CSCI 192

Comparisons, Implied Conditions and Decision

Relational and logical operators are used to compare values in variables and constants. Conditional expressions may contain relational operators or may have an implied condition. The processing of data often requires the use of an **if** statement to decide if the calculation or another operation should be performed, or to select a particular option to be performed.

Relational operators can be used in conditional statements to determine whether or not a block will be executed.

No matter how complex, any algorithm can be constructed using a combination of four standardized flow of control structures: sequential, selection, repetition, and invocation.

The term flow of control refers to the order in which a program's statements are executed. Unless directed otherwise, the normal flow of control for all programs is sequential. This means that statements are executed in sequence, one after another, in the order in which they are placed within the program.

Selection, repetition, and invocation structures permit the sequential flow of control to be altered in precisely defined ways. The selection structure is used to select which statements are to be performed next and the repetition structure is used to select which statements are to be performed next and the repetition structure is used to repeat a set of statements.

Making Comparisons Using Relational Operators

Relational operators are used to compare data items. It is possible to compare two variables or a variable and a constant. The comparison determines the relationship between the two fields.

Relational expressional are sometimes called conditions, for short, and we use both terms to refer to these expressions. Like all C++ expressions, relational expressions are evaluated to yield a numerical result. The value of a relational expression can only be the integer value of 1 or 0, which is interpreted as true or false, respectively.

Operator	Meaning	Example
<	Less Than	age < 30
>	Greater Than	height > 6.2
<=	Less than or equal to	taxable <= 20000
>=	Greater than or equal to	temp >= 98.6
==	Equal to	grade == 100
!=	Not equal to	number != 250

Expression	Value	Interpretation
‘A’ > ‘C’	0	False
‘D’ <= ‘Z’	1	True
‘E’ == ‘F’	0	False
‘G’ >= ‘M’	0	False
‘B’ != ‘C’	1	True

Comparing Data

Numeric

```
int  iCount;
iCount == 0


float  fNumber;
fNumber < 200.00
```

Character

```
char  cLetter1 = ‘A’;
char cLetter2 = ‘a’;


cLetter2 > cLetter1
```

Numeric comparisons of data are determined to be true or false based upon the actual value of one number compared to the numerical value of the second operand.

To determine if the character condition is true or false, compare the value of the character as it is positioned in the ASCII code table (for DOS machines). This means that numeric characters would come before uppercase characters, which are then followed by lowercase characters.

String Comparisons

Strings cannot be compared using relational operations. Instead, they require the use of special functions designed for handling string data. The basic function used to compare strings is strcmp().

Name	Description	Example
strcpy(string-var, string-exp)	Copies string-exp to string-var.	strcpy(test,"efgh")
strcat(string-var, string-exp)	Appends string-exp to the end of the string value contained in string-var.	strcat(test,"there")
strlen(string-exp)	Returns the length of the string. Does not include the '\0' in the length count.	strlen("Hello World!")
strcmp(string-exp1, string-exp2	Compares string-exp1 to string-exp2. Returns a negative integer if string-exp < string-exp2, 0 if string-exp1== string-exp2, and a positive integer if string-exp1 > string-exp2.	strcmp("Bebop", "Beehive")
strncpy(string-var, string-exp,n)	Copies at most n characters of string-exp to string-var. If string-exp has fewer than n characters it pads string-var with '\0's.	strncpy(str1,str2,5)
strncmp(string-exp1, string-exp2,n)	Compares at most n characters of string-exp1 to string-exp2. Returns the same value as strcmp() based on the number of characters compared.	strncmp("Bebop","Beehive",2)
strchar(string-exp, character)	Locates the position of the first character within the string. Returns the address of the character	strchr("Hello", 'l')
stricmp(string1, string2)	Compares the contents of string1 and string2 without regard to the case of the two strings.	stricmp("Happy", "happy")

String Comparisons (cont.)

Evaluation		Return Value
(if)	string1 less than string2	Negative value
(if)	string1 equal to string2	0
(if)	string1 greater than string2	Positive value

Expression	Interpreted as
strcmp(fSalaryStatus, “Salary”) == 0	If fSalaryStatus equal “Salary”
strcmp(stString1, stString2) !=0	If stString1 not equal stString2
(strncmp(string1, “All”,3) == 0)	If string1 length of 3 is equal to “All”
strlen(string1) != 0	If string length of string1 not equal 0

Logical Operators

In addition to using simple relational expressions as conditions, more complex conditions can be created using the logical operations **AND**, **OR**, and **NOT**.

Logical Operator	Purpose
&&	And
| |	Or
!	Not

When the AND operator, &&, is used with two expressions, the condition is only if both individual expressions are true by themselves. Thus, the compound condition

(age > 40) && (term < 10)

is true (has a value of 1) only if age is greater than 40 and term is less than 10)

The logical OR operator, | |, is also applied between two expressions, When using the OR operator, the condition is satisfied if either one or both of the two expressions are true. Thus, the compound condition

(age > 40) | | (term < 10)

is true if either age is greater than 40, term is less than 10, or both conditions are true.

Note: In both cases, being that logical operators has precedence over relational operators, the parentheses are not needed.

Binary and Unary Operators

Binary Operators	Purpose
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus or Remainder

Additive Operators

Multiplicative Operators

Binary operators has two factors or operands.

Ex: iAge = iNum1 + iNum2;

The order of precedence is multiplicative operators from left to right, followed by additive operators from left to right.

The order of precedence can be altered through the use of parentheses to indicate the desired sequence of calculations. Anything inside of parentheses will be calculated first.

Unary Operators	Purpose
+ +	Increment Operator
- -	Decrement Operator

Unary operators has one factor or operand.

Ex: iNum1 + +; is the same as writing iNum = iNum + 1;

The increment and decrement operators will increase or decrease the value of an integer variable by one.

The increment and decrement operators may be used either as a prefix or as a postfix. This means that the increment may be expressed as iNum++ or ++iNum.

NOTE: This placement determines the timing of the calculation.

Table 4.2 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

Write the following as an expression.
x greater than y or x less than z
t less than or equal to 0 and y greater than 2

Determine if expression returns True of False

X = 2, y = 4, t = 0, m = 5, c = 1

1. (x > y)
2. (t < x)
3. (m != 6)
4. (y <= 1)
5. (c >=1)
6. (x == 2)
7. (t > 0)
8. (m != 5)
9. (c <= m)
10. (y < 4)
11. (t < x + c)

8. (t < 4) && (x > t)
9. (m == 5) && (y < 2)
10. (m < 8) || (t ==0)
11. (x != m) || (y < 4)
12. (t >=2) && (x >=4)

13. ((t < 4) && (x > t)) && (t == 0)
14. (m = 5) && ((y < 2) || (y = t))
15. ((m < 8) || (t =0)) && ((x != m) || (y < 4))
16. (t >=2) && (x >=4) || ((c < m) && (x <= 4)