Université de Namur

Faculté d'Informatique

www.info.fundp.ac.be

Introduction à l'algorithmique et au langage C

Formation donnée dans le cadre de l'Olympiade belge d'informatique et des cours ouverts de carnaval

Xavier Devroey
Nicolas Genon
Fabian Gilson
Seweryn Dynerowicz

mer. 13 février 2013

Agenda

- Jeudi 14/02
  - 9.00 – 10.30 introduction à l'environnement UNIX
  - 10.30 – 10.45 *break*
  - 10.45 – 12.00 introduction à l'algorithmique
  - 12.00 – 13.30 *lunch*
  - 13.30 – 15.00 introduction au langage C
  - 15.00 – 15.15 *break*
  - 15.30 – 17.00 exercices 1
- Vendredi 15/02
  - 8.30 – 10.30 gestion entrée/sortie et pointeurs en langage C
  - 10.30 – 10.45 *break*
  - 10.45 – 12.00 fin exercices 1 et exercices 2
  - 12.00 – 13.30 *lunch*
  - 13.30 – 17.00 exercices machines des be-oi '12

VOIR présentation annexe

Introduction à l'algorithmique

- Histoire
- Définitions informelles
- Concepts de bases
- Structures de données
- Structures de contrôle

Histoire

- Résolution systématique de problème
  - résolution systématique de problèmes
  - premières traces : probablement au IIIe siècle avant JC
- Quelques exemples
  - algorithme d'Euclide (plus grand commun diviseur de 2 entiers)
  - algorithme de Dijkstra (plus court chemin entre les noeuds d'un réseau)
  - résolution d'équations du deuxième degré
- Terme né au XIXe
  - Lady Ada Lovelace, assistante de Charles Babbage
  - émergence de la structure de branchement conditionnel (i.e. if)
- Un algorithme n'est pas un *programme*
  - il décrit la structure d'exécution indépendamment d'un langage
  - peut être *implémenté* dans (presque) n'importe quel langage

Définitions informelles

- Algorithme

  séquence finie d'opérations à exécuter pour obtenir la solution à un problème donné

- Langage de programmation

  langage artificiel conçu pour exprimer de manière simple, précise et complète des algorithmes

- Programme (code source)

  traduction d'un algorithme dans un langage de programmation

- Compilation

  transformation d'un programme dans un langage de programmation vers un langage directement exécutable par la machine (par ex. du code C vers du binaire)

Concepts de base

- Données et types primitifs
- Deux types de structures de base
- Structures de données
  - servent à «stocker » de l'information
    - structures *simples*
    - structures complexes *finies*
    - structures *récursives* (cfr. présentation de demain)
- Structures de contrôle
  - servent à définir le flux d'exécution d'un programme
    - séquences
    - conditions
    - boucles

Données et types primitifs

- Une donnée est caractérisée par un type
  - nombre entier
  - nombre à virgule flottante (*notation scientifique*)
  - caractère
  - booléen (oui, non)
- Les opérations possibles sur les données
  - addition, multiplication, racine,... sur entiers ou réels
  - comparaison (supérieur, inférieur, égal, différent)
    - sur certains types et pas d'autres
    - parfois dépendant du langage de programmation

Structures finies

- Constante
  - une valeur qui ne change jamais

```
CONTENANCE_VERRE = 25
```

- Variable
  - nom symbolique contenant une valeur d'un certain type (entier, par ex)

```
contenu_verre : entier
contenu_verre = 25
contenu_verre = 20
contenu_verre = 13
```

- Structure (ou *tuple*)
  - une variable contenant d'autres variables en nombre fixe

```
verres (type_verre : entier, nombre_verre : entier)
```

Structures complexes finies et récursives

- Tableaux
  - ensemble fini et indexé contenant des données d'un même type

    `tab_verre_orval[1..10]`

- Tableaux associatifs
  - ensemble fini contenant des paires (clé, valeur) de données

    `tab_verre_bière[ verre(type_verre, nombre_verre) ]`

- Liste
  - ensemble ordonné fini (ou infini) de données

    `liste_client_au_bar(1e, 2e, 3e,..., 10e)`

Structures de contrôle – séquence

- Exécution séquentielle d'actions (*instructions*)
- Définit un ordre dans les choses à faire
- Exemple, servir une bière à la pompe

```
programme sert_une_bière
début
   prends_un_verre
   ouvre_la_pompe
   remplis_le_verre
   ferme_la_pompe
fin
```

Structure de contrôle - conditions

- Effectuer différentes actions (instructions) suivant une condition mathématique oui/non

- Permet de prévoir différents cas de figure

```
programme sert_une_bière
début
   prends_un_verre
   ouvre_la_pompe
   si (fut = vide) alors
      ferme_la_pompe
      change_fut
      ouvre_la_pompe
   remplis_le_verre
   ferme_la_pompe
fin
```

Structure de contrôle - conditions

- Possibilité d'effectuer deux ensembles d'actions suivant la condition

```
programme sert_une_bière
début
   si (client_bourré) alors
      prends_un_gobelet
   sinon
      prends_un_verre
   ouvre_la_pompe
   si (fut = vide) alors
      ferme_la_pompe
      change_fut
      ouvre_la_pompe
   remplis_le_verre
   ferme_la_pompe
fin
```

Structure de contrôle – boucles (1/2)

- Répéter un ensemble d'actions tant qu'une condition est satisfaite

```
programme sert_une_bière
début
   tant que (clients_assoiffés)
      prends_un_verre
      ouvre_la_pompe
      si (fut = vide) alors
         ferme_la_pompe
         change_fut
         ouvre_la_pompe
      remplis_le_verre
      ferme_la_pompe
fin
```

Structure de contrôle – boucles (2/2)

- Répéter un ensemble d'actions sur les éléments d'une structure

```
programme sert_une_bière
début
   tant que (clients dans liste_clients)
      prends_un_verre
      ouvre_la_pompe
      si (fut = vide) alors
         ferme_la_pompe
         change_fut
         ouvre_la_pompe
      remplis_le_verre
      ferme_la_pompe
fin
```


BON APPETIT

Introduction au langage C

- Types, constantes et variables
- Structures et tableaux
- Bloc de code et *visibilité*
- Instructions de contrôle
- Programme principal et fonctions
- Paramètres des fonctions
- Entrée/sortie standard
- Mon premier programme en C
- Exercices

Types

- `short int` : entiers courts (16bits)
  - signé ou non signé
  - de 0 à 65 535 (0 à 2^{16}-1)
  - de -32 768 à 32 767 (-2^{15} à 2^{15}-1)
- `int` : entiers long (32bits)
  - signé ou non signé
  - de 0 à 4 294 967 295 (0 à 2^{32}-1)
  - de -2 147 483 648 à 2 147 483 647 (-2^{31} à 2^{31}-1)
- `char` : caractère alphanumérique
- `float` : nombres à virgule flottante
  - de -3.4×10^{38} à 3.4×10^{38}
- pas de booléen (vrai/faux), mais utilisation des `int`

Constantes et variables

- Déclaration de constantes

```
const int C_INT = 12;
const char C_CHAR = 'g';
```

ou

```
#define C_INT 12
#define C_CHAR 'g'
```

- Enumération

```
enum colors {BLUE, RED, YELLOW, GREEN};
```

  - assignation de valeurs entières à partir de 0

```
enum colors {BLUE=1, RED=2, YELLOW=3, GREEN=4};
```

- Déclaration de variable
  - le nom doit commencer par une lettre ou « _ »
  - de la forme ***type nom;***

```
int intvar; int intvar2;
char charvar; char char_var;
unsigned int unsIntVar;
```

Structures et tableaux

- Une structure complexe
  - déclaration du type

```
struct point {
   int x;
   int y;
}
```

  - déclaration de la variable et accès

```
struct point p;
p.x = 10;
p.y = 2;
```

- Tableaux
  - déclaration
  - `int tabint[10];` ou `int tabint[] = {1, 12, 43, -4};`
  - accès, les indices d'un tableau[n] vont de 0 à (n-1)

```
tabint[1] = 12;
```

Bloc de code et visibilité

- Un bloc de code est délimité par des `{ }`
  - dans une condition
  - dans une boucle
- Une variable n'est visible qu'à l'intérieur du bloc de code où elle est définie
- Variable *globale*
  - définie en tête de programme
  - accessible partout
- Variable *locale*
  - définie au sein d'un bloc de code
  - accessible uniquement dans ce bloc de code après sa déclaration
  - peut *cacher* une variable globale si elles ont le même nom

Instructions de contrôle (1/2)

- Conditions

```
if (x == 0) then
   y = 12;
else
   y = 23;
```

  - équivalent à

```
y = (x == 0) ? 12 : 23;
```

- Boucles
  - *tant que*

```
i = 0 ;
while (i < 10) {
   fait_quelque_chose
   i = i +1 ;
}
```

  - *pour toute valeur entre 2 bornes*

```
for (int i = 0 ; i < 10 ; i++){
   fait_quelque_chose
}
```

Instructions de contrôle (2/2)

- Condition plus complexe
  - alternative à des `if-then-else` imbriqués en nombre trop important

```
switch (une_variable)
{
   case une_valeur :
      code_à_exécuter si une_variable = une_valeur
      break;
   case une_autre_valeur :
      code_à_exécuter si une_variable = une_autre_valeur
      break;
   /* autres possibilités */
   default :
      code_à_exécuter si une_variable = aucune_valeur
      break;
}
```

Programme principal et fonction

- programme principal en C

```c
void main()
{
    /* du code */
}
```

```c
int main()
{
    /* du code */ return 0;
}
```

- fonction
  - sous programme
  - permet de structurer et/ou réutiliser des morceaux de codes
  - peut renvoyer une *valeur de retour*

```c
void proc ()
{
    /* du code */
}
```

```c
int proc ()
{
    /* du code */ return 10;
}
```

Paramètres des fonctions – passage par valeur

- Une fonction peut recevoir des paramètres
  - besoin de données en entrée venant du programme principal

```
int addition(int a, int b)
{
   a = a + b; return a;
}
```

- Les variables en entrée ne sont pas modifiées globalement
  - la fonction renvoie la valeur de `a`
  - la valeur de la variable utilisée par la fonction est « restaurée »

```
int x, y;
void main()
{
   x = 8; y = 6;
   addition(x, y); /* renvoie 14 */
   addition(x, y); /* renvoie 14 à nouveau */
}
```

Paramètres des fonctions – passage par référence

- Si on veut modifier les valeurs des paramètres, il faut
  - passer les paramètres par *adresse* (via l'opérateur *unaire* &)
  - les « déréférencer » (via l'opérateur unaire *) pour manipuler leurs contenus (valeurs)
  - la valeur de la variable utilisée par la fonction est alors modifiée

```
int addition(int *a, int *b)
{
   *a = (*a) + (*b); return (*a);
}
int x, y;
void main()
{
   x = 8; y = 6;
   addition(&x, &y); /* renvoie 14 */
   addition(&x, &y); /* renvoie 20 */
}
```

Entrée/sortie standard

- On peut écrire à la *sortie standard* (console) avec `printf`
  - un message (attention au `\n`)

```
printf("un message\n");
```

  - le contenu d'une variable

```
int i = 10;
printf("un message %d\n", i);
```

- On peut lire à *l'entrée standard* (clavier) avec `scanf`
  - on va lire ce qui a été tapé au clavier et le stocker dans la variable `x`

```
char x;
scanf("%c", &x);
```

- Note
  - `%d` définit le type de la variable écrite/lue (`%d` pour `int`, `%c` pour `char`)
  - plus de détails dans la partie « avancée » de demain

Mon premier programme C

- Fichier source *helloworld.c*

```
#include <stdio.h>


int main()
   {
   /* My really first program */
   printf("Hello, World! \n");
   return 0;
}
```

- Compilation dans une ligne de commande

```
gcc helloworld.c -o helloworld
```

- Exécution

```
./helloworld
```


BREAK . .

Exercices (1/3)

- ***Ex 1*** - Plus grand commun diviseur
  - pgcd (a,b) = pgcd (b, a % b) (% = reste de la division entière)
- ***Ex 2.a*** - Manipulation de tableaux
  - on additionne tous les éléments d'un tableau
- ***Ex 2.b*** - Manipulation de tableaux
  - on multiplie les éléments d'un tab. avec la somme des élé. d'un autre
  - soient `t1`, `t2` des tableaux d'entiers
  - pour tout élément de `t1`, on le multiplie par la somme des élé. de `t2`
- ***Ex 3*** - Tri des éléments d'un tableau – tri à bulles

```
procédure tri_bulle(tableau T, entier n)
    tant que échange_effectué = faux
    pour j de 0 à n – 1
        si T[j] > T[j + 1], alors
            échanger T[j] et T[j + 1]
            échange_effectué = vrai
```

Exercices (2/3)

- ***Ex 4.a*** **-** Recherche dans un tableau – recherche naïve
  - rechercher un élément dans un tableau en parcourant tous les éléments
- ***Ex 4.b*** **-** Recherche dans un tableau – recherche dichotomique
  - trier les éléments du tableau
  - on « coupe en deux » l'espace de recherche de manière successive
  - si la valeur choisie est >, on continue avec la « partie gauche »
  - si la valeur choisie est <, on continue avec la « partie droite »
- ***Ex 5*** **-** Plus longue sous-suite
  - deux tableaux tailles équivalentes contenant des caractères
  - trouver la plus longue suite de valeur commune aux deux tableaux
    - `t1=[1,2,3,4,5,6,7,8]` et `t2=[2,3,5,6,7]`
    - alors `tRes=[5,6,7]`
  - généraliser à deux tableaux de tailles différentes

Exercices (3/3)

- ***Ex 6*** **-** Codage RLE (*run lenght encoding* - compression naïve)
  - compresser un tableau avec des suites de valeurs identiques
    - ex : *AAAAAAAAZZEEEEEER* devient *8A2Z6E1R*

- ***Ex 7.a*** **-** Palindrôme
  - pour un tableau de valeur, vérifier si c'est un palindrôme ou non
  - on ne considère pas les caractères accentués (« à », « ç », etc).
  - un palindrôme = mot lisible dans les deux sens (ex : kayak)

- ***Ex 7.b*** **-** Palindrôme dans une phrase
  - détecter les palindrômes dans une phrase
  - supprimer les espaces et la ponctuation avant d'effectuer la recherche
  - vérifier si on peut extraire des palindrômes
    - ex : j'essayasse de faire du kayak en été
    - devient jessayassedefairedukayakenete
    - résultat : essayasse, kayak, ete


Merci de votre attention
Questions? Remarques?