Pemrograman Perangkat Keras

Struktur Bahasa C

Fakultas Informatika
Telkom University

September 4, 2020

Overview

Building block

All programming language are built from the same elements [1]

- Expressions
- Statements
- Statement blocks
- Function blocks

Building block (cont.)

Expression is created by operand and operator

- Operator is mathematical and logical action
- Operand is a data that acted to operator

Form of expression

- Unary : one operand and operator
- Binary : Two operands and one operator
- Ternary : three operands and two operators

Building block (cont.)

Example of expression

- Arithmetic Operators
  - *not*(A), A<=B, A==B , A! =B, A>B, A<B, A>=B
- Relational Operators
  - A−B, A/B, A+B, A%B, A∗B, A++, A−−
- Logical Operators
  - AB, A||B, !A
- Bit-wise Operators
  - A&B, A$\tilde{;}$ A|B, A<<, A$\hat{B}$, B>>
- Assignment Operators
  - A=B, A∗ =B, A+ =B, A/=B, A− =B, A%=B, A<<=B, A>>=B, A=B, A≙B, A| =B

Building block (cont.)

Statement is a complete instructions

- All statement end with a semicolon (;)
- Consists of expressions

Introduce the variable

- Address location of memory

Example

- J = ((2*30) % 7) * 4;
- A *= A;
- A %= 7;
- A = 50 ;
- B = 10 ;
- C = ((A + B) % B) * A ;

Building block (cont.)

Statement block is a group of statements

- Block statement begin with opening brace () and end with closing brace ()
- Express the set of actions

Building block (cont.)

Example of statement block

Algorithm 1 Check the boiling water

```
1: procedure BOILING
   Input: isboil.
   if isboil == true then
2:     Call Pocedure: turnoffstove
3:     Call Pocedure: take
4:     Call Pocedure: pour
     end
     else
5:     Call Pocedure: turnonstove
     end
6:     =0
```

Building block (cont.)

Function block designed to specific task

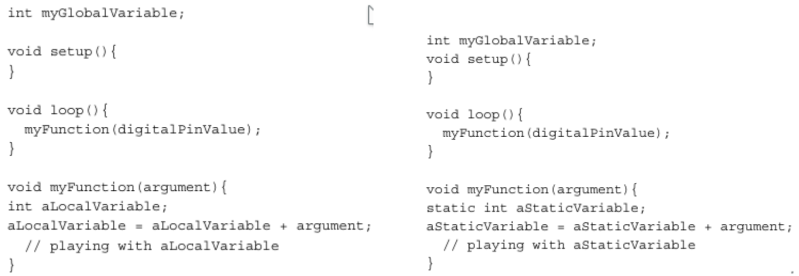
- Function statement begin with opening brace () and end with closing brace ()
- Express the specific action
- Some function need input(s)
- Returns value

Variabel and Type

- A variable is a memory storage location bounded to a symbolic name
- A type, also called data type, defines the possible nature of data
  - void,
  - boolean,
  - char,
  - string,
  - int,
  - byte,
  - word,
  - long,
  - double
  - float,
  - array

Qualifier static

When you use the static qualifier for a variable inside a function, this makes the variable persistent between two calls of the function. Declaring a variable inside a function makes the variable, implicitly, local to the function [2].



```
int myGlobalVariable;

void setup(){
}

void loop(){
  myFunction(digitalPinValue);
}

void myFunction(argument){
int aLocalVariable;
aLocalVariable = aLocalVariable + argument;
  // playing with aLocalVariable
}
```

(a) Non static qualifier

```
int myGlobalVariable;
void setup(){
}

void loop(){
  myFunction(digitalPinValue);
}

void myFunction(argument){
static int aStaticVariable;
aStaticVariable = aStaticVariable + argument;
  // playing with aStaticVariable
}
```

(b) Static qualifier

Figure 1: Example using static qualifier [2]

Qualifier Volatile

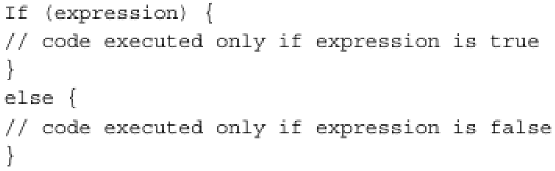
When you use the volatile qualifier in a variable declaration statement, this variable is loaded from the RAM instead of the storage register memory space of the board [2].
Basically, your code runs normally, and some instructions are triggered not by this code, but by another process such as an external event. Loading the variable from the RAM prevents some possible inconsistencies of variable value.

Qualifier Const

The const qualifier means constant. Qualifying a variable with const makes it unvariable, which can sound weird [2].
const int masterMidiChannel = 10;
same with
#define masterMidiChannel 10

IF and Else condition

This structure is very intuitive because it is very similar to any conditional pseudo code. Here the general syntax 2



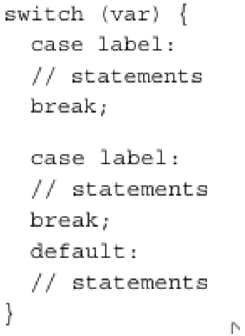
```
If (expression) {
// code executed only if expression is true
}
else {
// code executed only if expression is false
}
```

Figure 2: IF and Else condition

Expression evaluation generally results in a Boolean value.

Switch Case Break conditions

This structure use to choose one of criteria from the case. Here the general syntax 3



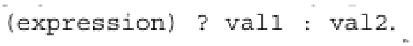
```
switch (var) {
  case label:
  // statements
  break;

  case label:
  // statements
  break;
  default:
  // statements
}
```

Figure 3: Switch and Case condition

The structure will choose the one from set case, if there are not , then the default used.

Ternary operator

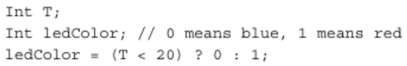
This ternary operator takes three elements as input: Expression, true value, the false value. The general syntax as shown 4



```
(expression) ? val1 : val2.
```

Figure 4: Ternary operator

The expression is tested. If it is true, this whole statement returns (or equals) val1, if it is false, it equals val2. Here some example how to use as shown 5

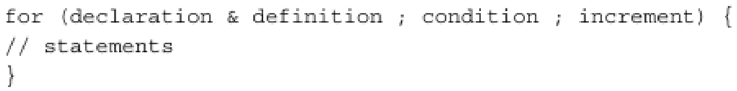


```
Int T;
Int ledColor; // 0 means blue, 1 means red
ledColor = (T < 20) ? 0 : 1;
```

Figure 5: Example of ternary operator

For loop

One counter starting from a particular value you define, and increments or decrements it until another defined value. The general syntax as shown 6

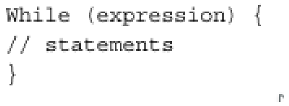


```
for (declaration & definition ; condition ; increment) {
// statements
}
```

Figure 6: For Loop

While loop

The expression is evaluated as a Boolean, true or false. While the expression is true, statements are executed, then as soon as it will be false, the loop will end. The general syntax as shown 7

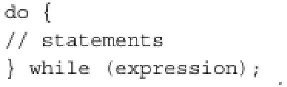


```
While (expression) {
// statements
}
```

Figure 7: While Loop

Do While loop

The do...while loop structure is very similar to the while structure, but makes its expression evaluation at the end of the loop, which means after the statements execution. The general syntax as shown 8



```
do {
// statements
} while (expression);
```

Figure 8: Do While Loop

References

[1] J. J. Purdum and B. Levy, *Beginning C for Arduino*. Springer, 2012.
[2] J. Bayle, *C programming for Arduino*. Packt Publishing Ltd, 2013.

Thank You

Thank You!