Chapter 2
Control in Programming
Topics

• Expressions and Statements
• Blocks and Compound Statements
• If and If else Statements
• Relational Operators
• The While Loop
• Logical Operations
• The For Loop
• Making C Readable
Expressions and Statements

• What is a Token?

• A token is a symbol that follows a set of rules. This can be a function name, a keyword, or an identifier, or even a single character.

• After a token is recognized by a compiler, that token now has a meaning.

• A token in computer languages is like a word in a spoken language.
Expressions and Statements

• An Expression is a sequence of tokens that evaluates to a value.
• Examples:
  X
  X+1
  2*a+b
Expressions and Statements

- A statement is a sequence of tokens that ends with a ‘;’ that performs some action.
- Examples
  - $A = b_1 + b_2 - c_1$
  - $Q = a_1 * b_1 + 25$
- Statements that affect the execution of other statements are control statements.
Expressions and Statements

- **lvalue and rvalue**
  - An lvalue is an expression that has a location in memory. In other words, it is a single variable.
  - An rvalue is an expression that can be evaluated but not changed.
Expressions and Statements

- **Examples of lvalues**
  - `x`
  - `y`

- **Examples of rvalues**
  - `x`
  - `y`
  - `5*y+2`
  - `2+5*x`
Expressions and statements

• Another way of looking at lvalues and rvalues is how they are used in an assignment statement. Lvalues are on the left side, and rvalues on the right.

  x=y;
  x=2*y+5;

• But **not**

  x+y=5+z;
Blocks and Compound Statements

• A *block* is a sequence of statements inside of computer code that can act as a single statement.

• All functions consist of a block of code.

• A block is denoted by an opening brace and a closing brace

```java
{i=4;
 Y=15;
 Z=i*Y;
}
```
Blocks and Compound Statements

- If a variable is declared inside of a block of code it is a local variable of that block of code.
- Blocks can be nested inside of one another. The variable declared in the lowest block has the priority in value in C

```c
int i=5;
{
    int i=8; j = 7;
    printf("In the inner block, i is %d\n",i);
}
printf("in the outer block, i is %d\n",i);
printf("in the outer block, j is %d\n",j);
}
```

- The results is
  
  In the inner block, i is 8
  In the outer block, i is 5
If and If else Statements

• Branching
• The expression is a control expression
• The expression can evaluate to TRUE or FALSE
• In C, a value that equates to 0 is considered FALSE
• Any value that is not 0 is considered to be TRUE

if(expression)  
Statement;
If and If else Statements

- An else statement can also be added.
- If the expression is FALSE, then it automatically goes to the else block

```plaintext
if (expression)
    statement 1;
else
    statement2;
statement3;
```
If and If Else Statements

• The processed statement of an If-Else pair can also be another if-else statement.
• This is a compound if-else statement

```java
if(expression1)
    statement1;
else if (expression 2)
    statement 2;
else
    statement 3;
```
If and If else Statements

• Remember, any block can replace a single statement

```c
if (expression1)
{
    statement1;
    statement2;
}
else
{
    Statement3;
    if (expression 2)
        Statement4;
}
```
Relational Operators

- $x \leq y \quad$ x is less than or equal to y
- $x \geq y \quad$ x is greater than or equal to y
- $x = y \quad$ x is equal to y
- $x \neq y \quad$ x is not equal to y
- $x > y \quad$ x is greater than y
- $x < y \quad$ x is less than y
Relational Operators

- Operators do not work the same way as they are written.
- \( x < 13 \) is \( x \) is less than 13
- \( 8 < x < 13 \) responds as \( ((8 < x) < 13) \) which will compile, but is actually nonsense.
- If \( x \) is 10, then the first statement is TRUE, usually a 1. 1 is always less than 13.
- If \( x \) is 5, then the first statement is FALSE, usually a 0. 0 is always less than 13.
Relational Operators

• However, the code to the right would actually represent $8 < x < 13$

• Another solution to this will be discussed shortly in logical operators.

```java
if (x>8) 
  if (x<13) 
    statement;
```
A loop is when any section of code repeats itself over and over again.

A while loop controls a section of code that repeats over and over again so long as a comparative expression equates to TRUE.

while (expression)
    statement;

while (expression)
{
    statement1;
    statement2;
}
The While Loop

• Be careful with while loops

• When using a while loop, it is important to be sure that the expression can equate to FALSE after conditions are met in the program.
The While Loop

```c
void ExWhileLoop()
{
    int Counter=0;
    int Sum=0;
    while(Counter > 200)
    {
        Sum=Sum+2;
        Counter=Counter+1;
    }
}
```
Logical Operations

- Expressions can be combined or their values altered by using Logical Operators.

- AND      &&
- OR         ||
- NOT        !
Logical Operators

- AND operations mean in order for the expression to be true, both sides of the expression must be true.

- \((FALSE) \&\& (FALSE)\) becomes \(FALSE\)
- \((FALSE) \&\& (TRUE)\) becomes \(FALSE\)
- \((TRUE) \&\& (FALSE)\) becomes \(FALSE\)
- \((TRUE) \&\& (TRUE)\) becomes \(TRUE\)
Logical Operators

if ((x>3) && (x<5))
    printf("Here is the First spot\n");
else if ((x>5) && (x<8))
    printf("Here is the Second spot\n");
else
    printf("Here is the Third Spot\n");

• Which line will run if X is 5?
• Which line will run if X is 6?
• Which line will run if X is 15?
Logical Operators

- OR operations means in order for the expression to be true, only one side of the expression must be true.

- \((\text{FALSE}) \lor (\text{FALSE})\) becomes \(\text{FALSE}\)
- \((\text{FALSE}) \lor (\text{TRUE})\) becomes \(\text{TRUE}\)
- \((\text{TRUE}) \lor (\text{FALSE})\) becomes \(\text{TRUE}\)
- \((\text{TRUE}) \lor (\text{TRUE})\) becomes \(\text{TRUE}\)
Logical Operators

```c
if ((x>4) || (x<9))
    printf ("Here is the First spot\n");
else if ((x>16) || (x<34))
    printf ("Here is the Second spot\n");
else
    printf ("Here is the Third Spot\n");
```

- Which line will run if X is 8?
- Which line will run if X is 13?
- Which line will run if X is 21?
Logical Operators

• NOT operations mean that the value of the expression is reversed.

!(FALSE) becomes TRUE
!(TRUE) becomes FALSE
Logical Operators

```c
if (!(x<5))
{
    printf("First Spot Here\n");
}
else if (!(x>10))
{
    printf("Second Spot Here\n");
}

• What happens if x is 3?
• What happens if x is 20?
The For Loop

- The for loop repeats a section of code depending on the conditions specified.
- The *initialization* is the starting condition of the loop variable.
- The *test* is the end condition of the loop variable.
- The *processing* is how the loop variable is altered on each cycle of the loop.

```plaintext
for (initialization; test; processing)
    statement;

for (initialization; test; processing)
{
    statement1;
    statement2;
}
```
The For Loop

```c
int Counter;
for( Counter=0; Counter <15; Counter ++)
{
    printf("Counter =%d\n", Counter);
}
```
Making C Readable

• One of the key elements of writing a good program is not only its use and efficiency, but how it can be tested, debugged, maintained, and modified.

• Even simple programs become complicated over time, so it is important that code is readable.
Making C Readable

- Good Names for identifiers is essential
- Should be long enough to be useful, and short enough to be painless
- Should be descriptive
- Should be recognizable
Making C Readable

• Example 1

\[ y = a \times b + c; \]

is easier to read as

\[ y = \text{slope} \times x + \text{intercept}; \]

• Example 2

```c
int i;
for (i=0;i<365;i++)
{
}
```

is easier to read as

```c
int DayLoop;
for (DayLoop=0;DayLoop<365;DayLoop++)
{
}
```
Making C Readable

- Indentation
  - As blocks of code are placed one inside of another, the inner block, called a nested block,
  - Nested blocks ought to be indented to show that it is a lower level down.
  - This makes the code blocks easier to see as the program is read by a programmer
Making C Readable

```c
int LoopIndex;
for(LoopIndex=0;LoopIndex<10;LoopIndex++)
{
    if (LoopIndex<5)
    {
        printf("Value is less than 5\n");
    }
}
```
Making C Readable

- Sometimes the brace is placed at the end of the control statement
  ```c
  while (Control < 10) {
  }
  ```

- Sometimes the brace is placed on the next line
  ```c
  while (Control < 10) {
  }
  ```

- Both can be considered correct
Making C Readable

• Block Structuring
• Break down the program into sub tasks
• Try to make the sub tasks short and readable.
• It is easier to understand 10 lines of code than a 100 lines of code
Making C Readable

• Parenthesis help the person reading the code
• A compiler may not actually require parenthesis to run a section of code.
• It is good practice to use them to make the code more readable.
  if (a+b*c<d*4-22)
• or
  if ((a+b*c)<(d*4-22))
Making C Reable

- **Comments**
- **In C it is possible to add a comment, or non-executing line of text, into a program.**
- **Example**
  ```c
  while (expression) /*Begin the while loop*/
  {
  }/*End the while loop*/
  ```
Making C Readable

• Another factor in C is adding paragraphing and whitespace.

• Adding spacing to a program makes it more readable.