Expressions and Control

Maxwell James Dunne
Expressions

- Represents a single data item (e.g. character, number, etc.)
- May consist of:
  - A single entity (a constant, variable, etc.)
  - A combination of entities connected by operators (+, -, *, / and so on)
Expressions

Examples

```
a + b
x = y
speed = dist / time
z = ReadInput()
c <= 7
x == 25
count++
d = a + 5
```
Statements

• Cause an action to be carried out

• Three kinds of statements in C:
  – Expression Statements
  – Compound Statements
  – Control Statements
Expression Statements

- An expression followed by a semi-colon
- Execution of the statement causes the expression to be evaluated

Examples

\[ i = 0; \]
\[ i++; \]
\[ a = 5 + i; \]
\[ y = (m * x) + b; \]
\[ \text{printf}("\text{Slope} = \%f", m); \]
Compound Statements

- A group of individual statements enclosed within a pair of curly braces `{ and }`
- Individual statements within may be any statement type, including compound
- Allows statements to be embedded within other statements
- Does NOT end with a semicolon after `}`
- Also called Block Statements
Compound Statements

Example

```c
{
    float start, finish;
    start = 0.0;
    finish = 400.0;
    distance = finish - start;
    time = 55.2;
    speed = distance / time;
    printf("Speed = \%f m/s", speed);
}
```
Control Statements

- Used for loops, branches and logical tests
- Often require other statements embedded within them

```
while (distance < 400.0) {
    printf("Keep running!"));
    distance += 0.1;
}
```

(while syntax: `while expr statement`)
Boolean Expressions

• Boolean data type added in C99
• Boolean expressions return integers:
  – 0 expressions evaluate as false
  – non-zero expressions evaluate as true (generally 1)

```c
{  
  int x = 5;
  bool y, z;
  
  y = (x > 4);  // y = true (1)
  z = (x > 6);  // z = false (0)
  while (1);  
}
```
Boolean Expressions

Equivalent Expressions

• If a variable, constant, or function call is used alone as the conditional expression:
  \((\text{myVar}) \text{ or } (\text{Foo}()))\)

• This is the same as saying:
  \((\text{myVar} \neq 0) \text{ or } (\text{Foo}()) \neq 0)\)

• In either case, if \(\text{myVar} \neq 0\) or \(\text{Foo}() \neq 0\), then the expression evaluates as true (non-zero)
**if Statement**

**Syntax**

```
if (expression) statement
```

- *expression* is evaluated for boolean true(≠0) or false (=0)
- If true, then *statement* is executed

**Note**

Whenever you see *statement* in a syntax guide, it may be replaced by a compound (block) statement.

```
if (expression) {
    statement_1
    statement_2
}
```

Remember: spaces and new lines are not significant.
**if Statement**

Flow Diagram

**Syntax**

```
if (expression) statement
```

![Flow Diagram of the 'if' statement](image)
if Statement

Example

```c
{
    int x = 5;
    if (x) {
        printf("x = %d\n", x); // If x is TRUE (non-zero)...
    }
    while (1);
}
```

- What will print if x = 5? ... if x = 0?
- ...if x = -82?
- ...if x = 4294967296?

Maxwell James Dunne
if Statement

Testing for TRUE

- if (x) vs. if (x == 1)
  - if (x) only needs to test for not equal to 0
  - if (x == 1) needs to test for equality with 1
  - Remember: true is defined as non-zero, false is defined as zero

Example: if (x)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>011B4</td>
<td>E208C2</td>
</tr>
<tr>
<td>011B6</td>
<td>320004</td>
</tr>
</tbody>
</table>

Example: if (x == 1)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>011C0</td>
<td>804610</td>
</tr>
<tr>
<td>011C2</td>
<td>500FE1</td>
</tr>
<tr>
<td>011C4</td>
<td>3A0004</td>
</tr>
</tbody>
</table>

MOV.W 0x08C2, 0x0000
SUB.W 0x0000, #1, [0x001e]
BRA NZ, 0x0011ce
Nested `if` Statements

Example

```c
int power = 10;
float band = 2.0;
float frequency = 146.52;

if (power > 5) {
    if (band == 2.0) {
        if ((frequency > 144) && (frequency < 148)) {
            printf("Yes, it's all true!\n");
        }
    }
}
```
**if-else Statement**

**Syntax**

```markdown
if (expression) statement₁
else statement₂
```

- `expression` is evaluated for boolean true (≠0) or false (=0)
- If true, then `statement₁` is executed
- If false, then `statement₂` is executed
**if-else Statement**

**Flow Diagram**

**Syntax**

```plaintext
if (expression) statement₁
else statement₂
```

**Diagram:**

START

- **expression**
  - true → statement₁
  - false → statement₂

END

expression ≠ 0
if-else Statement

```
float frequency = 146.52;  // Frequency in MHz

if ((frequency > 144.0) && (frequency < 148.0)) {
    printf("You're on the 2 meter band\n");
} else {
    printf("You're not on the 2 meter band\n");
}
```
if-else if Statement

Syntax

if (expression₁) statement₁
else if (expression₂) statement₂
else statement₃

- $expression₁$ is evaluated for boolean true ($\neq 0$) or false ($= 0$)
- If true, then $statement₁$ is executed
- If false, then $expression₂$ is evaluated
- If true, then $statement₂$ is executed
- If false, then $statement₃$ is executed
if else if Statement

Flow Diagram

Syntax:

```c
if (expression_1) statement_1
else if (expression_2) statement_2
else statement_3
```
if-else if Statement

Example

```c
if ((freq > 144) && (freq < 148)) {
    printf("You're on the 2 meter band\n");
} else if ((freq > 222) && (freq < 225)) {
    printf("You're on the 1.25 meter band\n");
} else if ((freq > 420) && (freq < 450)) {
    printf("You're on the 70 centimeter band\n");
} else {
    printf("You're somewhere else\n");
}
```
**while Loop**

**Syntax**

```
while (expression) statement
```

- If `expression` is true, `statement` will be executed and then `expression` will be re-evaluated to determine whether or not to execute `statement` again.

- It is possible that `statement` will never execute if `expression` is false when it is first evaluated.
while Loop

Flow Diagram

Syntax

while (expression) statement

START

expression?

true

false

statement

END

START
while Loop

Example

Example (Code Fragment)

```c
int i = 0;  // Loop counter initialized outside of loop

while (i < 5) {  // Condition checked at start of loop iterations
    printf("Loop iteration %d\n", i++);
}

i++;  // Loop counter incremented manually inside loop
```

Expected Output:

- Loop iteration 0
- Loop iteration 1
- Loop iteration 2
- Loop iteration 3
- Loop iteration 4
**while Loop**

- Primary looping mechanism
- Completely generic
- Frequently used for main loop of program

```c
while (HaveData()) {
    PrintData();
}

Generic loop:
```

```c
[while (1) {
    ...
}
```

Main loop:
Functions

Program Structure

```
main()
{
    ...  
    eat();
    ...  
}

eat()
{
    ...
    return;
}

be_merry()
{
    ...
    return;
}

drink()
{
    ...
    be_merry();
    return;
}
```
Functions

What is a function?

Definition

Functions are self contained program segments designed to perform a specific, well defined task.

- All C programs have one or more functions
- The main() function is required
- Functions can accept parameters from the code that calls them
- Functions return a single value (but can export more data)
- Functions help to organize a program into logical, manageable segments
Functions

Remember Algebra Class?

- Functions in C are conceptually like an algebraic function from math class...

\[ f(x) = x^2 + 4x + 3 \]

- If you pass a value of 7 to the function: \( f(7) \), the value 7 gets "copied" into \( x \) and used everywhere that \( x \) exists within the function definition: 
  \[ f(7) = 7^2 + 4*7 + 3 = 80 \]
Functions
Definitions

Syntax

Data type of
return expression

Name

type identifier(type₁ arg₁, ..., typeₙ argₙ)

Parameter List
(optional)

Body

Header

declarations

statements

return expression;

Return Value (optional)
Functions

Function Definitions: Syntax Examples

Example

```c
int Maximum(int x, int y)
{
    int z;

    z = (x >= y) ? x : y;
    return z;
}
```

Example – A more efficient version

```c
int Maximum(int x, int y)
{
    return ((x >= y) ? x : y);
}
```
Functions

Function Definitions: Return Data Type

Syntax

```c
(type identifier)(type\_1 arg\_1, ..., type\_n arg\_n)
{
    declarations
    statements
    return expression;
}
```

- A function's `type` must match the type of data in the return `expression`
Functions

Function Definitions: Return Data Type

- A function may have multiple return statements, but only one will be executed and they must all be of the same type.

Example

```c
int bigger(int a, int b)
{
    if (a > b) {
        return 1;
    } else {
        return 0;
    }
}
```
Functions

Function Definitions: Return Data Type

- The function type is **void** if:
  - The `return` statement has no **expression**
  - The `return` statement is not present at all

- This is sometimes called a **procedure function** since nothing is returned

```
void identifier(type1 arg1, ..., type n arg n)
{
    declarations
    statements
    return;  // may be omitted if nothing is being returned
}
```
Functions

Function Definitions: Parameters

- A function's parameters are declared just like ordinary variables, but in a comma delimited list inside the parentheses.
- The parameter names are only valid inside the function (local to the function).

Syntax

```c
(type identifier(type1 arg1, ..., type_n arg_n)
{
    declarations
    statements
return expression;
}
```

Function Parameters
Functions

Function Definitions: Parameters

- Parameter list may mix data types
  - `int Foo(int x, float y, char z)`

- Parameters of the same type must be declared separately – in other words:
  - `int Maximum(int x, y)` will **not** work
  - `int Maximum(int x, int y)` is correct

```c
int Maximum(int x, int y)
{
    return ((x >= y) ? x : y);
}
```
Functions

Function Definitions: Parameters

• If no parameters are required, use the keyword `void` in place of the parameter list when defining the function

Example

```c
type identifier(void)
{
    declarations
    statements
    return expression;
}
```
Functions
How to Call / Invoke a Function

Function Call Syntax

- No parameters and no return value
  \texttt{Foo();}
- No parameters, but with a return value
  \texttt{x = Foo();}
- With parameters, but no return value
  \texttt{Foo(a, b);}
- With parameters and a return value
  \texttt{x = Foo(a, b);}

\texttt{if (foo(a, b))}
Functions

Function Prototypes

• Just like variables, a function must be declared before it may be used
• Declaration must occur before main() or other functions that use it
• Declaration may take two forms:
  – The entire function definition
  – Just a function prototype – the function definition itself may then be placed anywhere in the program
Functions

Function Prototypes

- Function prototypes may be take on two different formats:
  - An exact copy of the function header:
    ```
    int Maximum(int x, int y);
    ```
  - Like the function header, but without the parameter names – only the types need be present for each parameter (bad form!):
    ```
    int Maximum(int, int);
    ```
Functions

Declaration and Use: Example 1

Example 1

```c
int a = 5, b = 10, c;

int Maximum(int x, int y)
{
    return ((x >= y) ? x : y);
}

int main(void)
{
    c = Maximum(a, b);
    printf("The max is %d\n", c)
}
```

Function is declared and defined before it is used in main()
Functions

Declaration and Use: Example 2

Example 2

```c
int a = 5, b = 10, c;

int Maximum(int x, int y);

int main(void)
{
    c = Maximum(a, b);
    printf("The max is %d\n", c);
}

int Maximum(int x, int y)
{
    return ((x >= y) ? x : y);
}
```

Function is declared with prototype before use in main()

Function is defined after it is used in main()