Variables in C

Variables change during runtime so are required to be stored in RAM..

Integers can be defined according to their length and sign as given below…

```
unsigned char MyVar; //8-bit unsigned integer (0,... 255)
char MyVar;           //8-bit signed integer (-128,…127)
signed char MyVar;  //8-bit signed integer (-128,…127)
unsigned int MyVar; //16-bit unsigned integer (0,..65535)
int  MyVar;          //16-bit signed integer(-32768,....32767)
signed int MyVar;  //16-bit signed integer(-32768,.....32767)
unsigned long (int) MyVar; //32-bit unsigned integer (0,...4294967295)
long (int) MyVar; //32-bit signed integer (-21474836476,… 21474836475)
signed long (int) MyVar; //32-bit signed integer(-21474836476,.21474836475)
```

Variables in C

```
unsigned long long MyVar; //64-bit unsigned integer
long long MyVar;          //64-bit signed integer
signed long long MyVar;   //64-bit signed integer
```

When we need to use the variables with a floating part. We can use…

```
float MyVar;  //32-bit floating point
double MyVar; //64-bit floating point
long double MyVar; //64-bit floating point
bool MyVar;        //8-bit boolean type variable, which means logical 0 or 1
                   //requires to add #include <stdbool.h>
```

Variables in C

Static and Volatile variables are different.

A **static** variable is simply a variable that exists for the lifetime of the application. Static variables can be global: defined outside of a function and accessible everywhere, or local: defined within a function and only accessible from within that function.

Local static variables are created on the first invocation of the function and remain in memory for the function to use when next called.

The `static` keyword enforces the compiler to ensure that the RAM for the variable is always reserved and not reused for other variables.

So if you need a variable that is only used within a function and only updated by that function and you need the value of that variable to remain for the next time that you call the function then you need to define it as a static variable.

```
static int MyVar = 0;
```

Variables in C

A **volatile** variable is one that can be changed without the compilers knowledge for example by an interrupt. This means that whenever this variable is accessed by the program it must be reread from the actual memory location in case it has changed, rather than simply read from the cache or registers on the ALU if it has already been used recently by the program.

```
volatile int MyVar = 0;
```

A variable can be both static (always present) and volatile (changed from anywhere).

```
static volatile int MyVar = 0;
```

Bitwise Logic operators in C

Operator	Description	Example
~	Complement the variable bit-by-bit	`MyVar =~ MyVar; //complements all bits`
|	OR the variable bit-by-bit	`MyVar = MyVar | 0b00000001; //Set the last bit`
&	AND the variable bit-by-bit	`MyVar =MyVar & 0b1111111**0**; //Clear the last bit`
^	XOR the variable bit-by-bit	`MyVar =MyVar^0b**1**0000000; //Toggle the first bit`
<<	Rotate left the variable n times **arithmetically**	`MyVar = MyVar<<4; //Rotate left 4 times`
>>	Rotate right the variable n times **arithmetically**	`MyVar = MyVar>>3; //Rotate right 3 times`

The table given above summarizes all the Bitwise Logic Operators in C

Bitwise Logic Operators in C

Ex: Run the following program and observe the changings in e and f variables.

```
#include <msp430.h>
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
int e= 0b1111111111110000;
int f= 0x0001;
while(1)
{
    e=~e; //Complement all the bits in e,                    e=0000 0000 0000 1111

    e=e|BIT7;  //Set bit 7 by using OR with 0b10000000,      e=0000 0000 1000 1111
    e=e&~BIT0; //Clear bit 0 by using AND with 0b11111110,   e=0000 0000 1000 1110
    e=e^BIT4;  //Toggle bit 4 by using XOR with 0b00010000,  e=0000 0000 1001 1110

    e|=BIT6;  //Set bit 6 by using OR with 0b01000000,       e=0000 0000 1101 1110
    e&=~BIT1; //Clear bit 1 by using AND with 0b11111101,    e=0000 0000 1101 1100
    e^=BIT3;  //Toggle Bit 3 by using XOR with 0b00001000,   e=0000 0000 1101 0100

    f=f<<1; //Rotate f left 1 time arithmetically,    f=0x0002, f=0b0010
    f=f<<2; //Rotate f left 2 times arithmetically,   f=0x0008, f=0b1000
    f=f>>1; //Rotate f right 1 time arithmetically,   f=0x0004, f=0b0100
}
return 0;
}
```

Conditional Statements (If)

```
if(condition) // condition to satisfy
    {
            . // codes
            . // to
            . // execute
    }
```

If the condition inside the parentheses is satisfied, the codes between { } are executed. Otherwise, program flow continues.

Conditional Statements (If)

Ex. Write a C program that turns the LED on, which is connected to P4.7 when button on P2.1 is pressed.

```
#include <msp430.h>
#define BUTTON   P2IN
#define LED      P4OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P2DIR=0x00; //P2 is input
P4DIR=0xFF; //P4 is output
P4OUT=0x00; //Clear P4
while(1) //Always check!, otherwise it checks once and ends the program
{
    if (BUTTON==0xFD) //Means "If button is pressed
    {
        LED |= BIT7; //LED is ON
    }
   }
return 0;
}
```

Conditional Statements (If)

* In the previous example, even if the button is released (after pressing) program does nothing or do not know how to handle and the LED keeps ON. It is also possible to handle if the condition is not satisfied.
* Examine the example given nearby!
* In the example, if button is pressed, LED is ON. If not pressed, LED is OFF.

```
#include <msp430.h>
#define BUTTON   P2IN
#define LED      P4OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P2DIR=0x00; //P2 is input
P4DIR=0xFF; //P4 is output
P4OUT=0x00; //Clear P4
while(1)// Always check!, otherwise it checks once and ends the program
{
    if (BUTTON==0xFD) //If button is pressed
    {
        LED |= BIT7; //LED is ON
    }
    LED &=~BIT7; //LED is OFF if button is not pressed
  }
return 0;
}
```

Conditional Statements (If)

- Since the total lines of code for **if** statement is only one. It is not necessary to use {} symbols.
- Check the same example without {}

```
#include <msp430.h>
#define BUTTON   P2IN
#define LED      P4OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P2DIR=0x00; //P2 is input
P4DIR=0xFF; //P4 is output
P4OUT=0x00; //Clear P4
while(1) //Always check!, otherwise it checks once and ends the program
{
    if (BUTTON==0xFD) //If button is pressed
        LED |= BIT7; //Condition is satisfied, LED is ON
        LED &=~BIT7; //Condition is NOT satisfied, LED is OFF
  }
return 0;
}
```

Conditional Statements (If-else)

```
if(condition) //condition to satisfy
      {
            . // codes
            . // to
            . // execute
      }
else //if the condition is not satisfied
      {
            . // codes
            . // to
            . // execute
      }
```

* In if-else statement, if the condition is not satisfied, program flowing continues with the block of codes in `else` statement.

Conditional Statements (If-else)

```
#include <msp430.h>
#define BUTTON    P2IN
#define LED2      P4OUT
#define LED1      P1OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P2DIR=0x00; P4DIR=0xFF; P4OUT=0x00; P1DIR=0xFF; P1OUT=0x00;
while(1) //Always check!, otherwise it checks once and ends the program
{
    if (BUTTON==0xFD) //If button on P2.1 is pressed
    {
        LED1 |= BIT0; //LED on P1.0 is ON
        LED2 &=~BIT7; //LED on P4.7 is OFF
    }

    else //If button on P2.1 is NOT pressed
    {
        LED2 |= BIT7; //LED on P4.7 is ON
        LED1 &=~BIT0; //LED on P1.0 is OFF

    }
    }

return 0;
}
```

* In if-else statement, if the condition is not satisfied, program flowing continues with the block of codes in `else` statement.

* If the number of lines of code to execute is more than 1 line and the condition is not satisfied, `else` statement can really be useful.

Conditional Statements (If-else if- else)

```
if(condition 1) //condition to satisfy
        {
              . // codes
              . // to
              . // execute
         }
else if (condition 2) //if the cond. 1 is not but cond.2 is satisfied
        {
              . // codes
              . // to
              . // execute
        }
else if (condition 3) //if cond. 1 and cond.2 are not but cond.3 is satisfied
        {
              . // codes
              . // to
              . // execute
        }
              .
              .
              .
else  //If none of the conditions are not satisfied
        {
              . // codes
              . // to
              . // execute
        }
```

- In **`if - else if - else`** statement, if the condition1 is not satisfied, program flow continues with the next condition **`(else if)`** checking and continues in this way until a condition is satisfied. Code block of satisfied condition is executed. **(Just like in C++)**
- If none of the condition is satisfied, code block belonging to **`else`** is executed.
- Only one condition can be satisfied at the same time.
- Conditions MUST NOT overlap!

Conditional Statements (If-else if- else)

- **Ex:** Write a C language program that..
- Turns ON both LEDs when both buttons are pressed
- Turns OFF both LEDs when no both buttons are pressed.
- If first button is pressed and second one is not, turns the first LED ON and the second OFF.
- If second button is pressed and first one is not, turns the first LED OFF and the second ON.
- **Default values on P2IN is FFH and P1IN is 02H.**
- **When the button on P2 and button on P1 are pressed, they create a logic 0**

```
#include <msp430.h>
#define BUTTON1   P2IN
#define BUTTON2   P1IN
#define LED2      P4OUT
#define LED1      P1OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P2DIR=0x00; P4DIR=0xFF; P4OUT=0x00; P1DIR=0xFD; P1OUT=0x00;
while(1) //Always check!, otherwise it checks once and ends the program
{
    if (BUTTON1==0xFD && BUTTON2==0) //If both buttons are pressed
     {
        LED1 |= BIT0; //LED on P1.0 is ON
        LED2 |= BIT7; //LED on P4.7 is ON
     }
    else if (BUTTON1==0xFD && BUTTON2==2) //If BTN1 is pressed, BTN2 is not pressed
     {
        LED1 |= BIT0; //LED on P1.0 is ON
        LED2 &=~BIT7; //LED on P4.7 is OFF
     }
    else if (BUTTON1==0xFF && BUTTON2==0) //If BTN2 is pressed, BTN1 is not pressed
     {
        LED1 &=~BIT0; //LED on P1.0 is OFF
        LED2 |= BIT7; //LED on P4.7 is ON
     }
     else //If none is pressed
      {
            LED1 &=~BIT0; //LED on P1.0 is OFF
            LED2 &=~BIT7; //LED on P4.7 is OFF
      }
  }
return 0;
}
```

Conditional Statements (while)

```
while(condition) //condition to satisfy
        {
                . // codes
                . // to
                . // execute
        }
```

While condition inside the parentheses is satisfied, the codes between { } are executed. Otherwise, program flow continues.

Conditional Statements (while)

Ex. Write a C program that turns the LED ON, which is connected to P4.7 **while** the button on P2.1 is not pressed.

```
#include <msp430.h>
#define BUTTON   P2IN
#define LED      P4OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P2DIR=0x00; //P2 is input
P4DIR=0xFF; //P4 is output
P4OUT=0x00; //Clear P4
while(1) //Always check!, otherwise it checks once and ends the program
{
    while (BUTTON!=0xFD) //means "While button is not pressed", it is connected to P2.1
    {
        LED |= BIT7; //LED is ON
    }
    LED &=~BIT7; //LED is OFF
}
return 0;
}
```

- Like in `if` statement, if the number of codes to execute is nomore than 1 line, we do not have to use {}.
- Therefore, we can delete {} for the inner while loop in the example

Conditional Statements (do while)

```
do
{ // codes
. // to
. // execute
}
while(condition); // condition to satisfy
```

In case we need to execute the code once at least or check the condition to satisfy after the execution of the code. We can employ **`do-while`** loop.

Conditional Statements (do while)

Ex: Write the C program that toggles the LEDs on P1.0 and P4.7 alternatively with some delay.

```
#include<msp430.h>
void Delay_Func(void);
void Delay_Func(void)
{
volatile unsigned long i;
i=50000;
do i--;
while(i!= 0);
}
int main(void) {
WDTCTL = WDTPW | WDTHOLD; //Stop watchdog timer
P4DIR = 0x80; //Set P4.7 is output
P1DIR = 0x01; //Set P1.0 is output
while(1)
{
P4OUT = 0x00; //P4.7 is OFF
P1OUT = 0x01; //P1.0 is ON
Delay_Func();
P4OUT = 0x80; //P4.7 is ON
P1OUT = 0x00; //P1.0 is OFF
Delay_Func();
}
}
```

Conditional Statements (for)

```
for (initialization; condition; iteration) //condition to satisfy
      {
            . // codes
            . // to
            . // execute
      }
```

In **for** loop, after each iteration, condition is checked if it is satisfied or not. As long as it is satisfied, codes inside {} are executed. Otherwise, program flow continues from the end point of the for loop.

Unlike C++, declaration of the variable in for loop must be done before the loop.

Conditional Statements (for)

Ex. Write a C program that **blinks** the LED, which is connected to P4.7 for four times when button on P2.1 is pressed.

```
#include <msp430.h>
#define BUTTON   P2IN
#define LED      P4OUT
void MyDelay(void);
void MyDelay(void) //Delay Function
    {
        volatile unsigned long int i;
        for(i=1; i<50000; i++);
    }
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P2DIR=0x00; //P2 is input
P4DIR=0xFF; //P4 is output
P4OUT=0x00; //Clear P4
while(1) //Always check!, otherwise it checks once and ends the program
{
    char j;
    if (BUTTON==0xFD) //means "If button is pressed"
    {
        for (j=0; j<8; j++)
        {
                LED ^= BIT7;
                MyDelay();
        }
}
}
return 0;
}
```

- `for` loop is terminated when j=8. Because it is expected to blink the LED four times
- Even if it is possible to declare a variable in `for` loop in C++, it is not possible in C!

Conditional Statements (switch)

```
switch(choice)
{
        case 1: //do something //

        ...
        break;
        case 2: //do something else //
        ...
        break;
        .

         .
         .

        default: //for all other values //
        break;
}
```

In **switch** loop, choice is received as a parameter and the corresponding case is executed. Program execution continues until it sees **break.** If no case is selected, default case is executed.

Conditional Statements (switch)

```
#include <msp430.h>
#define BUTTON1    P2IN
#define BUTTON2    P1IN
#define LED2       P4OUT
#define LED1       P1OUT
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P2DIR=0x00; P4DIR=0xFF; P4OUT=0x00; P1DIR=0xFD; P1OUT=0x00;

char m;
while(1) //Always check!, otherwise it checks once and ends the program
{
    if (BUTTON1==0xFD && BUTTON2==0) //If both buttons are
pressed
     m=3;
     else if (BUTTON1==0xFD && BUTTON2==2) //If BTN1 is
pressed, BTN2 not pressed
     m=2;
     else if (BUTTON1==0xFF && BUTTON2==0) //If BTN2 is
pressed, BTN1 not pressed
     m=1;
     else //If none is pressed
     m=0;
```

*** Same example of `else if` with a `switch` way**

```
switch(m)
    {
    case 1:
        LED1 &=~BIT0; //LED on P1.0 is OFF
        LED2 |= BIT7; //LED on P4.7 is ON
    break;

    case 2:
        LED1 |= BIT0; //LED on P1.0 is ON
        LED2 &=~BIT7; //LED on P4.7 is OFF
    break;

    case 3:
        LED1 |= BIT0; //LED on P1.0 is ON
        LED2 |= BIT7; //LED on P4.7 is ON
    break;
    default:
        LED1 &=~BIT0; //LED on P1.0 is OFF
        LED2 &=~BIT7; //LED on P4.7 is OFF
    }

  }
return 0;
}
```

Conditional Statements (switch)

*** Let's remove `break`s and see what happens**

```
#include <msp430.h>
#define BUTTON1    P2IN
#define BUTTON2    P1IN
#define LED2       P4OUT
#define LED1       P1OUT
void MyDelay(void);
void MyDelay(void) //Delay Function
    {
        volatile unsigned long int i;
        for(i=1; i<50000; i++);
    }
int main(void)
{
WDTCTL = WDTPW | WDTHOLD; //stop watchdog timer
P2DIR=0x00; P4DIR=0xFF; P4OUT=0x00; P1DIR=0xFD; P1OUT=0x00;
char m;
while(1) //Always check!
{
   if (BUTTON1==0xFD && BUTTON2==0) //If both buttons are pressed
     m=3;
   else if (BUTTON1==0xFD && BUTTON2==2) //If BTN1 is pressed, BTN2 not pressed
     m=2;
   else if (BUTTON1==0xFF && BUTTON2==0) //If BTN2 is pressed, BTN1 not pressed
     m=1;
     else //If none is pressed
     m=0;
```

```
switch(m)
    {
    case 1:
        LED1 &=~BIT0; //LED on P1.0 is OFF
        LED2 |= BIT7; //LED on P4.7 is ON
        MyDelay();

    case 2:
        LED1 |= BIT0; //LED on P1.0 is ON
        LED2 &=~BIT7; //LED on P4.7 is OFF
        MyDelay();

    case 3:
        LED1 |= BIT0; //LED on P1.0 is ON
        LED2 |= BIT7; //LED on P4.7 is ON
        MyDelay();

    default:
        LED1 &=~BIT0; //LED on P1.0 is OFF
        LED2 &=~BIT7; //LED on P4.7 is OFF
        MyDelay();
    }
  }
return 0;
}
```