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;
printf("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

Control Statements

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

Example

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

(while syntax: while expr statement)
Decisions and Branching

Boolean Expressions

• C has no Boolean data type
• Boolean expressions return integers:
  – 0 if expression evaluates as FALSE
  – non-zero if expression evaluates as TRUE (usually returns 1, but this is not guaranteed)

```c
int main(void) {
    int x = 5, y, z;
    y = (x > 4);  // y = 1 (TRUE)
    z = (x > 6);  // z = 0 (FALSE)
    while (1);
}
```

Equivalent Expressions

• If a variable, constant or function call is used alone as the conditional expression:
  ```c
  (MyVar)
  or (Foo())
  ```
  This is the same as saying:
  ```c
  (MyVar != 0)
  or (Foo() != 0)
  ```
• In either case, if MyVar ≠ 0 or Foo() ≠ 0, then the expression evaluates as TRUE (non-zero)
• C Programmers almost always use the first method (laziness always wins in C)

if Statement

- expression is evaluated for boolean TRUE (≠0) or FALSE (=0)
- If TRUE, then statement is executed

```c
if (expression) statement
```

Note

Whenever you see statement in a syntax guide, it may be replaced by a compound (block) statement.
Remember: spaces and new lines are not significant.
**if Statement**

**Syntax**

```
if (expression) statement
```

**Flow Diagram**

- START
- expression ≠ 0
  - TRUE
  - expression = 0
  - FALSE
- END

---

**Example**

```
{  
    int x = 5;
    if (x)  
    {  
        printf("x = %d\n", x);  ...then print the value of x.
        while (1);
    }
}
```

- What will print if x = 5? ... if x = 0? ... if x = -82? ... if x = 65536?

---

**if Statement**

**Solution to Trick Question**

- If x = 65536, this is the same as x = 0
- Why?
  - An integer, whether signed or unsigned can only hold 16-bit values (65536 requires 17 bits)
  - signed int: -32768 to 32767 (twos complement)
  - unsigned int: 0 to 65535 = \(2^{16}-1\)

---

**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)**

8:                if (x) 011B4  E208C2     cp0.w 0x08c2 011B6  320004     bra z, 0x0011c0

**Example: if (x == 1)**

8:                if (x == 1) 011B4  E208C2     cp0.w 0x08c2 011B6  320004     bra z, 0x0011c0
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

```c
if (expression) statement_1
else statement_2
```

- `expression` is evaluated for boolean TRUE (≠0) or FALSE (=0)
- If TRUE, then `statement_1` is executed
- If FALSE, then `statement_2` is executed
**if-else if Statement**

**Syntax**

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

1. `expression1` is evaluated for boolean TRUE (≠0) or FALSE (=0)
2. If TRUE, then `statement1` is executed
3. If FALSE, then `expression2` is evaluated
4. If TRUE, then `statement2` is executed
5. If FALSE, then `statement3` is executed

---

**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");
```

---

**Lab Exercise 5**

Making Decisions: if Statements
Exercise 05
Making Decisions (if)

• Open the lab Project:
  On the class website Examples -> Lab5.zip

Open MPLAB® X and select Open Project Icon (Ctrl + Shift + O)
Open the Project listed above.
If you already have a project open in MPLAB X, close it by “right clicking” on the open project and selecting “Close”

Solution: Steps 1 and 2
/*########################################################################### # STEP 1: Increment intVariable1 if BOTH operations. ###########################################################################*/
if((floatVariable2 >= floatVariable1) && (charVariable2 >= charVariable1)) {
    intVariable1++;
    //Increment intVariable1
}

Solution: Step 3
/*########################################################################### # STEP 3: If neither of the above are true, set charVariable2 equal to 1. # HINT: else) ###########################################################################*/
else if(floatVariable1 > 50) {
    intVariable2--;
    //Decrement intVariable2
}

ccharVariable2 = 1;
//Set charVariable2 equal to 1

Solution: Step 4
/*########################################################################### # STEP 4: If the above is not true, and floatVariable1 is greater than or equal to floatVariable2 then decrement intVariable2. (HINT: else if) ###########################################################################*/
else if(floatVariable1 > floatVariable2) {
    intVariable2--;
    //Decrement intVariable2
}
Exercise 05
Conclusions

- if statements make it possible to conditionally execute a line or block of code based on a logic equation
- else if / else statements make it possible to present follow-up conditions if the first one proves to be false

switch Statement

Syntax

```
switch (expression)
{
  case const-exp_1: statements_1
  ...
  case const-exp_n: statements_n
  default: statements_{n+1}
}
```

- `expression` is evaluated and tested for a match with the `const-exp` in each `case` clause
- The `statements` in the matching `case` clause is executed

switch Statement

Flow Diagram (default)

Notice that each statement falls through to the next
This is the default behavior of the `switch` statement

Flow Diagram (modified)

Adding a `break` statement to each statement block will eliminate fall through, allowing only one case clause's statement block to be executed
**switch Statement**

**Example 1**

```c
switch (channel)
{
    case 2: printf("WBHM Chicago\n"); break;
    case 3: printf("DVD Player\n"); break;
    case 4: printf("WTMJ Milwaukee\n"); break;
    case 5: printf("WMAQ Chicago\n"); break;
    case 6: printf("WITI Milwaukee\n"); break;
    case 7: printf("WLS Chicago\n"); break;
    case 9: printf("WGN Chicago\n"); break;
    case 10: printf("WMVS Milwaukee\n"); break;
    case 11: printf("WTTW Chicago\n"); break;
    case 12: printf("WISN Milwaukee\n"); break;
    default: printf("No Signal Available\n");
}
```

**Example 2**

```c
switch (letter)
{
    case 'a':
        printf("Letter 'a' found.\n");
        break;
    case 'b':
        printf("Letter 'b' found.\n");
        break;
    case 'c':
        printf("Letter 'c' found.\n");
        break;
    default: printf("Letter not in list.\n");
}
```

**Example 3**

```c
switch (channel)
{
    case 4 ... 7:
        printf("VHF Station\n"); break;
    case 9 ... 12:
        printf("VHF Station\n"); break;
    case 3:
    case 8:
    case 13:
        printf("Weak Signal\n"); break;
    case 14 ... 69:
        printf("UHF Station\n"); break;
    default:
        printf("No Signal Available\n");
}
```

---

Lab Exercise 6

Making Decisions: **switch** Statements
Exercise 06
Making Decisions (switch)

• Open the lab Project:

Open the lab Project:
On the class website
Examples -> Lab6.zip

Open MPLAB® X and select Open Project Icon (Ctrl + Shift + O)
Open the Project listed above.
If you already have a project open in MPLAB X, close it by “right clicking” on the open project and selecting “Close”

Exercise 06
Making Decisions (switch)

Solution: Step 1

```c
/*###########################################################################
 # TASK: Write a switch statement to print the network's initials with the
 # channel (based on Chicago TV stations).
 # * If channel = 2, print "CBS" to the output window.
 # * If channel = 5, print "NBC" to the output window.
 # * For all other channels, print "---" to the output window,
 #   where "---" is the channel number.
 # (HINT: Use printf(), and use the newline character '\n' at the end of
 #   each string you print to the output window.)
 # NOTE: The switch statement is in a loop that will execute 9 times. Each
 # pass through the loop, 'channel' will be incremented. The output
 # window should display a line of text for channels 2 to 10.
 # STEP 1: Open a switch statement on the variable 'channel'
 ############################################################################*/
switch (channel) {
}
```

Solution: Steps 2 and 3

```c
/*###########################################################################
 # STEP 2: Write case for channel = CBS (CBS is a constant defined to equal 2)
 ############################################################################*/
case CBS:
 { // If channel = CBS (CBS = 2)
   printf("CBS %d\n", channel); // Display string "CBS 2" followed by newline
   break; // Prevent fall through to next case
 }

/*###########################################################################
 # STEP 3: Write case for channel = NBC (NBC is a constant defined to equal 5)
 ############################################################################*/
case NBC:
 { // If channel = NBC (NBC = 5)
   printf("NBC %d\n", channel); // Display string "NBC 5" followed by newline
   break; // Prevent fall through to next case
 }
```
Exercise 06
Making Decisions (switch)

Solution: Steps 4 and 5

```c
/*########################################################################### 
# STEP 4: Write case for channel = ABC (ABC = 7) 
# This should look almost identical to step 2. 
#*******************************************************************************/
case ABC:
    //If channel = ABC (ABC = 7)
    {
        printf("ABC %d\n", channel); //Display string "ABC 7" followed by newline 
        //Prevent fall through to next case
        break;
    }
/*########################################################################### 
# STEP 5: Write default case. If channel is not ABC then print "--- #" 
# For all other channels
#*******************************************************************************/
default:
    //For all other channels
    {
        printf("--- %d\n", channel); //Display string "--- #" followed by newline
    }
```

Exercise 06
Conclusions

- `switch` provides a more elegant decision making structure than `if` for multiple conditions (if – else if – else if – else if...)
- The drawback is that the conditions may only be constants (match a variable's state to a particular value)

Remember that you can use ... fall through (don't put break)

for Loop

**Syntax**

```c
for (expression1; expression2; expression3)
    statement
```

- `expression1` initializes a loop count variable once at start of loop (e.g. i = 0)
- `expression2` is the test condition – the loop will continue while this is true (e.g. i <= 10)
- `expression3` is executed at the end of each iteration – usually to modify the loop count variable (e.g. i++)
for Loop

Syntax

```
for (expression_1; expression_2; expression_3)
statement
```

Example (Code Fragment)

```
int i;
for (i = 0; i < 5; i++)
{
    printf("Loop iteration %d\n", i);
}
```

Expected Output:

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

Note

- Infinite Loops
  - A for loop without any expressions will execute indefinitely (can leave loop via `break` statement)

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

Syntax

```
while (expression) statement
```

Flow Diagram

- The expression must always be there, unlike with a for loop
- while is used more often than for when implementing an infinite loop, though it is only a matter of personal taste
- Frequently used for main loop of program

**Note**

- Infinite Loops
  A while loop with expression = 1 will execute indefinitely (can leave loop via break statement)

Example (Code Fragment)

```
int i = 0;  // Loop counter initialized outside of loop
while (i < 5) // Condition checked at start of loop iterations
    {       // Loop counter incremented manually inside loop
        printf("Loop iteration %d\n", i++);
    }
```

Expected Output:

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

do-while Loop

Syntax

```
do statement while (expression);
```
**do-while Loop**

**Syntax**

```
do statement while (expression);
```

**Flow Diagram**

![Flow Diagram for do-while Loop]

**Example (Code Fragment)**

```c
int i = 0;
do {
    printf("Loop iteration %d\n", i++);
} while (i < 5);
```

**Expected Output:**

Loop iteration 0
Loop iteration 1
Loop iteration 2
Loop iteration 3
Loop iteration 4

**break Statement**

**Syntax**

```
break;
```

- Causes immediate termination of a loop even if the exit condition hasn't been met.
- Exits from a `switch` statement so that execution doesn't fall through to next `case` clause.

**Flow Diagram Within a while Loop**

![Flow Diagram for break Statement]
**break Statement**

Example (Code Fragment)

```c
int i = 0;
while (i < 10) {
    i++;
    if (i == 5) break;
    printf("Loop iteration %d\n", i);
}
```

Expected Output:
- Loop iteration 1
- Loop iteration 2
- Loop iteration 3
- Loop iteration 4

**continue Statement**

Syntax

```c
continue;
```

- Causes program to jump back to the beginning of a loop without completing the current iteration.

Flow Diagram Within a `while` Loop

```
START

expression?

TRUE

statement

FALSE

continue

statement

END
```

Example (Code Fragment)

```c
int i = 0;
while (i < 6) {
    i++;
    if (i == 2) continue;
    printf("Loop iteration %d\n", i);
}
```

Expected Output:
- Loop iteration 1
- Loop iteration 3
- Loop iteration 4
- Loop iteration 5

Iteration 2 does not print.
Lab Exercise 7

Loops

Exercise 07

Loops

• Open the lab Project:

On the class website
Examples -> Lab7.zip

Open MPLAB® X and select Open Project Icon (Ctrl + Shift + O)
Open the Project listed above.

If you already have a project open in MPLAB X, close it by “right clicking” on the open project and selecting “Close”

Solution: Step 1

/*########################################################################### # STEP 1: Create a for loop to iterate the block of code below. The loop should do the following: # "Initialize counter1 to 1 # Loop as long as counter1 is less than 5 # Increment counter1 on each pass of the loop # (HINT: for(int; test; action) # Write the opening line of the for loop # for(counter1 = 1; counter1 < 5; counter1++) # intVariable1 *= counter1; printf("FOR: intVariable1 = %d, counter1 = %d\n", intVariable1, counter1); # end of for loop block */
Loops

Exercise 07

Solution: Step 2

```c
while (charVariable1 != 0)
{
    charVariable1--; charVariable2 += 5;
    printf("WHILE: charVariable1 = %d, charVariable2 = %d\n", charVariable1, charVariable2);
}
```

Solution: Step 3

```c
do
{
    counter1 += 5; counter2 = counter1 * 3;
    printf("DO: counter1 = %d, counter2 = %d\n", counter1, counter2);
}
while (counter1 <= 100);
```

Conclusions

- C Provides three basic looping structures
  - for – checks loop condition at top, automatically executes iterator at bottom
  - while – checks loop condition at top, you must create iterator if needed
  - do...while – checks loop condition at bottom, you must create iterator if needed

Questions?
```c
double radius = 1.0;

double PI = 3.14159;

double area = PI * radius * radius;

if (area > PI)
    printf("Area is too large.");
else
    printf("Area is acceptable.");

// Example of an array

int myArray[] = {1, 2, 3, 4, 5};
```
test 2 3 5 m

if (a=3) {
    if (b=5) {
        
    }
}

int main() {
    int i;
    int n;
    
    for (i=0; i<n; i++) {
        if (a=3) {
            
        }
    }
}