**STATEMENTS AND BLOCKS**

An expression such as x=0 or i++ or printf(...) becomes a statement when it is followed by a semicolon, as in

x=0;

i++;

printf(...);

In C, the semicolon is a statement terminator.

Braces **{** and **}** are used to group statements together into a compound statement, or block. There is no semicolon after the right brace that ends a block.

**If-Else**

if( expression )

statement1

else

statement2

where the **else** part is optional. The expression is evaluated; if it is true, **statement1** is executed. If it is false and if there is an else part, **statement2** is executed.

**Else-if**

if( expression )

statement

else if( expression )

statement

else if( expression )

statement

else if( expression )

statement

else

statement

The last **else** part handles the “none of the above” or default case where none of the other conditions are satisfied. Sometimes there is no explicit action for the **default**; in that case the trailing else can be omitted.

**Switch**

The **switch** statement is a multi-way decision that tests whether an expression matches one of a number of constant integer values, and branches accordingly.

switch ( expression )

{

case const-expr: statements

case const-expr: statements

case const-expr: statements

default: statements

}

Each **case** is labelled by one or more integer-valued constants or constant expressions. If a case matches the expression value, execution starts at that case. All expressions must be different. The case labelled **default** is executed if none of the cases are satisfied. A **default** is optional; if it is not there and if none of the cases match, no action at all takes place. **Cases** and the **default** clause can occur in any order.

**While loop**

while( expression )

statement

the **expression** is evaluated, If it is non-zero, **statement** is executed and **expression** is re-evaluated. This cycle continues until **expression** becomes zero, at which point execution resumes after **statement**.

**For loop**

for( expr1; expr2; expr3 )

statement

is equivalent to

expr1;

while( expr2 )

{

statement

expr3;

}

The three components of a **for** loop are expression. Most commonly, expr1 and expr3 are assignments of function calls and expr2 is a relational expression. Any of the three parts can be omitted, although the semicolons must remain.

for( ; ; )

{

.....

}

is an “infinite” loop.

**Do-while loop**

The **while** and **for** loops test the termination condition at the top. But the **do-while**, tests at the bottom after making each pass through the loop body; the body is always executed at least once.

do

statement

while( expression );

The **statement** is executed, then **expression** is evaluated. If it is true, **statement** is evaluated again, and so on. When the **expression** becomes false, the loop terminates.

**Break and Continue**

The ***break*** statement provides an early exit(immediate exit) from **for, while,** and **do**. A **break** causes the innermost enclosing loop or **switch** to be exited immediately.

The ***continue***statement causes the next iteration of the enclosing for, while, or do loop to begin. In the **while** and **do**, this means that the test part is executed immediately; in the **for**, control passes to the increment step. The **continue** statement applies only to loops, not to **switch**.

**Goto and Labels**

The most common situation where ***goto*** finds its place is, when we want to abandon processing some deeply nested structure, such as breaking out of two or more loops at once. The ***break*** statement cannot be used directly since it only exits from the innermost loop.

A ***label*** has same form as a variable name, and is followed by a colon. It can be attached to any **statement** in the same function as the **goto**. The scope of a label is the entire function.