11/2023

Structure d'un programme RAPE

Dans ce cours nous présentons la structure retenue pour les programmes RAPE qui auront toujours la même forme avec des parties plus ou moins longue mais toujours dans cet ordre.

Table des matières

Le programme et ses différentes parties :

- COMMENTAIRES
- ENTÊTE
- DIRECTIVES
- DECLARATION DES VARIABLES
- DECLARATION DES FONCTIONS (VOID)
- FONCTION PRINCIPALE (MAIN°

  BOUCLE

  FIN BOUCLE
- FIN FONCTION PRINCIPALE
- ICI CE TERMINE LE PROGRAMME

```
/**
 *
 * COMMENTAIRES
 *
*/
```

---------- ENTÊTE pour appeler les fonctions préprogrammées----------

```
#include "mcc_generated_files/mcc.h"
```

……..

#include <fichier_en_tête.h> ou #include "chemin_du_fichier_en_tête"

Si le nom du fichier en tête est mis entre '<' et '>', alors le préprocesseur commence par rechercher le fichier en tête dans le répertoire prédéfinit par le compilateur comme étant celui les contenant tous. Si en revanche le fichier en tête est mis entre guillemets, le préprocesseur le recherche grâce au chemin indiqué. C'est le cas de vos propres librairies.

---------- DIRECTIVES----------

Cette directive remplace dans le programme toutes les occurrences du symbole par la suite de caractères. Cette suite peut représenter un nombre, un texte, etc….

DIRECTIVES SUR LES ENTREES SORTIES

```
#define det_cent_ANA PORTAbits.RA0 // ;RA0 P2 ANA entrée sur détecteur optique central
```

……

//PORT C

```
#define sortie_mot_enable_B_PW4 LATCbits.LATC0 // patte 15 commande du pwm
```

……

//PORT D

```
#define sortie_mot_ENA_PW3 LATDbits.LATD0 // // patte 19 commande du pwm dr
```

……

//**********DECLARATION DES VARIABLES**********

unsigned int capteur_central_avant_AN0, capteur_droit_avant_AN1, capteur_gauche_avant_AN2, tension_batterie_AN3;

Toute variable d'un fichier doit être déclarée AVANT son utilisation dans une expression quelconque. Chaque donnée utilisée par un programme C doit posséder un type prédéfini. Ce type détermine l'ensemble des valeurs possibles que peut prendre la donnée pendant l'execution du programme. C'est ainsi qu'une donnée pourra contenir des valeurs de types :

///**********DECLARATION DES FONCTIONS VOID**********

Dans un programme, on peut avoir besoin d'exécuter le même ensemble d'instructions plusieurs fois, mais avec des données différentes. On regroupe alors cet ensemble d'instructions en une fonction.

Le prototype d'une fonction est une ligne décrivant, au reste du programme, ses caractéristiques principales.

//void sortie (commentaire)

void commande_sorties(void);

//void cycle

void cycle (void);

void FM_tempo_100ms(void);

……

//**********

/* commentaire

Main application

*/ fin commentaire

**********FONCTION MAIN **********

void main(void) *corps principal ne peut pas fonctionner sans, de là toutes les fonctions sont appelées*

//**********

{ *la parenthèse définit le début de la fonction « main » une autre parenthèse dans l'autre sens terminera la fonction « main »*

// Initialize the device *intialisation du pic*

SYSTEM_Initialize();

Dans cette partie les fonctions sont appelées une fois à la mise sous tension

****

////////// PROGRAMME APPELE EN BOUCLE //////////

while (1)

« tant que » on est à 1, on y est toujours !!! donc on reboucle. C'est une boucle infinie c'est-à-dire qu'à chaque fois que la deuxième parenthèse de fermeture de la fonction est atteinte on reboucle sur cette position.

```
{


  capteur_central_avant_AN0 = ADCC_GetSingleConversion(channel_ANA0);
```

capteur_central_avant_AN0 : texte choisi par nos soins ADCC_GetSingleConversion(channel_ANA0) traité dans le MCC Generated Files puis dans la fonction adcc.c.....

```
//---------------------------------------------------------------------
//**********************cde moteur************************************
FM_tempo_100ms();
commande_sorties();
```

la fonction a été déclarée en début de programme dans VOID mais ensuite pour être active elle doit être appelée soit sur chaque tour de cycle comme c'est le cas actuellement ou dans le programme quand on en a besoin.

……

```
    if (fm_bit_500ms )  {
            printf("\r\n");
             printf("    cmpt_roue_GA:  %d " , cmpt_roue_GA);//
                        }
```

Explication : à chaque fois que l'on aura un front montant du bit 500ms on passera dans cette partie encadrée par les deux parenthèses dans ce cas on fait une impression sur le port série du compteur de la valeur de la roue gauche et le \r\n permet de revenir à la ligne suivant en début de ligne.

```
}//fin while(1))    FIN BOUCLE
}//fin main
/**
 End of File
*/


//=============================================================================
//                                  sous programmes
//=============================================================================
```

Ici il y a toutes les fonctions appelées par le programme principal (main).

```
void cycle ()
```

```
{
  if (fm_bit_100ms)  {sortie_led_coeur_ve = !sortie_led_coeur_ve;}
}
```

Dans cette fonction toutes les 100s il y a un changement d'état de la led verte qui est sur la carte, le nom est choisi par nos soins.

```
void commande_sorties(void)
   {
     sortie_mot_brosse = 0;
……..……
```

Commande des sorties à un seul endroit impératif pour avoir des sorties qui sont fixes.

```
     }
```

Fonction de front montant

```
void FM_tempo_100ms(void)
  {
  if (bit_100ms & !inter_100ms ) {fm_bit_100ms = 1;} //
  else {fm_bit_100ms = 0;}
  /* bit intermédiaire */
  inter_100ms = bit_100ms;
  return;
  }
```

Petite aide à la compréhension sur la fonction

if(condition) { instruction1 ; } else { instruction2 ; }

traduction : si j'ai la condition qui est validée alors j'exécute la l'instruction 1 sinon (la condition n'est pas valable j'exécute l'instruction2.

<u>Dans notre exemple ci-dessus</u>

si le bit_100ms et pas le bit inter_100ms alors on a le fm_bit_100ms qui est mis à 1 dans le cas contraire on a le bit qui est mis à zéro.

ICI SE TERMINE LE PROGRAMME