What does this document covers?

This document attempts to explain the basic structure of a C18 program. It is followed by some simple examples.

A very simple C18 program is shown below:

Example 1

```
void main (void)
{
        char x;
        int num;

        x = 10;
        num = 12345;
}
```

What does this program do?

This program, although not very useful for a practical application, describes the basic structure of a C18 program.

- The program consists of four 'C' statements
- The first two statements declare two variables that will be used to hold numerical values (data)
- The next two statements assign values to these two variables.
- The program then terminates

Discussion:

void main (void)

main

- It is the main program (that is why the name 'main') that is always executed.
- It is correctly known as a function, which essentially is same as subroutine in an assembly language.
- It is always written (expressed) in lower case
- It is preceded by '**void**', again expressed in lower case
- It is followed by '(**void**)', again in lower case
- Every C18 program will have a main function
- If a program has multiple function, main program (function) is always executed first
- main (function) can call (execute) other parts (functions) of the overall program, if these are present.

{

- It is known as the starting brace (curly bracket)
- It forms the starting boundary of the main program
- It has to be matched by a closing brace

}

- It is known as the closing brace (curly bracket)
- It forms the end of the main program
- It forms a pair with the starting brace
- Between these two braces lies the function (main program, here), which is executed

Variable Declarations

```
char x;
int num;
```

char x;

- This is the first line of program, also known as a 'C' statement
- It has 3 parts:
  - **char**
    Indicates that we want to use a variable that will be able to hold an 8-bit value
  - **x**
    This is the name given to this variable
    It is the name of the place holder, where 8-bit value will be held
  - **;**
    Indicates the end of this program line ('C' statement).
    Every line of program has to be terminated by a semicolon (with some exceptions)

Int x;

- This is the second line of the program. It again is a 'C' statement
- It too has 3 parts:
  - **Int**
    Indicates that we want to use a variable that will be able to hold a 16-bit value
  - **num**
    This is the name given to this variable
  - **;**
    Indicates the end of the program line ('C' statement).
)

Variable Assignments

```
x = 10;
num = 12345;
```

x = 10 ;

- This is the third line of program (a 'C' statement)
- It has 4 parts:
  - **x**
    The name of the variable to which we wish to assign value
  - **=**
    This is assignment statement (same as in algebra)

- **10**
  This is value that is assigned to the variable
- **;**
  Indicates the end of this 'C' statement, as is required.

num = 12345 ;

- This is the 4th 'C' statement
- It too has 4 parts:
  - **num**
    The name of the variable to which we wish to assign value
  - **=**
    This is the assignment statement
  - **12345**
    This is value that is assigned to the variable
  - **;**
    Indicates the end of this 'C' statement, as is required.

Closing Remarks

- All variable declarations should be done before assigning any value to these.
- The following two programs are incorrect:

Incorrect Example 1b

In the following the values are assigned first and the name of variables is declared later.

```
void main (void)
{
        x = 10;
        num = 12345;

        char x;
        int num;
}
```

(Incorrect) Example 1c

In the following

- The first variable x is correctly declared first and then a value is assigned to it
- The second variable num is (incorrectly) assigned the value first are assigned first and then it is declared.

```
void main (void)
{
        char x;
        x = 10;

        num = 12345;
        int num;
}
```

(Incorrect) Example 1d

In the following:

- The first variable x is correctly declared first and then a value is assigned to it
- It may appear that the second variable num is correctly declared first and then a value assigned to it.
- Although, permitted in some 'C' compilers, it is not acceptable in 'C18'.
- All variables have to be first declared and then only values can be assigned to these.

```
void main (void)
{
        char x;
        x = 10;

        int num;
        num = 12345;
}
```

Order of Variable assignment

Assignment can be done in any order and does not have to follow the same order as the declaration of the variables. The following is correct:

(Correct) Example 1e

```
void main (void)
{
        char x;
        int num;

        num = 12345;
        x = 10;
}
```

Summary

- A C18 program (at the minimum) consists of a main program (function)
- The entire body of main function is enclosed by a pair of braces
- The variables are declared at the start
- Any number of variables can be declared
- Any **legal** variable name can be used
- After all variable have been declared, values can be assigned to required variables
- Every C statement is terminated by a semi-colon.

The following examples extend the program given in Example 1;

Example 2

```
void main (void)
{
        char x;
        char y;
        int num;

        x = 10;
        y = 5;
        num = 12345;
}
```

Discussion:

An additional variable y is declared and assigned a value later

Example 3

```
void main (void)
{
        char x;
        char y;
        int num;
        int sum;

        x = 10;
        y = 5;
        num = 12345;
        sum = num + 1000;
}
```

Discussion:

An additional variable sum is declared and is assigned a value in an algebraic expression

Identical variable types can be declared in a single statement

The following shows reworked examples 2 and 3

Example 2b

```
void main (void)
{
        char x, y;
        int num;

        x = 10;
        y = 5;
        num = 12345;
}
```

Discussion:

Both variables x and y are of type ‘char’ and can thus be declared in a single statement.

Example 3b

```
void main (void)
{
        char x, y;
        int num, sum;

        x = 10;
        y = 5;
        num = 12345;
        sum = num + 1000;
}
```

Discussion:

- Here x and y are declared in a single statement as both are of type ‘char’
- Similarly, both num and sum are of type ‘int’ and are declared in a single statement.

Only the identical variable types can be declared in a single statement

The following shows incorrect declaration for reworked example 2

(Incorrect) Example 2c

```
void main (void)
{
        char x, y, int num;

        x = 10;
        y = 5;
        num = 12345;
}
```

Discussion:

num is an 'int' type and cannot be declared in the same statement as char types

A variable name can only be used once in a function

(Incorrect) Example 4

```
void main (void)
{
        char x, y;
         int x;

        x = 10;
        y = 5;
}
```

Discussion:

Here x is first declared as a 'char' type variable and in the next statement it is used again and declared as an 'int' type. It is not permitted.

Notes:

A variable name can again be used in another function.

These variables with same names, but in different functions will behave in a manner similar to two different people with same name but in different households

pgm1.c - A program dealing with PIC18F hardware

Purpose:

The program is intended to turn on a LED connected to RC0 of PIC18F2520 microcontroller.

LED Operation:

RC0 pin High – LED On
RC0 pin High – LED On

Program:

```
// Name: SG
// pgm1.c

#include <p18F2520.h>
void main (void)
{
        TRISCbits.TRISC0 = 0;
        LATCbits.LATC0 = 1;
}
```

Discussion:

// Name: SG

- This line is a comment line
- It is used to provide general information, such as programmer's name, date, program name etc.
- It is ignored by C18 compiler
- A comment line starts by two front slashes '**//**'
- Any number of comment lines can be used
- A comment line can be used at the end of any C statement. (another form of comment can even be inserted in the middle of a C statement)

// pgm1.c

- This is also a comment line
- It is providing the name of the program file.
- A C18 program file will have '.c' extension

#include <p18F2520.h>

- This line indicates the name of file that is to be included along with the main program
- The file contains name and corresponding addresses in PIC18F2520 hardware
- The addresses of TRISC and LATC registers and their bits are contained in this file.
- This file is part of C18 compiler and is contained in ..MCC18/h folder.

void main (void)

- It is the main program (function) that is going to be executed.

{ }

- These two braces contain the main body of the program (main function)

TRISCbits.TRISC0 = 0;

- This is the first C18 statement
- It sets bit 0 in TRISC register to 0, setting the direction of RC0 as an output

LATCbits.LATC0 = 1;

- This is the second (and last) C18 statement
- It sets bit 0 in LATC register to 1, setting the pin voltage to Vcc (High)
- This turns on the LED
- After executing this statement the program terminates (when the program execution reaches the closing brace).

Pgm2.c – LED Flash

Purpose:

The program is intended to turn on a LED connected to RC0 of PIC18F2520 microcontroller and then turn it off.

LED Operation:

RC0 pin High – LED On
RC0 pin High – LED On

Program:

```
// Name: SG
// pgm2.c

#include <p18F2520.h>
void main (void)
{
        TRISCbits.TRISC0 = 0;     // set RC0 as an output
        LATCbits.LATC0 = 1;       // set RC0 High to turn on LED
        LATCbits.LATC0 = 0;       // set RC0 Low to turn off LED
}
```

Discussion:

The program operation is as follows:

- It first sets the direction of RC0 as output.
- Pin level of RC0 is set to high to turn on LED
- Pin level of RC0 is then set low to turn off LED

Pgm3.c – LED Flash with small delay inserted

Purpose:

The program is intended to turn on a LED connected to RC0 of PIC18F2520 microcontroller and then after a brief delay, the LED is turned off.

LED Operation:

RC0 pin High – LED On
RC0 pin High – LED On

Program:

```
// Name: SG
// pgm3.c

#include <p18F2520.h>
void main (void)
{
        TRISCbits.TRISC0 = 0;     // set RC0 as an output
        LATCbits.LATC0 = 1;       // set RC0 High to turn on LED
        Nop();                    // Insert one instruction cycle delay
        LATCbits.LATC0 = 0;       // set RC0 Low to turn off LED
}
```

Discussion:

The program improves on pgm2.c by allowing LED to stay on for 1 instruction cycle, before turning off the LED. Considering typical clock frequencies, it is still too fast for human eye, but it can be easily tested in simulator.

Running a (section of) program indefinitely

pgm4.c – Flashing a LED

Purpose:

The program is intended to continuously flash a LED connected to RC0 of PIC18F2520 microcontroller.

LED Operation: RC0 pin High – LED On
RC0 pin High – LED On

Program:

```
// Name: SG
// pgm4.c
#include <p18F2520.h>
void main (void)
{
        TRISCbits.TRISC0 = 0;     // set RC0 as an output
        while (1)                 // set up an indefinite (forever) loop
        {                         // start of the block of statements in the loop
                LATCbits.LATC0 = 1;       // set RC0 High to turn on LED
                Nop();                    // Insert one instruction cycle delay
                LATCbits.LATC0 = 0;       // set RC0 Low to turn off LED
        }                         // end of the block of statements in the loop
}
```

Discussion:

The program essentially uses the code in pgm3.c and encases all the statements that flash the LED (turn the LED on, an instruction cycle delay, and turn the LED off) inside a while loop.

- LATCbits.LATC0 = 1; is first executed
- Nop (); is the next statement to execute
- LATCbits.LATC0 = 0; is the last statement to execute
- On reaching the closing brace, the execution is repeated starting with the first statement.

Note: It is essential to have a 'while (1);' statement at the end of main function, provided it does not contain an indefinite loop.

The main function in pgm1.c should (correctly) be written as:

(Modified pgm1.c)

```
void main (void)
{       TRISCbits.TRISC0 = 0;     // set RC0 as an output
        LATCbits.LATC0 = 1;       // set RC0 High to turn on LED
        LATCbits.LATC0 = 0;       // set RC0 Low to turn off LED
        while (1);                // keep executing this loop forever
}
```

Explanation

- The developed program is burnt in program memory (Flash)to be executed on a POR.
- In practically all cases, the program placed in program memory is much smaller than the program memory capacity.
- On a POR, the MCU reads the first instruction from location 0x0000 and starts executing the instruction.
- Once a microcontroller has completed executing an instruction, it reads the next instruction as laid out in the program and executes it.
- It continues to execute one instruction after another tirelessly.
- In a program such as the original pgm1.c, there are only 3 'C' statements. The MCU is supposed to stop execution after completing the 3rd 'C' statement.
- Unfortunately, the MCU will strive to read instructions from the next program memory location and execute it.
- Even though, these memory locations may not contain valid instructions, MCU will do its best to interpret these as codes and execute these.
- This can result is unexpected outputs to be generated at Port pins, resulting in unsafe operation.
- By placing a 'while (1);' statement, the process repeated execute this 'C' statement is prevented from trying to execute code from the subsequent program memory locations.

Practical Situation

It is very rare in a real life situation, where MCU is only required to execute a small program just once. In practically all cases, the program will contain a loop that will execute a number of 'C' statements on a repeated basis.