Système d'exploitation

Jean-Sébastien Coron

Université du Luxembourg

October 25, 2009

Le langage C

- Langage de programmation le plus populaire.
- Programmation structurée.
  - Programmation par procédures.
- Typage de données.
  - Types proches de la machine (int, float...).

Programme "Hello world"

```
#include <stdio.h>
int main()
{
  printf("Hello world \n");
}
```

- #include: inclusion d'autres modules ou librairies.
  - stdio.h: librairie d'entrées/sorties.
- int main(): définition de la fonction principale.
- printf(): instruction d'affichage.
  - \n: retour à la ligne.

Utilisation

- Saisir le programme.
  - `xemacs hello.c &`.
- Le compiler.
  - `gcc hello.c -o hello.exe`
- L'exécuter.
  - `./hello.exe`

La fonction main

```
#include <stdio.h>
int main()
{
  printf("Hello world \n");
}
```

- Fonction main:
  - C’est la fonction appelée lors du lancement du programme.
  - Aucune autre n’est exécutée automatiquement.
- Indispensable.

Variables

- Le programme peut stocker des variables en mémoire.

```
int a;
a=2;
printf("a=%d\n",a);
```

- Déclaration d'un entier: `int a;`
- Assignation: `a=2;`
- Affichage:
  - `printf` avec `%d` pour un entier.
  - Résultat: `a=2`

Typage

- Il faut définir le type des variables.
- Variables entières.
  - `short`: 16 bits ± 32767.
  - `int`: 16 ou 32 bits $\pm 2.10^9$.
  - `long`: 32 bits $\pm 9.10^{18}$.
- `unsigned short`, `unsigned int`, `unsigned long` $\rightarrow$ entiers positifs.

Les flottants

- Codage:
  - Mantisse: m.
  - Exposant: e.
  - $m * 2^e$.
- `float`: 24+8 bits.
  - Précision 2^{-23}, $< 10^{38}$.
- `double`: 53+11 bits.
  - Précision 2^{-53}, $< 10^{308}$.
- `long double`: 64+16 bits.
  - Précision 2^{-64}, $< 10^{4932}$.

Déclaration des variables

- Variables globales : au début du programme, valables dans tout le programme.
- Variables locales : à l'intérieur d'une fonction, valables à l'intérieur de la fonction.

```
#include <stdio.h>
int u;
int main()
{
  int a;
  a=2;
  u=3;
  printf("a=%d,u=%d\n",a,u);
}
```

Initialisation des variables

- Lors de la déclaration d'une variable, son contenu est aléatoire.
- On peut l'initialiser simultanément.

```
#include <stdio.h>

int u=3;
int main()
{
  int a=2;
  printf("a=%d,u=%d\n",a,u);
}
```

Opérateurs sur les nombres

- On peut réaliser les opérations suivantes sur les entiers et les flottants.
  - $a + b$: addition.
  - $a - b$: soustraction.
  - $a * b$: multiplication.
  - a/b: division.
    - Division euclidienne pour les entiers.
  - $a\%b$: reste modulaire.

Affichage des variables

- L'instruction `printf` affiche sur la sortie standard (l'écran) du texte et le contenu des variables.
  - %d pour un `int` ou `long`.
  - %f pour un `float` ou `double`.

```
float a=2.3;
int b=4;
printf("a=%f,b=%d\n",a,b);
```

Exécution conditionelle

- Un test permet d'exécuter une ou plusieurs instructions en fonction de l'état de certaines variables.

```
if (i>5)
  printf("La variable i est supérieure à 5");
```

- Utilisation de `else`.

```
if (i>=0)
  printf("La variable i est positive.");
  else printf("La variable i est négative");
```

Exécution conditionelle

- Exécution de plusieurs instructions:

```
if (x>0)
{
  y=2*x;
  z=3*x;
}
else
{
  y=x-1;
  z=x-2;
}
```

- L'instruction `else {...}` est optionelle.

Opérateurs de test

- Tests possibles:
  - Egalité: a==b
  - Différence: a!=b
  - Comparaison stricte: a<b
  - Comparaison large: a<=b
- Résultat d'un test.
  - Le résultat du test est un entier nul si le test est faux, non nul si le test est vrai.

Opérations sur les tests

- Négation:
  - `!(test)`.
- Conjonction (ET)
  - `((test1) && (test2))`
- Disjonction (OU)
  - `((test1) || (test2))`

Exemple

```
int maximum(int a,int b)
{
  int m;
  if (a>b) m=a;
  else m=b;
  return m;
}
```

Exemple

- On souhaite vérifier que deux quantités sont positives, et afficher un message si ce n'est pas le cas.

```
float x,y;
...
...
if((x<0) || (y<0))
{
  printf("Erreur: la variable x<0 ou y<0");
}
```

Exemple

- On souhaite déterminer si une variable x appartient à un certain intervalle $[a, b]$.

```
int intervalle(float x,float a,float b)
{
   if ((x>=a) && (x<=b)) return 0;
   return 1;
}
```

Boucles conditionnelles

- Il est possible de répéter plusieurs fois le même bloc d'instruction, en fonction du résultat d'un test.
- Instruction `while (test) instruction`
  - L'instruction s'exécute tant que le résultat du test est vrai.
  - L'instruction n'est pas exécutée si le résultat du test est faux la première fois.
- Instruction `do (instruction) while (test)`
  - L'instruction est exécutée une fois, puis répétée si le résultat du test est vrai.
  - L'instruction est exécutée au moins une fois.

Instruction `while`

- Exemple: affiche les entiers de 1 à 10.

```
#include <stdio.h>
int main()
{
  int i=1;
  while(i<=10)
  {
    printf("%d\n",i);
    i=i+1;
  }
}
```

Instruction do while

- Exemple: on demande à l'utilisateur d'entrer un entier qui doit être positif, sinon on repose la question.

```
#include <stdio.h>
int main()
{
  int a;
  do
  {
    printf("Entrez un nombre positif.\n");
    scanf("%d",&a);
  } while (a<0);
}
```

Boucle for

- Une boucle `for` permet de répéter une instruction plusieurs fois, à l'aide d'une variable de contrôle.
  - `for(init;test;itération)  opération;`
  - `init`: initialiser la variable de contrôle.
  - `test`: test de la variable de contrôle
  - `itération`: opération sur la variable de contrôle
  - `opération`: opération de la boucle
- Exemple: afficher les entiers de 1 à 10.
  - ```
    for (i=1;i<=10;i=i+1)
      printf("%d\n",i);
    ```
    ```

Boucle for

- Syntaxe :
  - `for(init;test;itération)  opération;`
- Séquence d'opération :
  - ```
    init
    test:si test=faux, saut à fin du for
      opération
      itération
      retour à test
    fin du for
    ```
- Exemple: somme des entiers de 1 à 10 :
  - ```
    s=0;
    for (i=1;i<=10;i=i+1)  s=s+i;
    ```

Variable de contrôle

- On peut aussi décrémenter la variable :
  - Boucle affichant les nombres de 10 à 0 :

```
for (i=10;i>=0;i=i-1)
   printf("%d\n",i);
```

- Explication :
  - `i=10`: valeur initiale de la variable `i`
  - `i>=0`: la boucle continue tant que `i>=0`
  - `i=i-1`: après chaque exécution de l'opération de la boucle, la variable `i` est décrémentée.

Boucle For

- Exemple :
  - `for (i=0;i<6;i=i+1)`
    `printf("%d\n",i);`
  - La boucle affiche `0,1,2,3,4,5`.
  - Elle commence à `i=0`, vérifie que `i<6`, et affiche `0` à l'écran.
  - La boucle continue avec `i=1,2,3,4,5`.
  - Lorsque `i=6`, la condition `i<6` n'est plus réalisée, et on sort de la boucle.
    - La valeur 6 n'est pas affichée.
    - La boucle s'est exécutée en tout 6 fois, pour `i=0,1,2,3,4,5`

Exemple

- On veut calculer $a \cdot b$ en calculant

$$a \cdot b = b + \ldots + b$$

- Avec une boucle :

```
int mult(int a,int b)
{
  int i,c=0;
  for(i=0;i<a;i++)
  {
    c=c+b;
  }
  return c;
}
```

Attention

- Toujours vérifier que la boucle se termine :
  - Incorrect : `for (i=0;i<6;i=i-1)`
- Si nécessaire, utiliser `printf` dans la boucle pour vérifier la valeur du compteur.
- Bien compter le nombre d'exécutions de la boucle :
  - `for(i=0;i<10;i++)` exécute 10 fois la boucle (i de 0 à 9)
  - `for(i=0;i<=10;i++)` exécute 11 fois la boucle (i de 0 à 10)
  - `for(i=1;i<10;i++)` exécute 9 fois la boucle (i de 1 à 9)

Utilisation de `for`

- On utilise généralement une boucle `for` lorsqu'on souhaite exécuter une opération un nombre de fois connu à l'avance.
- Exemple: l'instruction va s'exécuter 10 fois.

```
int n=10;
int i;
for(i=0;i<n;i++)
{
  <instruction(i)>
}
```

for = while

- Les instructions `for` et `while` sont en faites équivalentes.

```
for (init;test;itération)
  opération
```

```
init
while (test)
{
  opération
  itération
}
```