et c de type booléens */
2 a=true;       /* On affecte à la variable booléenne a la valeur true */
3 b=false;      /* On affecte à la variable booléenne b la valeur false */
4 c= 15>2 ;
```

Comme pour les autres variables, le membre droit de l'affectation de la variable `c` est d'abord évalué. Pour l'instruction `c=15>2`, le membre droit vaut `true` et la valeur `true` est affectée à `c`.

Comparaisons

Il est possible d'utiliser les opérateurs ci-dessous pour comparer 2 expressions (de **même type** ou de **types compatibles**). :

Opérateur	Signification	Exemple
<	*strictement inférieur*	3<6 vaut true
<=	*inférieur ou égal*	6<=6 vaut true
>	*strictement supérieur*	4>16 vaut false
>=	*supérieur ou égal*	18>=2 vaut true
==	*égal*	4==5 vaut false
!=	*différent*	4!=5 vaut true

FIGURE 2.2 – Principaux opérateurs de comparaisons

Attention : Le traitement se fait en deux étapes :

1. les expressions sont évaluées,
2. les résultats sont comparés.

Exemple :

```
1 int x=20 ;
2 int y=13 ;
3 bool z;
4 z= x+3>y ;
5 affichage(z,'\n');
```

L'exécution de l'instruction de la ligne 4 du listing précédent est représenté par la Figure 2.3.

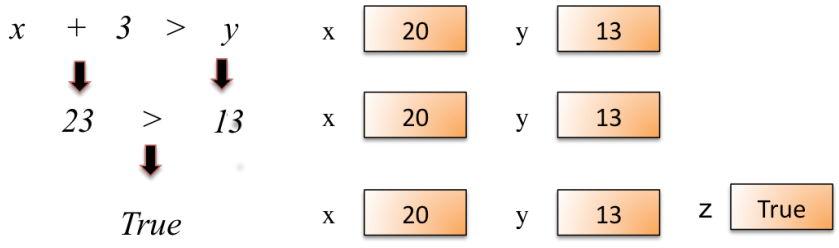

FIGURE 2.3 – Dynamique des variables lors de l'évaluation d'une comparaison

L'affichage sera donc : *true*.

Expressions booléennes

Une expression booléenne est une expression formée de variables ou constantes de type booléen reliées par des opérateurs logiques classiques : *et* (noté `&&`, et également `and`), *ou* (noté `||`, et également `or`) et *non* (noté `!`, et également `not`).

L'expression booléenne résultante est évaluée suivant la table de vérité de la Figure 2.4.

exp1	exp2	!(exp2)	exp1 && exp2	exp1 || exp2
true	true	false	true	true
true	false	true	false	true
false	true	false	false	true
false	false	true	false	false

FIGURE 2.4 – Principaux opérateurs logiques

Remarque : Dans une expression de la forme `exp1 && exp2` (resp. `exp1 || exp2`), `exp2` n'est pas évaluée si `exp1` vaut `false` (resp.`true`).

Expressions booléennes équivalentes

Ce sont des expressions qui ont toujours la même valeur de vérité :

— Expressions booléennes quelconques :

e1==true	⇔	e1
e1==false	⇔	!(e1)
!(e1 && e2)	⇔	!(e1) || !(e2)
!(e1 || e2)	⇔	!(e1) && !(e2)

— Relations avec les opérations de comparaison :

!(e1<e2)	⇔	e1>=e2
!(e1<=e2)	⇔	e1>e2
!(e1>e2)	⇔	e1<=e2
!(e1>=e2)	⇔	e1<e2
!(e1==e2)	⇔	e1!=e2
!(e1!=e2)	⇔	e1==e2

▶ Sur ce thème : **EXERCICES 1 ET 2, TD2**

2.2 Comment faire des choix au sein d'un algorithme ?

Structure de contrôle conditionnelle `if`

La structure de contrôle conditionnelle `if` permet que certaines instructions soient exécutées uniquement si une expression booléenne vaut vrai. Sa syntaxe est la suivante :

```
1 I1;
2 I2;
3 if (condition)
4 {
5   I4;
6   I5;
7 }
8 I6;
9 I7;
```

1. les instructions `I1` et `I2` (lignes *1* et *2*) sont exécutées ;
2. puis, si `condition` est vérifiée (évaluation de la condition à `true`), alors les instructions des lignes `I4` et `I5` (lignes *5* et *6*) sont exécutées ;
3. dans tous les cas, le programme exécute les instructions des lignes `I6` et `I7` (lignes *8* et *9*).

Le traitement de la structure `if` lors de l'exécution du programme est représenté dans la Figure 2.5. Les numéros de la figure correspondent aux numéros de lignes du listing de code précédent.

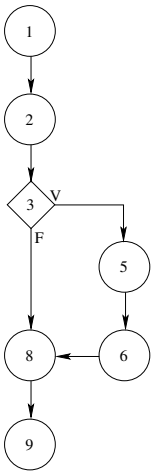

FIGURE 2.5 – Principe de la structure `if`

On dit que :

— l'exécution de la séquence des lignes I4 et I5 est *conditionnelle* car elle **dépend de l'évaluation de la condition**.

— l'exécution de la séquence des lignes I1 et I2, et celui de I6 et I7 est *inconditionnelle* car elle **ne dépend de l'évaluation d'aucune condition**.

Remarques :

— La `condition` associée au `if` doit être **entre parenthèses**.

— Les *accolades* insérées au début de la ligne 4 et au début de la ligne 7 délimitent le bloc d'instructions conditionnelles et sont obligatoires dès que ce bloc contient plus d'une instruction. Il peut contenir autant d'instructions que nécessaire.

— Toute variable créée au sein d'un bloc n'existe que dans ce bloc et est détruite une fois exécutées les instructions de ce bloc.

Exemple :

```
1  int main()
2  {
3    int temperature ;
4    int pression=100 ;
5    affichage("Indiquez la temperature\n");
6    saisie(temperature);
7    if (temperature > 55)
8    {  // debut du bloc d'instructions
9      affichage("alerte\n");
10     pression = pression - 5; // fin du bloc d'instructions
11   }
12   affichage(pression,'\n');   //cette instruction ne fait pas partie du "if"
13   return 0;
14 }
```

Dans cet exemple, le programme affichera à l'écran le message `alerte` et diminuera la pression de 5 uniquement si la température saisie par l'utilisateur est strictement supérieure à 55.

Structure de contrôle conditionnelle `if, else`

Cette structure de contrôle permet d'exécuter une séquence d'instructions si une *condition* est remplie. Cependant, si cette condition n'est pas remplie, une autre séquence d'instructions est exécutée.

```
1  if (condition)
2  {
3    I1;
4    I2;
5  }
6  else
7  {   // !(condition)
8    I3;
9    I4;
10 }
11 I5;
12 I6;
```

1. Si la condition `condition` est vérifiée (évaluation de la condition à `true`) **alors** les instructions `I1` et `I2` de (lignes *3* et *4*) sont exécutées **sinon** (c'est-à-dire si la condition est évaluée à `false`) la séquence d'instructions `I3` et `I4` (lignes *8* et *9*) est exécutée ;
2. Dans tous les cas, le programme exécute ensuite les instructions `I5` et `I6` (lignes *11* et *12*).

Le traitement de la structure `if, else` lors de l'exécution du programme est représenté dans la Figure 2.6. Les numéros de la figure correspondent aux numéros de lignes du listing précédent.

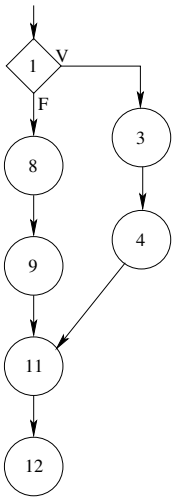

FIGURE 2.6 – Principe de la structure `if, else`

Exemple :

```
1  int main()
2  {
3    int temperature ;
4    affichage("Indiquez la temperature\n");
5    saisie(temperature);
6    if (temperature > 55)
7      affichage("alerte\n");
8    else
9      affichage("Tout va bien au niveau de la température\n");
10   return 0;
11 }
```

Dans cet exemple, le programme affichera à l'écran le message `alerte` si la température saisie par l'utilisateur est strictement supérieure à 55. Dans le cas contraire, il affichera à l'écran le message `Tout va bien au niveau de la température`.

▶ Sur ce thème : **Exercices 3, Questions 3.1 à 3.3, 4 et 5, TD2**

Structures de contrôle conditionnelles imbriquées

Les alternatives peuvent être imbriquées pour exprimer des choix "complexes" et exclusifs les uns des autres, ce qui permet d'affiner le traitement selon un contexte donné.

L'instruction `else if` (sinon si), permet d'exécuter un bloc d'instructions si la condition associée est évaluée à `true`. Cependant, cette condition ne sera évaluée que si toutes les conditions du `if` et des `else if` précédents ont été évaluées à `false`. Dans une structure conditionnelle, on peut avoir autant d'instructions imbriquées `else if` que nécessaire. Le traitement de la structure `if, else if, else` lors de l'exécution du programme est représenté dans la Figure 2.7. Les numéros de la figure correspondent aux numéros de lignes du listing de code suivant.

```
1  I1;
2  if (condition1)
3    I2;
4  else if (condition2)
5    I3;
6  else if (condition3)
7    I4;
8  else
9    I5;
10 I6;
```

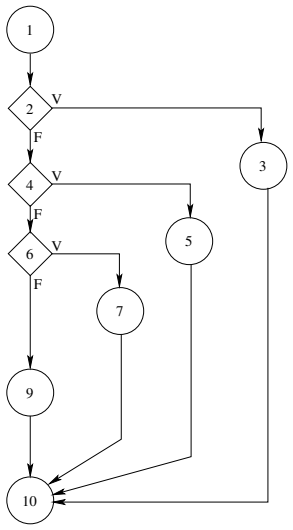

Figure 2.7 – Structures conditionnelles emboîtées

Remarques :

— les conditions doivent être entre parenthèses.

— Le `else` se rapporte toujours au `if` le plus proche qui n'est pas encore associé à un `else`.

— Quand il y a plus d'une instruction dans un bloc, les accolades sont obligatoires.

Dans l'exemple suivant, un message approprié est affiché suivant la valeur de la température. Il y a trois messages disctincts en fonction de la situation : `danger d'explosion`, `alerte`, `régime normal`.

```
1 int main()
```

```
2  {
3    int temperature ;
4    affichage("Indiquez la temperature\n");
5    saisie(temperature);
6    if (temperature > 75)      //  temperature > 75
7      affichage("danger d'explosion\n");
8    else if (temperature > 55) // 55 < temperature <= 75
9      affichage("alerte\n");
10   else                       // temperature <= 55
11     affichage("régime normal\n");
12   return 0;
13 }
```

► Sur ce thème : **Exercices 3, Questions 3.4 à 3.6, 6, 7 et 8, TD2**

TD2 : Alternatives

✓ Exercice 1 : Les variables booléennes*

Question 1.1 : [Opérations sur les booléens] Que fait le programme suivant ? Suivre le contenu des variables.

```
int main()
{
  bool flag = false;
  flag = !flag;
  flag = !flag;
  flag = !flag;
  flag = !flag;
  flag = !flag;
  return 0;
}
```

Question 1.2 : [Évaluation d'expressions booléennes (1)] Prévoir les valeurs de la variable booléenne `test` dans l'algorithme suivant :

```
int main()
{
  int x = 12;
  bool test;

  test = x>12;
  test = x!=9;
  return 0;
}
```

Important : Vous prendrez soin de suivre pas à pas l'évolution des variables.

Question 1.3 : [Évaluation d'expressions booléennes (2)]
Prévoir les valeurs de la variable booléenne `test` dans l'algorithme suivant :

```
int main()
{
  int x = 12;
  bool test;

  test = x<11 || (x>40 && x<100);
  test = !(x>10 && x<=12) && x%2==0;
  test = x>=10 && test;
  return 0;
}
```

Important : Vous prendrez soin de suivre pas à pas l'évolution des variables.

✓ Exercice 2 : Test de valeurs entières**

1. Saisir une valeur entière. Si cette dernière est paire et positive ou si cette valeur est impaire et comprise entre 5 (inclus) et 25 (inclus), alors l'expression booléenne vaut `true` sinon `false`. Afficher le résultat.

   Remarque : Il existe un opérateur appelé *modulo* dont le symbole est `%` qui permet de calculer le reste de la division euclidienne. Par exemple, `3%2` vaut 1.
2. Donner un jeu d'essai (4 tests significatifs) et prévoir les résultats.

✓ Exercice 3 : Tests*

Remarque : Les tests `if(a==true)` et `if(a)` sont équivalents. La deuxième formulation permet d'alléger l'écriture de la condition.

Question 3.1 : [Tests élémentaires] Quelle différence y a-t-il entre les deux morceaux de programme suivants ?

```
int main()
{
  bool a;

  a=5<2;
  if (a)
    affichage("V\n");
  else
    affichage("F\n");
  return 0;
}
```

et

```
int main()
{
  bool a;

  a=5<2;

  if (a == true)
    affichage("V\n");
  if (a==false)
    affichage("F\n");
  return 0;
}
```

Question 3.2 : [De la nécessité du `else`] Le `else` est-il obligatoire après un `if` ? après un `if ...else if` ?

Question 3.3 : [Evaluation d'expressions booléennes] Considérons le code suivant.

```
int main()
{
    int a = 5;
    bool b = true;
    bool c = false;

    if (a >= 30 || 12/4 == 3)
        a = a + 25;
    else
        a = -a;
    if (b || 12/5 == 2)
       b = !(b != c);
    return 0;
}
```

- Quelle est la valeur de `a` à la fin du programme ?
- Quelle est la valeur de `b` à la fin du programme ? `true` ou `false` ?

Question 3.4 : Pour quelle(s) valeur(s) de `a` (int) l'instruction `affichage("B\n");` (qui affiche la lettre B à l'écran) est-elle executée dans chacun des cas suivants ?

```
int main()
{
  int a;
   ...

  if (a > 10)
    affichage("A\n");
  else
    affichage("B\n");
  return 0;
}
```

```
int main()
{
  int a;
   ...

  if (a > 10)
    affichage("A\n");
  else if (a > 200)
    affichage("B\n");
  return 0;
}
```

```
int main()
{
  int a;
   ...

  if (a > 10)
    affichage("A\n");
  if (a > 200)
    affichage("B\n");
  return 0;
}
```

```
int main()
{
  int a;
   ...

  if (a > 10 && a< 10)
    affichage("A\n");
  else
    affichage("B\n");
  return 0;
}
```

Question 3.5 : Pour quelle(s) valeur(s) de `a` (int) l'instruction `affichage("C\n");` (qui affiche la lettre C à l'écran) est-elle executée ?

```
int main()
{
  int a;
    ...

  if (a < 100)
    affichage("A\n");
  else if (a>=100)
    affichage("B\n");
  else
    affichage("C\n");
  return 0;
}
```

Question 3.6 : [Imbrications de `if`] Dans le programme suivant quelles instructions dépendent du premier `if` ? du deuxième `if` ?

```
int main()
{
  int a;
  int b;
    ...

  affichage("1\n");
  if (a>2)
  {
    affichage("2\n");
    if (b>=a)
      affichage("3\n");
    affichage("4\n");
  }
  affichage("5\n");
  return 0;
}
```

Qu'affiche le programme dans les différents cas suivants ?

1. quand $a = 1$ et $b = 0$
2. quand $a = 2$ et $b = 2$
3. quand $a = 3$ et $b = 0$
4. quand $a = 4$ et $b = 5$

Ⓡ Exercice 4 : Programme mystère*

Que fait le programme suivant ?

Important : Suivez l'évolution du contenu des variables sur papier :

```
int main()
{
  bool interrupteur;
  int intensite;

  interrupteur=false;
  intensite=1;
  interrupteur =false;
  intensite=0;

  interrupteur =!(interrupteur);
```

```
12    if (interrupteur)
13    {
14      affichage("lampe allumee\n");
15      intensite=intensite+1;
16    }
17    else
18      affichage("lampe eteinte\n");
19
20    interrupteur =!(interrupteur);
21    if (interrupteur)
22    {
23      affichage("lampe allumee\n");
24      intensite=intensite+1;
25    }
26    else
27      affichage("lampe eteinte\n");
28
29    interrupteur =!(interrupteur);
30    if (interrupteur)
31    {
32      affichage("lampe allumee\n");
33      intensite=intensite+1;
34    }
35    else
36      affichage("lampe eteinte\n");
37
38    interrupteur =!(interrupteur);
39    if (interrupteur)
40    {
41      affichage("lampe allumee\n");
42      intensite=intensite+1;
43    }
44    else
45      affichage("lampe eteinte\n");
46
47    interrupteur =!(interrupteur);
48    if (interrupteur)
49    {
50      affichage("lampe allumee\n");
51      intensite=intensite+1;
52    }
53    else
54      affichage("lampe eteinte\n");
55    return 0;
56  }
```

® Exercice 5 : Affichage de la valeur absolue d'un nombre*

Écrire un programme qui calcule et affiche la valeur absolue d'un nombre.

✓ Exercice 6 : Conception des tests**

Peut-on réduire le nombre de tests dans le morceau de programme suivant ? Si oui, comment ?

```
1   int main()
2   {,
3     double a;
4
5     affichage("Donnez un réel\n");
```

```
  saisie(a);

  if (a <= 10)
    affichage("A\n");
  else if (a> 10 && a <= 50)
    affichage("B\n");
  else if (a> 50 && a < 100)
    affichage("C\n");
  else if (a >=100)
    affichage("D\n");
  return 0;
}
```

Exercice 7 : Tri de 3 entiers **

Écrire un algorithme qui demande 3 nombres entiers à l'utilisateur et les range dans l'ordre croissant. Un affichage devra être prévu.

Exemple : Si à un instant donné trois variables sont initialisées telles que :

```
a=4;
b=7;
c=1;
```

alors après le traitement, les instructions suivantes :

```
affichage("a=", a," b=", b, " c=", c,"\n");
```

afficheront :

```
a=1 b=4 c=7
```

® Exercice 8 : Calcul de gabarit**

On définit le *gabarit* d'un objet en fonction de sa *taille*. Le gabarit peut prendre les valeurs `'Grand'`, `'Moyen'` ou `'Petit'` (qui sont des chaînes de caractères) selon que la *taille*, qui est un nombre entier, est respectivement supérieure ou égale à 10, comprise entre 4 (inclus) et 10 (non inclus) ou strictement inférieure à 4.

Écrire un programme qui demande à l'utilisateur la taille d'un objet et affiche après l'avoir déterminé le garabit correspondant. On s'attachera à ne pas faire de test inutile.

TP2 : Les alternatives

Méthodologie à adopter pour les travaux pratiques

1. Ouvrir le terminal :

FIGURE 2.8 – Mon terminal est ouvert

2. Créer un répertoire `m1102` dans votre `home_directory`, par exemple en utilisant la commande système `mkdir ~/m1102`

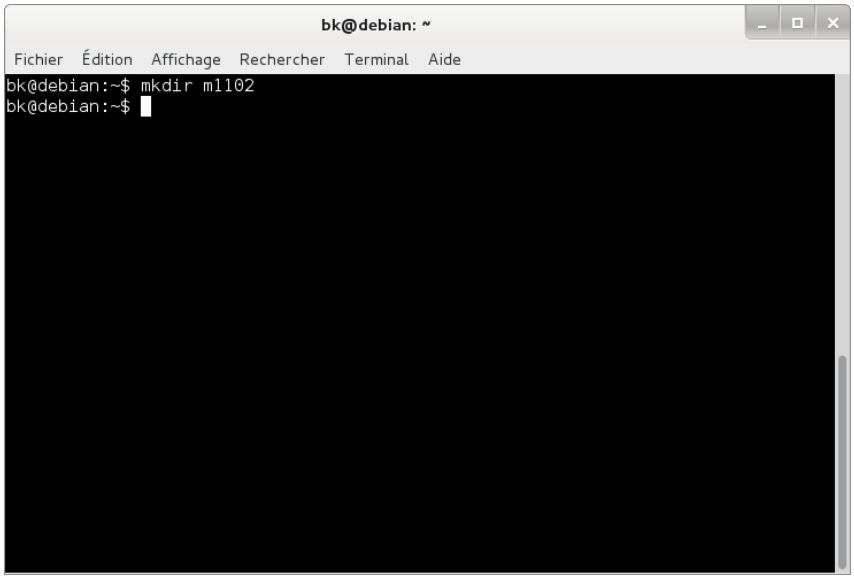

FIGURE 2.9 – Je crée le répertoire `~/m1102`

3. Créer un sous-répertoire `tp2` dans le répertoire `m1102`, par exemple en utilisant la commande système `mkdir ~/m1102/tp2`

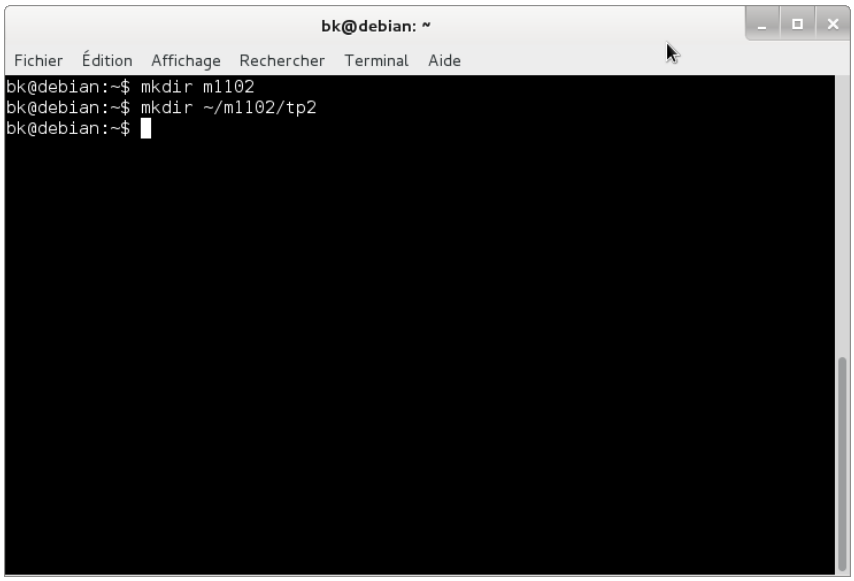

FIGURE 2.10 – Je crée le répertoire `~/m1102/tp2`

4. Recopier le fichier (script) `lancer` qui se trouve dans le répertoire `/home/TP/TPINFO/m1102_2017/` dans votre répertoire `~/m1102/tp2`

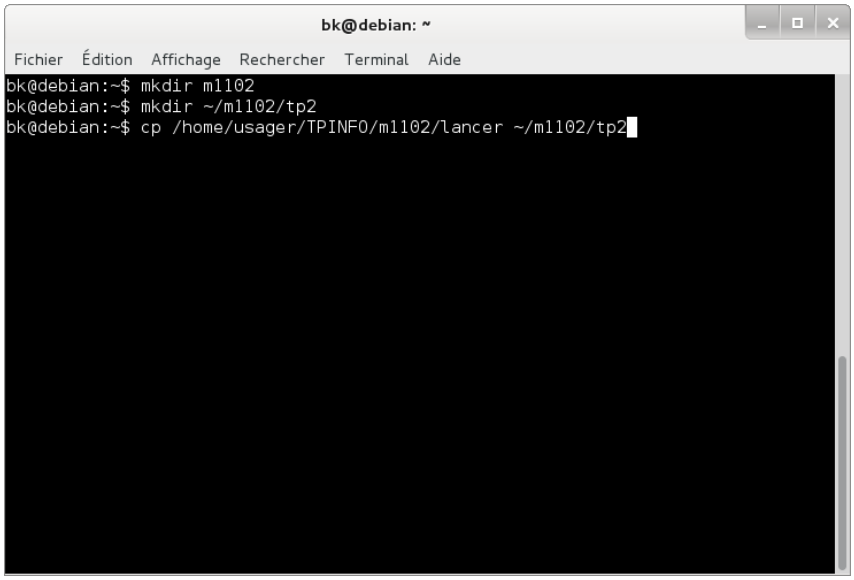

FIGURE 2.11 – Je recopie le fichier `lancer` dans mon répertoire

5. Se placer dans le répertoire `~/m1102/tp2` et vérifier que le fichier `lancer` se trouve bien dans ce répertoire, par exemple en utilisant la commande système `cd ~/m1102/tp2` puis `ls` (pour visualiser le contenu du répertoire).

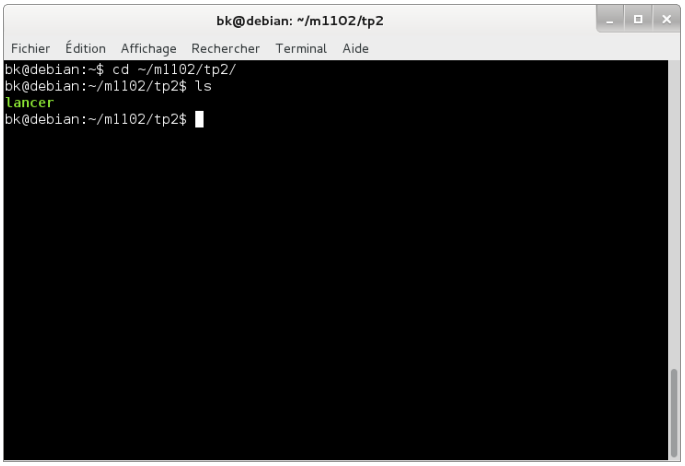

FIGURE 2.12 – Je me déplace dans le répertoire `tp2` et vérifie que le fichier `lancer` a bien été recopié

6. Pour avoir le droit d'exécuter le script `lancer`, exécuter la commande `chmod 733 lancer`.

Pour chaque algorithme que vous voudrez tester :

1. Lancer un éditeur de texte (`nedit`, `gedit`...), par exemple, avec la commande `gedit&` (pour lancer l'utilitaire `gedit` en mode multi-tâche).

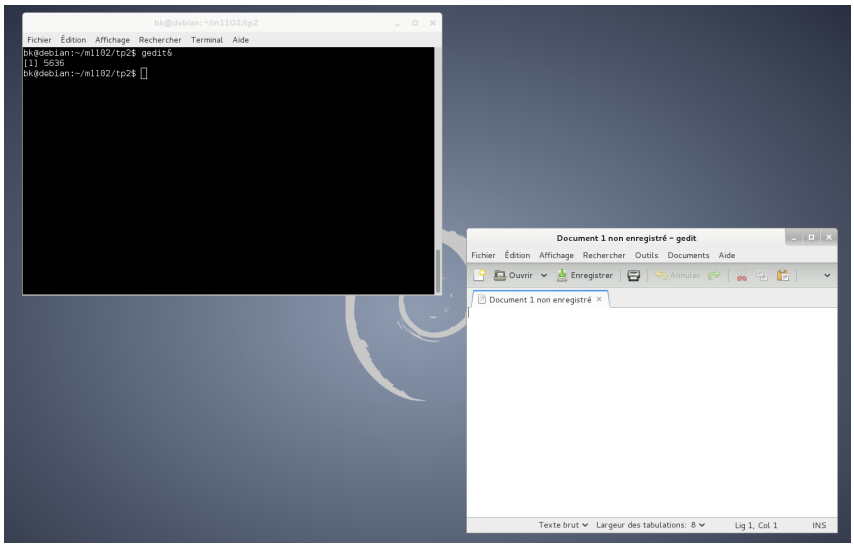

FIGURE 2.13 – Je lance l'éditeur de texte `gedit` (par exemple) en mode multi-tâche

2. Saisir votre algorithme :

```
1 int main()
2 {
3   int x;
4   x = 10;
5   affichage("la valeur de x est ", x, '\n');
6   return 0;
7 }
```

3. Sauvegarder ce dernier dans le répertoire `~/m1102/tp2`, par exemple sous le nom `exo1_tp2.cpp` ; vous veillerez à bien utiliser l'extension `.cpp` :

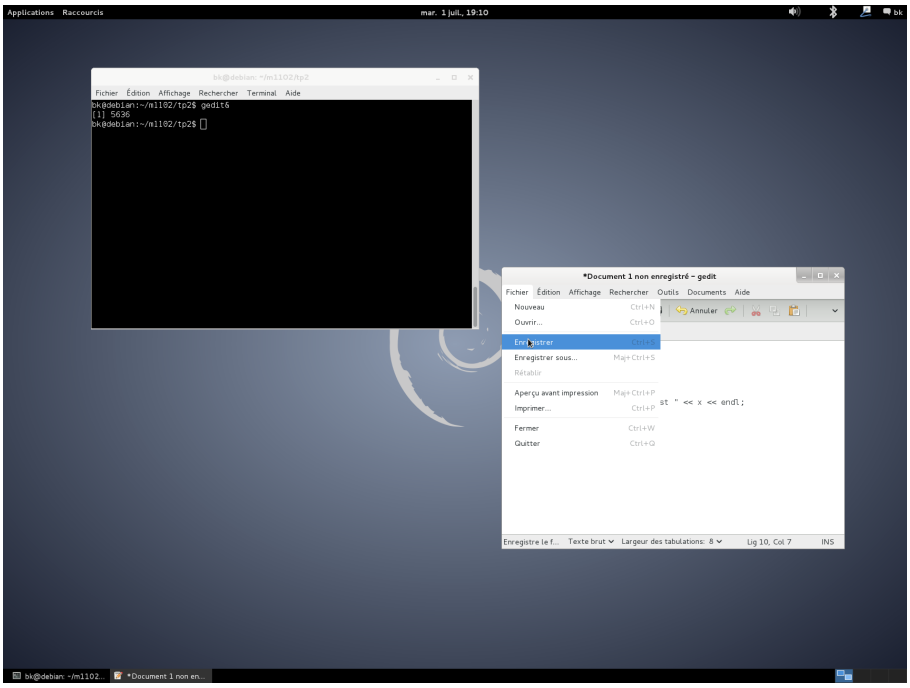

FIGURE 2.14 – Je choisis la commande `enregistrer` du menu `fichier`

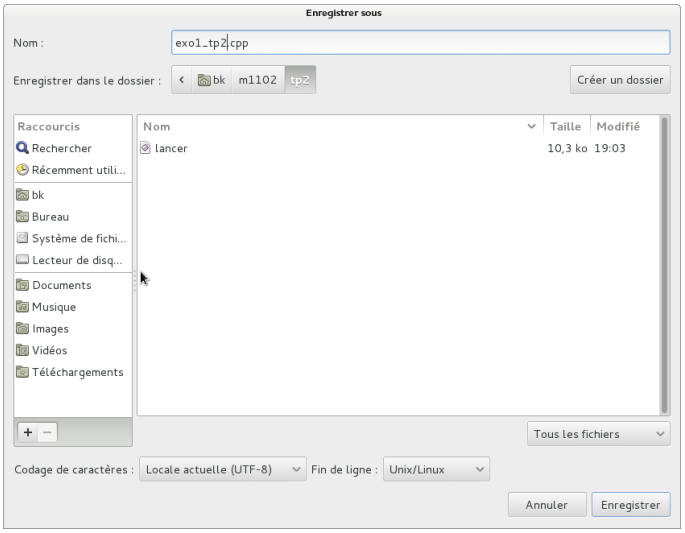

FIGURE 2.15 – Je sauvegarde l'algorithme dans le répertoire `~/m1102/tp2`

Remarque : Le code se colore, suivant les mots clefs utilisés. Il s'agit de ce qu'on appelle la **coloration syntaxique** et qui aide le programmeur à analyser la structure d'un programme (ou d'un algorithme). L'éditeur de texte détermine quelle coloration utiliser en fonction de l'extension du fichier sauvegardé ; ici `.cpp` indique que ce sera la coloration pour le C++.

4. Pour tester ce programme (ne pas fermer votre éditeur de texte), aller dans le terminal et utiliser le programme `lancer` ; par exemple : `lancer exo1_tp2.cpp` pour tester le programme `exo1_tp2.cpp`.

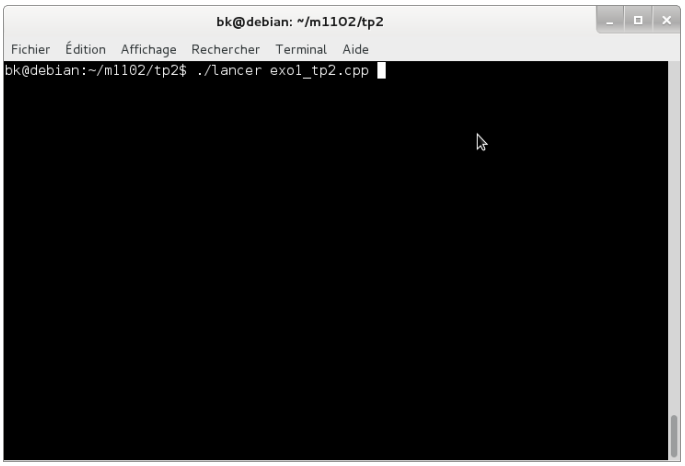

FIGURE 2.16 – Je teste mon programme

Attention : Le programme n'est pas exécuté, tant qu'il y a des erreurs dans ce dernier. Il faut alors faire des aller-retour entre la fenêtre du terminal et celle de l'éditeur de texte en**prenant soin de bien sauvegarder le programme à chaque modification de ce dernier**.

FIGURE 2.17 – Résultat du test du programme

Exercice 9 : Pair ou impair*

Écrire un algorithme qui demande un nombre entier à l'utilisateur et affiche à l'écran :
— "*Nombre pair*" si le nombre saisi est pair,
— "*Nombre impair*" si le nombre saisi est impair.

Exercice 10 : Calcul de la différence de deux nombres entiers**

Écrire un algorithme qui demande deux nombres entiers à l'utilisateur et calcule la différence du plus grand nombre avec le plus petit nombre, quel que soit l'ordre de saisie.

Exercice 11 : Facture pour la reprographie**

Un magasin de reprographie facture 0,10 Euro les dix premières photocopies, 0,09 Euro les vingt suivantes et 0,08 Euro au-delà. Ainsi, 18 photocopies coûtent 1,72 Euros (10*0,10+8*0,09). Ecrivez un algorithme qui demande à l'utilisateur le nombre de photocopies effectuées et qui affiche la facture correspondante. Vous veillerez à faire le moins possible de tests.

Exercice 12 : L'impôt au pays de *Zorglubland****

Les habitants de *Zorglubland* paient l'impôt selon les règles suivantes :
— Les hommes de plus de 20 ans paient l'impôt,
— les femmes paient l'impôt si elles ont entre 18 et 35 ans,
— les autres ne paient pas d'impôt,
— le programme demandera donc l'âge et le sexe du *Zorglubien*, et se prononcera donc ensuite sur le fait que l'habitant est imposable ou pas.

Exercice 13 : L'horloge**

Écrire un algorithme qui demande à l'utilisateur l'heure, les minutes et les secondes, et afficher l'heure qu'il sera une seconde plus tard.

Par exemple, si l'utilisateur tape 21 puis 32 puis 8, l'algorithme doit répondre : "Dans une seconde, il sera 21 heure(s) 32 minute(s) et 9 secondes".

Remarque : 0n commencera par vérifier que l'utilisateur entre une heure valide.

Exercice 14 : Racines réelles d'un polynome du second degré**

Écrire un algorithme qui calcule et affiche les racines réelles d'un polynôme du second degré.

Rappel : Ce sont les racines réelles de l'équation a*x*x + b*x + c = 0