TOPC

Techniques et Outils de Programmation en C

PLATYPUS

Interagir avec l'extérieur
Stocker de l'information

Le programme, l'utilisateur, la mémoire
Entrée, sortie, variables, types,
Déclarations et affectations

Note importante

Les informations contenus dans ce support sont susceptibles de révisions ultérieures.

Plan de la séance

- Théorie et algorithmique
  - Le programme, l'utilisateur, la mémoire
  - Début et fin
  - Entrées et sorties
  - Les variables et leur manipulation
- Langage C
  - Début et fin d'un programme
  - Entrée et sortie standards, instructions d'entrée et de sortie, stdio.h
  - Variables et types primitifs, instructions de déclaration et d'affection de variables
- Techniques et outils
  - Les options du compilateur (compilation en ligne de commande)
- Démos
  - Quelques programmes avec instructions d'entrée / sortie et affectations de variables

Théorie et algorithmique

- Le programme, l'utilisateur, la mémoire
- Début et fin
- Entrées et sorties
- Les variables et leur manipulation

Le programme, l'utilisateur, la mémoire

- Les constituants de la machine (architecture)
  - **Processeur** : il exécute les instructions
  - **Mémoire** : elle permet la stockage et la réutilisation de données
  - **Interface** : elle permet à l'utilisateur (le pilote) d'interagir avec elle
- Le programme
  - La séquence d'instructions exécutées par le processeur
    - Marquée par un début et une fin
  - Dont les instructions de contrôle de l'interface
    - Instructions d'entrée / sortie
  - Dont les instructions de contrôle de la mémoire
    - Instructions d'affectation et d'évaluation de variable

Le programme, l'utilisateur, la mémoire

- Notion d'application
  - Le programme du point de vue de l'utilisateur
  - Un ensemble de services qui lui sont rendus
  - Un service = une fonctionnalité
- L'utilisateur
  - Lance le programme
  - Interagit avec lui à travers l'interface
    - Il saisit des données (donne des informations)
    - Il reçoit des informations
  - Est notifié de la fin de son exécution et de son résultat

Structure de l'algorithme

1. L'entête (la déclaration de l'algorithme)
   - indique le nom de l'algorithme
   - ses arguments (on y reviendra plus tard)
   - son résultat éventuel (on y reviendra plus tard)
   - les variables locales qu'il utilise (on voit ça tout à l'heure)
   - le résultat qu'il produit
2. Le corps (la séquence d'instructions qui le définit)
   - commence avec **début** qui indique que le programme a été lancé
   - termine avec **fin** qui indique que le programme va s'arrêter

Exemple d'algorithme

Algorithme 2: $HowAreYou\,()$

```
// entête :
Variable locale        : nom, un texte, le nom de l'utilisateur
// corps :
début
    nom ← saisir
    afficher 'Bonjour ' + nom + ', comment allez-vous ?'
fin
```

Sa traduction en C

```c
#include <stdio.h>
void main() // programme HowAreYou
{ //début
  char nom[1024]; // nom de l'utilisateur (déclaration)
  scanf("%s", nom); // saisie du nom par l'utilisateur
  printf("Bonjour %s, comment allez-vous ?", nom);
} //fin
```

Commentaires vs. instructions

- // employé plusieurs fois dans les exemples précédents
- Commentaire
  - Commence avec // et se poursuit jusqu'à la fin de la ligne
  - En langage algorithmique comme en C
  - Non interprété (exécuté) par le processeur
  - Sert au concepteur / programmeur à documenter son algorithme / programme
    - Grandes étapes intermédiaires
    - Explicitation du principe des parties les plus techniques de l'algorithme / programme

Commentaires vs. instructions

- Exemple :
  - Langage algorithmique :

```
// Mon commentaire
Instruction
```

  - Traduction en langage C :

```c
// Mon commentaire
Instruction;
```

- En C
  - possibilité de commentaires sur plusieurs lignes
  - délimité par les deux séquences /* et */

```c
/**
 * (c) 2015 Junior@EFREI
 */
```

Entrées et sorties

- Entrée et sortie
  - **Entrée** : le flux de données venant vers l'interface (et de l'utilisateur)
  - **Sortie** : le flux de données allant vers l'interface (et l'utilisateur)
- Instructions d'entrée et de sortie
  - Ecriture sur la sortie : instruction *afficher*
    - Chaque nouvelle écriture complète les précédentes, séquentiellement
  - Lecture de l'entrée : instruction *saisir*
    - Chaque nouvelle lecture complète les précédentes, séquentiellement
    - Si aucune entrée n'est disponible
      - L'instruction lecture suspend le programme en attente de celle-ci

Entrées et sorties

- Afficher
  - Syntaxe : **afficher** *contenu*
  - Effet : affiche *contenu* sur l'interface
  - *contenu* : séquence d'expressions constantes ou évaluées
    - Expression constantes
      - On parle de littéraux
      - Ex. 'a' (un caractère), 'un texte' (chaîne de caractères) , 1 (entier), 1,234 (réel)
    - Expressions évaluées
      - A base de variables, ces expressions sont évaluées au moment de l'affichage
      - Ex : x, $x + y$, $x^2 + 2x + 1$

Entrées et sorties

- Saisir
  - Syntaxe : $x \leftarrow$ **saisir**
  - Effet : affecte à la variable *x* la valeur saisie par l'utilisateur

Algorithme: Exemples

$x \leftarrow$ **saisir**
afficher 'un texte'
afficher 'Comment vous appelez-vous ?'
$x \leftarrow$ **saisir**
afficher 'enregistrement n^o' + num + ' du ' + $date$
afficher 'Bonjour ' + nom + '. Comment allez-vous ?'
afficher x + ' + ' + y + ' = ' + $(x + y)$

Entrées et sorties

```
/*
>> Afficher :
Inclure : stdio.h
Syntaxe : printf(format, liste d'expressions évaluée);
format : ..
*/
// Exemples :
printf("un texte");
printf("Comment vous appelez-vous ? ");
scanf("%s", nom);
printf("enregistrement n°%d du %s", date);
printf("Bonjour %s. Comment allez-vous ?", nom);
printf("%d + %d = %d", x, y, x + y);
```

Les variables et leur manipulation

- Variable
  - Unité de stockage d'information référencée (**nom**)
  - Ne peut stocker que des valeurs d'une certaine nature (**type**)
  - Une nouvelle valeur stockée remplace la précédente (**état**)
- Déclaration d'une variable
  - Dans l'entête de l'algorithme, après le nom
  - Ajout d'une ligne pour chaque variable locale
  - Syntaxe : $\mathbf{Variable\ locale} \qquad : nom, \mathrm{type}, \mathrm{rôle}$
    - *nom* : référence symbolique (comme en maths)
    - *type* : nature des valeurs que peut stocker la variable (domaine de définition)
    - *rôle* : le rôle que va jouer cette variable dans le déroulement de l'algorithme

Les variables et leur manipulation

- Affectation (écrire sur une variable)
  - Stockage d'une nouvelle valeur dans une variable
  - Syntaxe : $nom \leftarrow valeur$
    - On dit aussi que la variable *nom* reçoit la valeur *valeur*
- Evaluation (lire une variable)
  - Syntaxe : nom, où *nom* est la nom de la variable à évaluer
  - Une référence à une variable est une expression simple
  - Pour une variable, être évaluée signifie simplement être lue

Les variables et leur manipulation

Algorithme: Exemples

Variable locale : x, un entier, le premier opérande
Variable locale : y, un entier, le second opérande
Variable locale : x, un entier, la somme de x et y
$x \leftarrow 2$
$y \leftarrow$ **saisir**
$z \leftarrow x + y$
afficher $x + ' + ' + y + ' = ' + z$
retourner z

Les variables et leur manipulation

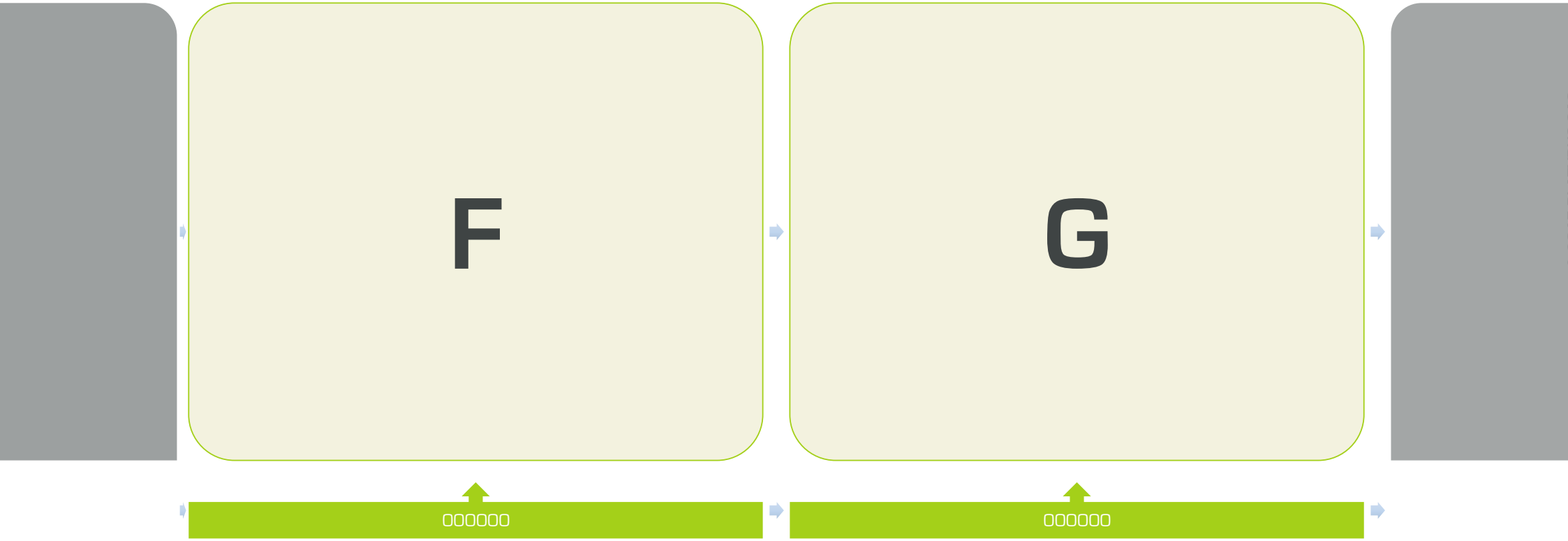
```c
// Exemples :
int x; // premier opérande
int y; // second opérande
int z; // la somme de x et y
x = 2;
scanf("%d", &y);
z = x + y;
printf("%d + %d = %d", x, y, z);
return z;
```

Travail autonome de la semaine

- Etudier les options de compilation de **gcc**
- Concerne :
  - La compilation en ligne de commande (cf. 1ère séance)
  - Le paramétrage de la compilation dans votre IDE préféré (Codeblocks, etc)

Démos

- Exemple 1
  - Saisir deux entiers *x* et *y* et afficher x + ‘ x ‘ + y ‘ = ‘ + *xy*
- Exemple 2
  - Saisir un entier *n* et afficher toutes les puissance de 2^n pour n dans [0, *n*]


F
G
000000
000000