Plan

Présentation générale du cours d'algorithmique

Principes de programmation impérative

Syntaxe du langage C

Les cours du lundi

À partir du 9 octobre,
sauf le 4 décembre (partiel)

1. Machines, algorithmes
2. Flottants
3. Entiers
4. Algorithmes de recherche
5. Graphes
6. Flots
7. Algèbre linéaire

Prérequis d'algorithmique

Pour rattraper l'option informatique des classes préparatoires. Facultatif, mais son contenu doit être su.

1. Structures de données
2. Méthodologie
3. Tri
4. Recherche de motifs

2 x 1 heure par semaine pendant 6 semaines

Les TD et TP en demi-groupe

TP : programmation en langage C (salle T)
TD : exercices d'algorithmique (salle U ou V)

début : TP le 13 octobre.

deux groupes de taille équilibrée (11 stations en salle T)

Séquence d'instructions

L'exécution d'un programme dans un ordinateur est
l'exécution d'une séquence d'instructions,
agissant sur la mémoire de la machine,
interagissant avec l'extérieur par des entrées/sorties.

La répétition d'une sous-séquence d'instruction se fait par une boucle,
Le choix entre plusieurs sous-séquences possibles se fait par un aiguillage.

Les briques de la programmation impérative sont
les séquences d'instructions, les boucles et les aiguillages.

Variables et types

L'action du programme sur la mémoire se fait par l'intermédiaire de variables, chacune ayant un type qui décrit ce qu'elle peut contenir.

Une variable correspond à un espace mémoire d'une taille adaptée au type.

Un type **pointeur** référence un espace mémoire, valide ou non, correspondant à une variable ou non.

Types élaborés

Tableaux : une variable qui contient plusieurs copies numérotées et du même type.

Structures/Enregistrements : une variable qui contient plusieurs champs successifs de types variés.

Unions : une variable qui contient plusieurs champs superposés.

Objets : la structure contient aussi les méthodes pour la manipuler.

Procédures, arguments

Une **procédure** (routine) est une séquence d'instructions ayant des paramètres, appelés arguments.

Ces **arguments** peuvent être transmis de plusieurs façons :
par nom, par valeur, par référence.

Une procédure peut manipuler des variables locales,
invisibles de l'extérieur.

Une **fonction** est une procédure ayant une valeur résultat.

Gestion de la mémoire

De la mémoire est réservée pour
- une déclaration de variable
- un passage d'argument par valeur
- une allocation mémoire manuelle

La mémoire est libérée
- en fin de zone de visibilité d'une variable
- par une libération manuelle
- par une libération automatique des zones qui ne serviront plus (GC : garbage collecting, glanage de cellules)

Langage compilé, interprété

Un interpréteur est un programme qui lit et exécute
au fur et à mesure les instructions du langage (ex. : maple).

Un compilateur traduit le fichier des instructions de haut niveau
en des instructions du microprocesseur.
Il faut exécuter le programme résultat de la compilation (ex. : langage C).

Types, opérations

entiers signés : `short`, `int`, `long`

entiers non signés : `unsigned ...`

flottants : `float`, `double`

caractères : `char`

vide : `void`

constantes : `-3`, `2.7`, `'c'`

déclaration de variable : `int i`

résultat de fonction : `return(`*expr*`)`

séquence : { *inst* ; *inst* ; }

affectation : `i = 3`

comparaison : `i == 3`, `<`, `<=`, `!=`

arithmétique : `+`, `-`, `*`, `/`, `%`

in(dé)crémentation : `i++`, `i--`

Contrôle

tests

`if` (*expr*) *inst* ;

`if` (*expr*) *inst* ; `else` *inst* ;

expr `?` *inst* `:` *inst*

switch (*expr*) {
  case *value* : *expr* ; break ;
  default : *expr* ;
}

boucles

`while` (*expr*) *inst* ;

`for` (*start* ; *end* ; *cont*) *inst* ;

exemple : `for (i=0 ; i<10 ; i++) ... ;`

Tableaux et pointeurs

`int *U` définit un pointeur vers un/des entiers.
`U = (int*) malloc(sizeof(int))` alloue un espace mémoire pour un entier.
On accède à cet entier par `*U`. On libère la mémoire par `free(U)`.

`int T[10]` définit un tableau de 10 entiers, numérotés de 0 à 9.
On accède à l'élément 4 par `T[4]`.
`int m[10][10]` définit un tableau à deux coordonnées.
Il y a définition implicite de pointeurs.

Fonctions, macros

fonction

```
int min(int a, int b)
{
    if (a < b) return (a);
    else return (b);
}
```

macro

```
#define min(a,b) ((a)<(b)?(a):(b))
```

Remplacement syntaxique

Préprocesseur

Headers dans `toto.h` :
prototypes

```
#ifndef H_TOTO
#define H_TOTO
int min(int a, int b);
typedef struct chaine {
    char s[80];
    struct chaine * next;
} chaine;
#endif
```

inclusion

```
#include <stdlib.h>
#include <stdio.h>
#include "toto.h"
```

commentaire

```
/* ceci est un commentaire
   sur deux lignes */
```

Allocation mémoire et portée

- **auto** réserve de la place, la durée de vie est celle du bloc (allocation dans la pile) ou sinon le programme
- **static** réserve de la place, la durée de vie est le programme,
- **extern** ne réserve pas de place mais référence un élément extérieur
- **register** identique à auto, c'est un conseil au compilateur pour garder la variable dans un registre.

la **portée** est le bloc (la fonction) ou sinon le fichier

Utiliser gcc

`toto.c` et `toto.h` $\longrightarrow$ `toto.i`

Préprocesseur : expansion des macros (`cpp` ou `gcc -E`)

`toto.i` $\longrightarrow$ `toto.s`

Compilateur : création du fichier assembleur (`gcc -S`)

`toto.s` $\longrightarrow$ `toto.o`

Assemblage : création du fichier objet (`as` ou `gcc -c`)

`toto.o` $\longrightarrow$ `toto`

Éditeur de liens : création de l'exécutable (`ld` ou `gcc`)

Debug : `-g` Optimisation : `-O` Messages : `-Wall`

Tout en une instruction : `gcc -Wall -g -O toto.c -o toto`