Programming Style and Documentation

Good programming style and proper documentation make a program easy to read and help programmers prevent errors.

Programming style deals with what programs look like. A program can compile and run properly even if written on only one line, but writing it all on one line would be bad programming style because it would be hard to read. *Documentation* is the body of explanatory remarks and comments pertaining to a program. Programming style and documentation are as important as coding.Good programming style and appropriate documentation reduce the chance of errors and make programs easy to read. This section gives several guidelines.

Appropriate Comments and Comment Styles

Include a summary at the beginning of the program that explains what the program does, its key features, and any unique techniques it uses. In a long program, you should also include comments that introduce each major step and explain anything that is difficult to read. It is important to make comments concise so that they do not crowd the program or make it difficult to read.In addition to line comments (beginning with **//**) and block comments (beginning with **/* and end with */)**

Proper Indentation and Spacing

A consistent indentation style makes programs clear and easy to read, debug, and maintain. *Indentation* is used to illustrate the structural relationships between a program's components or statements. C can read the program even if all of the statements are on the same long line, but humans find it easier to read and maintain code that is aligned properly.

A single space should be added on both sides of a binary operator, as shown in the following statement:

```
Serial.println(3+4*4);        Bad style
Serial.println(3 + 4 * 4);    Good style
```

Block Styles

A *block* is a group of statements surrounded by braces. There are two popular styles, *next-line* style and *end-of-line* style, as shown below.

Next-line style

```
if ( (i % 2) == 0)
 {
   Serial.println("Number is Even");
 } else
    {
     Serial.println("Number is Odd");
    }
```

End-of-line style

```
if ( (i % 2) == 0)   {
 Serial.println("Number is Even");
 } else {
 Serial.println("Number is Odd");
}
```

The next-line style aligns braces vertically and makes programs easy to read, whereas the end-of-line style saves space and may help avoid some subtle programming errors. Both are acceptable block styles. The choice depends on personal or organizational preference. You should use a block style consistently—mixing styles is not recommended.

Programming Errors

Programming errors can be categorized into three types: syntax errors, runtime errors, and logic errors.

Syntax Errors:

Errors that are detected by the compiler are called syntax errors or compile errors. Syntax errors result from errors in code construction, such as mistyping a keyword, omitting some necessary punctuation, or using an opening brace without a corresponding closing brace. These errors are usually easy to detect because the compiler tells you where they are and what caused them.

```
whil (count <= 4)            //syntex error    while
  {
        Serial.prit(count);    //syntex error  Serial.print
        count++;
  }
```

Runtime Errors:

Runtime errors are errors that cause a program to terminate abnormally. They occur while a program is running if the environment detects an operation that is impossible to carry out.

Input mistakes typically cause runtime errors. An input error occurs when the program is waiting for the user to enter a value, but the user enters a value that the program cannot handle.

For instance, if the program expects to read in a number, but instead the user enters a string, this causes data-type errors to occur in the program. Another example of runtime errors is division by zero. This happens when the divisor is zero for integer divisions.

Logic Errors:

Logic errors occur when a program does not perform the way it was intended to. Errors of this kind occur for many different reasons.

```
tempF = (tempC * 5/9) + 32;                    //     tempF = (tempC * 9/5) + 32;
Serial.print("Temperature in Fahrenheat = ");
```

In general, syntax errors are easy to find and easy to correct because the compiler gives indications as to where the errors came from and why they are wrong. Runtime errors are not difficult to find, either, since the reasons and locations for the errors are displayed on the console when the program aborts. Finding logic errors, on the other hand, can be very challenging. In the upcoming chapters, you will learn the techniques of tracing programs and finding logic errors.

Common Errors:

Missing a closing brace, missing a semicolon, missing quotation marks for strings, and misspelling names are common errors for new programmers.

Common Error 1: Missing Braces

The braces are used to denote a block in the program. Each opening brace must be matched by a closing brace. A common error is missing the closing brace. To avoid this error, type a closing brace whenever an opening brace is typed, as shown in the following example.

```
while (count <= 4)
  {

  }
```

Common Error 2: Missing Semicolons:

Each statement ends with a statement terminator (;). Often, a new programmer forgets to place a statement terminator for the last statement in a block, as shown in the following example.

```
Serial.println("Number is Even")    // missing a semi column
```

Common Error 3: Missing Quotation Marks:

A string must be placed inside the quotation marks. Often, a new programmer forgets to place a quotation mark at the end of a string, as shown in the following example.

```
Serial.println("Number is Even );   // missing quotation mark
```

Common Error 4: Misspelling Names:

C is case sensitive. Misspelling names is a common error for new programmers. For example, the word setup is misspelled as Setup and Serial is misspelled as serial in the following code.

```
void Setup()
{
 serial.begin(9600);
}
void loop()
{
 Serial.print("Welcome");
}
```

C Programming Examples

Finding odd or even

```
void setup()
{
  Serial.begin(9600);
    int i;

  Serial.println("Enter a number...");
  while (Serial.available() == 0) {

  }
  i = Serial.parseInt();

    if ( (i % 2) == 0)
    {
     Serial.println("Number is Even");
     } else {
     Serial.println("Number is Odd");
    }
}
void loop()
{

}
```

Celsius to Fahrenheit conversion

```
void setup()
{
  Serial.begin(9600);
    float tempC;
    float tempF;

  Serial.println("Enter temerature in degC.");
  while (Serial.available() == 0) {

  }
     tempC = Serial.parseInt();

     tempF = (tempC * 9/5) + 32;
     Serial.print("Temperature in Fahrenheat = ");
     Serial.println(tempF);

}
void loop()
{

}
```

Fahrenheit to Celsius conversion

```
void setup()
{
  Serial.begin(9600);
```

```
    float tempC;
    float tempF;

  Serial.println("Enter temerature in degF.");
  while (Serial.available() == 0) {

  }
  tempF = Serial.parseInt();

     tempC = (tempF - 32 ) * 5/9;
     Serial.print("Temperature in deg C = ");
     Serial.println(tempC);

}
void loop()
{

}
```

Simple Interest Calculation

```
void setup()
{
  Serial.begin(9600);

  float p,r,t,int_amt;
  p = 100000;
  r = 12;
  t = 1;
  int_amt=(p*r*t)/100;
  Serial.print("Simple interest = ");
  Serial.println(int_amt);
}
void loop()
{

}
```

Print a block F using hash (#)

```
void setup()
{
             Serial.begin(9600);
             Serial.println(" ");
             Serial.println("######");
            Serial.println("#");
            Serial.println("#");
            Serial.println("#####");
            Serial.println("#");
            Serial.println("#");
            Serial.println("#");
}
void loop()
{

}
```

Finding area of a circle

```
void setup()
{
  Serial.begin(9600);
   int radius;
   float area;
   radius = 23;

   area = 3.14*radius*radius;
   Serial.print("Area of the Circle = ");
   Serial.print(area);

}
void loop()
{

}
```

Example of if statement

```
void setup()
{
    Serial.begin(9600);

    int x = 20;
    int y = 22;
    if (x < y)
    {
        Serial.println("Variable x is less than y");
    }
}

void loop()
{

}
```

Example of multiple if statements

```
void setup()
{
    Serial.begin(9600);
    int x = 21;
    int y = 35;
    if (x > y)
    {
           Serial.println("x is greater than y\n");
    }
    if (x < y)
    {
           Serial.println("x is less than y\n");
    }
    if (x == y)
    {
           Serial.println("x is equal to y\n");
    }

    Serial.println("End of Program");
}

void loop()
{

}
```

Example of if else statement

```
void setup()
{
    Serial.begin(9600);

   int age = 20;

   if(age >=18)
   {
           Serial.println("You are eligible for voting");
   }
   else
   {
           Serial.println("You are not eligible for voting");
   }
}
void loop()
{

}
```

Example of nested if..else

```
void setup()
{
    Serial.begin(9600);

   int var1, var2;
   var1 = 25;
   var2 = 30;
   if (var1 != var2)
   {
           printf("var1 is not equal to var2\n");
           //Nested if else
           if (var1 > var2)
           {
    Serial.println("var1 is greater than var2\n");
           }
           else
           {
           Serial.println("var2 is greater than var1\n");
           }
   }
   else
   {
         Serial.println("var1 is equal to var2\n");
   }
}

void loop()
{

}
```

Example of else..if statement

```
void setup()
{
    Serial.begin(9600);

   int var1, var2;
   var1 = 25;
   var2 = 30;

   if (var1 > var2)
   {
         Serial.println("var1 is greater than var2\n");
   }
   else if (var2 > var1)
   {
         Serial.println("var2 is greater than var1\n");
   }
   else
   {
         Serial.println("var1 is equal to var2\n");
   }
}

void loop()
{

}
```

Example of For loop

```
void setup()
{
    Serial.begin(9600);

   int i;
   for (i=1; i<=10; i++)
   {
      Serial.println(i);
   }

}
void loop()
{

}
```

Multiplication table

```
void setup()
{
  Serial.begin(9600);

  Serial.println("Enter multiplication table no.");
  while (Serial.available() == 0) {

  }
  int mulTable = Serial.parseInt();

   int i;
   for (i=1; i<=10; i++)
   {
      Serial.print(mulTable);
      Serial.print(" X ");
      Serial.print(i);
      Serial.print(" = ");
      Serial.println(mulTable * i);
   }
}
void loop()
{

}
```

Example of while loop

```
void setup()
{
  Serial.begin(9600);

   int count=1;
   while (count <= 4)
   {
           Serial.print(count);
           count++;
   }
}
void loop()
{

}
```

Example of do-while loop

```
void setup()

{
  Serial.begin(9600);

    int j=0;
           do
           {
                        Serial.print("Value of variable j is:");
```

```
        Serial.println(j);
                        j++;
           }while (j<=3);
}
void loop()
{

}
```

While vs do..while loop in C

```
void setup()
{
  Serial.begin(9600);

    int i = 0;
    while(i == 1)
    {
           Serial.println("while vs do-while");
    }
    Serial.println("Out of loop");
}
void loop()
{

}
```

Same example using do-while loop

```
void setup()
{
  Serial.begin(9600);

    int i = 0;
    do
    {
           Serial.println("while vs do-while");
    }while(i == 1);
    Serial.println("Out of loop");
}
void loop()
{

}
```

Do-while runs at least once even if the condition is false because the condition is evaluated, after the execution of the body of loop.