C programming for embedded systems

Nasrine DAMOUCHE
nasrine.damouche@isae-supaero.fr

ISAE-SUPAERO
Département DISC
Équipe IpSC

Introduction

- **Algorithme** : l'ensemble des instructions séquentielles et structurées pour résoudre un problème
- **Programme** : s'adresser à une machine pour exécuter l'algorithme
- **Programmer** :
  - Éditer le texte
  - Compiler
  - Exécuter
  - Tester, debugger

Language C

- Langage impératif
- Crée en 1972 par Dennis Richie et Ken Thompson
- Conçu pour la programmtion système d'UNIX (développer une version portable d'UNIX)
- Inspiré beaucoup de langages (C++, Java, PHP, ...)

Exemple

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

Structures de base du C

- Le langage C est un langage typé
  - Toute variable, constante, fonction est typée
- Type
  - La façon dont l'élément est representé en mémoire
  - La taille occupée par l'élément
- Types élémentaires
  - int, float, double, short, long, char, signed, unsigned

Variables

- **Définition** : Possède un type, un nom, une location mémoire et une valeur
- **Déclaration** : type v ;
- Toute variable doit être définie avant d'être utilisée
- Une définition peut apparaître n'importe où dans un programme
- Une variable est définie jusqu'à la fin de la première instruction composée (marquée par }) qui contient sa définition
  - **Exemple**
    int a ; double x ; char c ;
- Une variable peut être initialisée lors de sa déclaration
  - **Exemple**
    int a = 12 ; double x = 12.33 ; char c = 'H' ;

Variables

Variable globale

- Définie hors de toute fonction
- Vue par toutes les fonctions du programme source
- Persistence pendant toute la durée de vie du programme

Variable locale

- Déclaration à l'intérieur d'une fonction
- Détruite en sortie de fonction

Exemple

```
#include <stdio.h>
void addition();
int a = 10,
int b = 20;
int main () {
   int c;
   c = a + b;
   printf ("Valeur de a = %d, b = %d and c = %d dans le main \n", a, b, c);
   return 0;
}
void addition(){
   int c;
   c = a + b;
   printf ("Valeur de a = %d, b = %d et c = %d dans la fonction addition \n", a, b, c);
}
```

Constantes

- **Déclaration** : const type nom = val ;
- Il ne sera pas possible de modifier la valeur de la constante dans le reste du programme (erreur à la compilation)

Exemple

const int largeur = 30 ;

Expressions

- **Affectation**
  Déclaration var = expr
  affecte la valeur de *expr* à la variable *var* et retourne la valeur affecté à *var* comme résultat

  Exemple
  i = (j = 2) ; /* affecte la valeur 2 à j puis à i et retourne 2 */

- Opérations classiques
  - **Opérateurs arithmétiques**
    $+, -, *, \div, \%$
  - **Opérateurs de comparaison**
    $<, <=, >, >=, ==, !=$
  - **Opérateurs booléens**
    && (l'opérateur "ET"), || (l'opérateur "OU"), ! ("NON")

Pré et post incrément

- ++ i
  incrémente la variable i et retourne la nouvelle valeur (i = i + 1)
- i ++
  incrémente la variable i et retourne l'ancienne valeur (i = (i + 1) - 1)
- - - i
  décrémente la variable i et retourne la nouvelle valeur (i = i - 1)
- i - -
  décrémente la variable i et retourne l'ancienne valeur (i = (i - 1) + 1)

Exemple 1

```
int i, x;
i = 2;
x = ++i;
```

Exemple 2

```
int i, x;
i = 2;
x = i++;
```

Instructions de contrôle

- **Condition**

  if (expr) { instr ; }
  if (expr) { instr1 ;} else { instr2 ; }

  Exemple

```
if ( x < 5 ) {
   i = i + 1 ;
}
if ((x==1) && (i > 10)) {
   i = i + 1 ;
   x = x * 2 ;
}
else {
   x = i ;
}
```

- **Boucle For**

  for (expr1; expr2; expr3) { instr ; }

  Exemple

```
for(i =0; i< 10; i++) { instr; }
/* Une autre façon d'écrire la boucle for (conditionnel ternaire) */
(condition) ? instr1 : instr2 ;
```

Instructions de contrôle

- **Boucle While**

```
while (expr) {
   instr ;
}
```

Exemple

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

- **Boucle Do**

```
do {instr}
while (expr) ;
```

Exemple

```
int n;
do {
   printf("Introduisez un nombre entre 1 et 10 :");
   scanf("%d", &n);
}
while (n<1 || n>10);
```

Instructions de branchement non conditionnel

- **break** : le programme quitte immédiatement la boucle et reprend l'exécution avec l'instruction suivant la boucle
- Exemple

```
main() {
int i;
for (i = 0; i < 5; i++) {
   printf("i = %d\n",i);
   if (i == 3){
      break;
   }
   }
printf("valeur de i a la sortie de la boucle = "%"d\n",i);
}
```

imprime à l'écran
i = 0 i = 1 i = 2 i = 3 valeur de i a la sortie de la boucle = 3

Continue

Instructions de branchement non conditionnel

- **Continue** : ignore l'itération actuelle de la boucle et continue à l'itération suivante

```
main() {
   int i;
   for (i = 0; i < 5; i++){
      if (i == 3)
      continue;
      printf("i = %d\n",i);
   }
printf("valeur de i a la sortie de la boucle = %d\n", i);
}
```

imprime

i = 0 i = 1 i = 2 i = 4 valeur de i a la sortie de la boucle = 5

go to

- **Définition** : permet d'effectuer un saut inconditionnel pour aller n'importe où à l'intérieur d'une fonction
- Très utilisé après des détections d'erreur, car il permet de sortir de plusieurs blocs imbriqués
- **Exemple**

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main() {
   int score = 1;

   START:
      if (score > 1000)
         goto EXIT;
      else {
         score += 1;
         goto START;
      }

   EXIT:
      printf("score: %d\n", score);
      exit(EXIT_SUCCESS);
}
```

Return

- **Définition** : provoque la terminaison de l'exécution de la fonction dans laquelle elle se trouve et le retour à la fonction appellante
- Appeler de manière implicite à la fin d'une fonction
- **Utilisation** : **return** ;
  return expression ;
- **Exemple**

```
int lecture() {
   int c ;
   switch (c = getchar ()) {
      case EOF : return -1;
      case 'A' : return 0 ;
      case 'a' : return 1 ;
      case '0' : return 2 ;
      default : break ;
   }
return -2;
}
```

Entrèe : Lire des données depuis le clavier
Sortie : Afficher des données sur l'écran

Exemple

```
#include<stdio.h>
int main(){
   int x ;
   int y ;
   printf("entrer le premier nombre:") ; // (Affichage à l'écran)
   scanf("%d",&x) ; // (Lecture au clavier)
   printf("entrer le deuxième nombre:") ;
   scanf("%d",&y) ;
   printf("La somme de x et y est : %d", x + y) ;
   return 0 ;
}
```

Formats d'impression pour printf

format	conversion en	écriture
%d	int	décimale signée
%ld	long int	décimale signée
%u	unsigned int	décimale non signée
%lu	unsigned long int	décimale non signée
%o	unsigned int	octale non signée
%lo	unsigned long int	octale non signée
%x	unsigned int	hexadécimale non signée
%lx	unsigned long int	hexadécimale non signée
%f	double	décimale virgule fixe
%lf	long double	décimale virgule fixe
%e	double	décimale notation exponentielle
%le	long double	décimale notation exponentielle
%g	double	décimale, représentation la plus courte parmi %f et %e
%lg	long double	décimale, représentation la plus courte parmi %lf et %le
%c	unsigned char	caractère
%s	char*	chaîne de caractères

Evaluation des expressions

Priorité des opérateurs

- f(x), tab[y], x++, x- -
- !x, -x, ++x, - -x
- x × y, x ÷ y, x % y
- x + y, x - y
- x » y, x « y // Décalage binaire à gauche / à droite
- x < y, x > y, x >= y, x <= y
- x == y, x != y //comparaison
- x && y
- x || y
- x = y, x op= y // affectation
- x ? y : z // opérateur conditionnel ternaire

Evaluation des expressions

Exemple 1

```
int i=3, j = 2, m;
double r = 3.4;
m = (i / j) * r ;
```

Evaluation des expressions

Exemple 1

```
int i=3, j = 2, m;
double r = 3.4;
m = (i / j) * r ;
```

- évaluer d'abord (i/j). Cela donne 1 (division entière)
- évaluer le produit 1 * r (d'abord convertir 1 en double (1.0), et faire le produit). Cela donne 3.4.
- Pour l'affectation, comme m est entier, 3.4 est converti en int. Cela donne $m = 3$.

Exemple 2

```
int i=3, j = 2, m;
double r = 3.4;
m = ((double)i / j) * r ;
```

Evaluation des expressions

Exemple 1

```
int i=3, j = 2, m;
double r = 3.4;
m = (i / j) * r ;
```

- évaluer d'abord (i/j). Cela donne 1 (division entière)
- évaluer le produit 1 * r (d'abord convertir 1 en double (1.0), et faire le produit). Cela donne 3.4.
- Pour l'affectation, comme m est entier, 3.4 est converti en int. Cela donne $m = 3$.

Exemple 2

```
int i=3, j = 2, m;
double r = 3.4;
m = ((double)i / j) * r ;
```

m = 5

Fonction et Procédure

- **Fonction**
  - **Définition** : Tâche répétitive englobée dans une partie de code délimitée et nomée et qui renvoie un résultat.
  - **Déclaration** : type nom (liste des paramètres) { corps }
    - **type** : est le type du résultat de la fonction
    - **liste des paramètres** : paramètres formels (type1 param1, type2 param2, ...)
    - **corps** : les instructions à efffectuer
    - le corps utilise ses propres variables locales, les éventuelles variables globales et les paramètres formels
    - Une **procédure** est une fonction mais qui ne renvoie aucun résultat (**void**)

Appel de Fonction

Utilisation : nom(liste des arguments) ;

- Une fois la fonction est définie, on peut l'utiliser
- La liste des arguments (paramètres réels) est $expr1, expr2, etc$ où chaque $expr$ est compatible avec le type $type_i$ du paramètre formel $param_i$

Exemple

```
int max(int a, int b) {
   int res = b ;
   if (a > b) {
   res = a ;
}
   else {
   res = b ;
}
   return res ;
}
int main(){
   int x = 34, y = 6;
   int z = max(x, 2*y+12) ;
   printf("%d", z);
}
```

Fonction récursive

- **Définition** : est une fonction qui fait appel à elle-même
- **Exemple**
  1! = 1
  n! = n * (n-1)! (n > 1)

```
int fact(int n) {
   if (n == 1) {
      return 1;
   }
   else {
      return n * fact(n-1);
   }
printf("%d i", fact(i)) ;
}
```

Fonction itérative

- **déclaration** : consiste à répéter n fois un processus en faisant changer la valeur des variables jusqu'à obtention du résultat
- Se programmer par une boucle (for, while, etc.)
- **Exemple**

```
int fact_iter(int n) {
   int i = 0;
   int result = 1 ;
   while(i < n ) {
      i = i + 1 ;
      result = result * i ;
   }
   return(result) ;
}
```

Types composés

- Tableaux
- Structures
- Énumération

Tableaux

- Un tableau est un type de donnée
- **Déclaration** : type nom_tab[Nbr_éléments] ;
  ou type nom_tab[] = {1, 2, 3} ;
- **Utilisation** : nom_tab[i]

Exemple

```
main(){
   int tab1[10] ;
   int i ;
   for(i=0; i<10; i++){
      printf("tab1[%d] = %d \ n", i, tab1[i]);
   }
}
```

Tableau à deux dimensions

- **Déclaration** : type nom_tab[Nbr_lignes][Nbr_colonnes] ;
- **Utilisation** : nom_tab[i][j]

Exemple

```
#define n 3 ;
#define m 4 ;
int tab2[n][m] ={{1,2,3},{4,5,6,7}} ;
main(){
   int i, j ;
   for(i = 0; j < m; i++){
      for(j = 0; j < n; j++){
         printf("tab2[%d][%d] = %d \n", i, j, tab2[i][j]) ;
      }
   }
}
```

Structure

- **Définition** : Une structure définit un nouveau type de données. Elle possède un nom et est composée de champs (sous-types nommés).
- **Déclaration** : struct nom_variable { type_1 champs_1; type_2 champs_2; ... };
- **Utilisation** : nom_variable.nom_champ_i

Exemple

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

Structure (suite)

Exemple

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

struct ligne {
   struct point p1 ;
   struct point p2 ;
};

void foo() {
   struct point p ;
   struct ligne l ;
   p.x = 1 ; p.y = 2 ;
   l.p1.x = 3 ;
   l.p1.y = 5 ;
   l.p2 = p ;
   printf("[%d %d] \n", l.p2.x, l.p2.y);
}
```

Énumération

- **Définition** : Définir une liste complète des valeurs (entières) qui peuvent être attribuées à une variable appartenant à ce type énuméré
- **Déclaration** d'un type énuméré : **enum** nom_de_énumération { enumerateur1 , enumerateur2 , ..., enumerateurN } ;
- **Exemple**

```
enum couleurs {
   rouge ,
   vert ,
   bleu
} ;
```

- **Déclaration** d'une vaiable de type énuméré : enum couleurs rvb ;
- **Utilisation** :

```
rvb = rouge ;
if (rvb == rouge) printf("Red");
```

Chaînes de caractères

- **Définition** : une chaîne de caractères est un tableau de caractères dont la fin est indiquée par un caractère particulier '\0'
- **Déclaration** : char chaine[TAILLE];
- **Initialisation** : char chaine[11] = "bonjour" ;
  char chaine[11] = 'b', 'o', 'n', 'j', 'o', 'u', 'r', '\0' ;
- **Modification** du contenu : chaine[3] = 's';
  chaine[5] = 'i';
- **Manipulation** : La bibliothèque *string.h* contient un ensemble de fonctions permettant de manipuler des chaînes de caractères :
  - La fonction strlen() : renvoie la longueur d'une chaîne de caractères ;
  - La fonction strcpy() : permet de copier une chaîne de caractères dans une autre ;
  - La fonction strstr() : permet de chercher la première occurrence d'un caractère dans une chaîne

Notions de pointeurs

- La mémoire physique est vue comme une suite finie d'octets
- Un pointeur est une variable contenant l'adresse d'une variable
- Une valeur de type pointeur est une adresse mémoire
- un pointeur est donc un espace mémoire pouvant contenir une adresse

Pointeurs (suite)

- **Déclaration** : type_pointé *nom_pointeur ;
- type_pointé * : type quelconque
- **Exemple** : int i = 20; int *p; p = &i ;
- L'opérateur **&** : retourne l'adresse mémoire d'un objet
- L'opérateur ***** : opérateur de déréférencement (opérateur d'indirection) qui retourne la valeur de l'objet pointé
- **Exemple**
  int i, j, *p =& i;
  *p = 4;
  j = *p + 1 ;

Arithmétique des pointeurs

- La valeur d'un pointeur est un entier (entier long)
- L'addition d'un entier à un pointeur
  - Le résultat est un pointeur de même type que le pointeur de départ
- La soustraction d'un entier à un pointeur
  - Le résultat est un pointeur de même type que le pointeur de départ
- La différence de deux pointeurs pointant tous deux vers des objets de même type
  - Le résultat est un entier

- **Exemple**

```
main(){
   int i = 3 ;
   int *p1, *p2 ;
   p1 = &i ;
   p2 = p1 + 1 ;
   printf("p1=%ld \t p2 = %ld \n", p1, p2) ;
}
```

Les directives au préprocesseur

- Préprocesseur est une programme exécuté lors de la première phase de la compilation
- Effectuer des modifications textuelles sur le fichier source à partir des directives
- Introduites par le caractères #
  - Incorporation de fichiers source (#include)
  - Définition de constantes symboliques et de macros (#define)
  - Compilation conditionnelle ($\#if$, $\#ifdef$, etc.)

Mots résevés & Commentaires

Reserved identifiers

auto const struct unsigned break continue else for long signed switch void case default enum goto register sizeof typedef volatile char do extern if return static union while

Commentaires

```
/* This is a comments */
// This is another one
/* This is a comment
   on two lines */
```

Institut Supérieur de l'Aéronautique et de l'Espace
10 avenue Édouard Belin – BP 54032
31055 Toulouse Cedex 4 – France
Phone: +33 5 61 33 80 80
www.isae-supaero.fr