



CMSC 106 Lecture Set #5

Set Started:
Thursday, September 22, 2010



"boolean" statements

- values possible: true and false
- never both and never neither
- Does not exist as a type in C
- many people use symbolic constants to define them so they look like they exist
- In C
 - 0 is false
 - any other value is true



Important Operators

■ Relational Operators

- Equality: $x == y$
- Inequality: $x != y$
- Less than: $x < y$
- Greater than: $x > y$
- Less than or equal to: $x \leq y$
- Greater than or equal: $x \geq y$

■ Logical Operators

- And: $a \&\& b$
- Or: $a || b$
- Not: $!a$



Operator Precedence

Operator	Associativity
!, unary -, ++, --	right to left
*, /, %	left to right
+, -	left to right
<, <=, >, >=	left to right
==, !=	left to right
&&	left to right
	left to right
=, +=, -=, *=, /=	right to left

* note – the unary increment and decrement operators have high precedence even when used as postfix but the time of operation doesn't get used until after



Conditions and Expressions

- **0 is considered 'false'**
- **any other value is considered 'true'**
- **equality or relational operators**
 - (`<` , `<=` , `>` , `>=` , `==` , `!=`)
 - produces result 0 or 1



The Logical Operators

- **&&** (and) (binary operator)
- **||** (or) (binary operator)
- **!** (not) (unary, prefix)
- **produce values of 0 and 1**
- Truth Tables are a good way to show what they mean.



The "if" statement

- conditional execution of the statement
- if (condition)**
statement;
- one statement ---- notice: no ; after the (condition)
 - indentation not needed for compiler – needed for people
 - Process:
 - condition is tested
 - execution continues based on the truth value of the condition
 - if true subsidiary statement is executed
 - if false subsidiary statement is skipped
 - in both cases execution continues with next statement (after entire if statement)



```
#include <stdio.h>
/* reads two ints - praises for following directions*/
int main()
{
    int x, y;
    printf("type the same positive value twice:");
    scanf("%d %d", &x, &y);
    if (x == y && x > 0)
        printf("Good Job\n");
    printf("We are done here\n");
    return 0;
}
```

- **Either praises you for following directions or just goes on (no criticism).**



Beware of the assignment operator used in an expression

```
#include <stdio.h>
/* reads two ints - praises for following
   directions(?) */
main()
{
    int x, y;
    printf("type the same positive value twice:");
    scanf("%d %d", &x, &y);
    if ((x = y) && x > 0)
        printf("Good Job\n");
    printf("We are done here\n");
    return 0;
}
```

- **Read carefully – does it really do what it says?**



The if/else statement

- The if/else contains two subsidiary statements; one is always executed.
- if (condition)**
statement1;
- else**
statement2;
- still considered "one statement" but it has 2 subsidiary statements
- Process:
 - condition is tested
 - execution continues based on the truth value of the condition
 - if true subsidiary statement 1 is executed
 - if false subsidiary statement 2 is executed
 - in both cases execution continues with next statement (after entire if statement)



Blocks / Compound Statements

- Any number of statements can be grouped inside braces {}.

```
if (num1 >= num2)
{
    printf("%d\n", num1);
    num3= num1 * num1;
}
```

- Semicolon not needed after a compound statement's }



Nested if statements

```
int month, day;
scanf("%d", &day);
if (day > 31)
    if (day <= 60)
        printf("February\n");
```

```
-----
int month, day;
scanf("%d", &day);
if (day <= 31)
    month= 1;
else
    if (day <= 60)
        month= 2;
```



Can be inside of a Block or not

```
int month, day;
scanf("%d", &day);
if (day <= 90)
{
    printf("first third of year\n");
    if (day <= 60){
        if (day <= 31){
            month= 1;
            printf("it's January\n");
        }
        else{
            month=2;
            printf("Feb\n");
        }
    }
}
else
    month = 3;
}
```



Dangling Else's

- To which if does this one else belong?

```
if (x < 10)
if (y > 10)
    printf("a\n");
else
    printf("b\n");
```



The Conditional Expression

- **C's only ternary operator**
- **condition ? expression1 : expression2**
- if condition is true expression1's value is calculated and
- becomes the whole conditional expression's value
- otherwise its value is expression2's value



Short-circuit Evaluation of Logical Operators

- Once the value of an expression can be determined – C stops the evaluation of that expression
- with && - if the left operand is false, the whole statement must be false
- with || - if the left operand is true, the whole statement must be true



Common Mistakes

- Forgetting that relational operators are only binary operators
- Assuming the && or || can do more than it can
- Assuming the ! has lower precedence than it does



The Switch Statement

- for testing one expression for equality with several different constant values.

```
switch (expression) {  
  case value1: statements1;  
  case value2: statements2;  
  .  
  .  
  .  
  case valuen: statementsn;  
}
```

- action:
 - the expression is calculated and execution jumps to case with same value as the expression's and executes statements beginning there.
 - each case can have many statements- braces not needed.