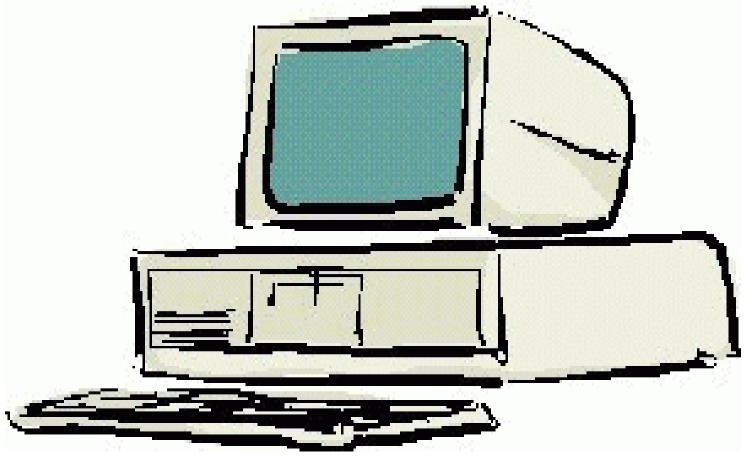
Programming in C



Session 3

Seema Sirpal
Delhi University Computer Centre

Conditional Processing

All the code in the previous examples executed, that is, from the first line of the program to the last, every statement was executed in the order it appeared in the source code.

This may be correct for some programs, but others need a way to choose which statements will be executed or run.

Conditional processing extends the usefulness of programs by allowing the use of simple logic or tests to determine which blocks of code are executed. In this lesson, a simple guessing game will be developed to illustrate the use of conditional execution.

If Statement

The if statement is used to conditionally execute a block of code based on whether a test condition is true. If the condition is true the block of code is executed, otherwise it is skipped.

```
#include <stdio.h>
int main()
{
    int number = 5;
    int guess;
    printf("I am thinking of a number between 1 and 10\n");
    printf("Enter your guess, please \n");
    scanf("%d",&guess);
    if (guess == number)
    {
        printf("Incredible, you are correct\n");
    }

    return 0;
}
```

If Statement

The if statement is used to conditionally execute a block of code based on whether a test condition is true. If the condition is true the block of code is executed, otherwise it is skipped.

```
#include <stdio.h>
int main()
{
   int number = 5;
   int guess;
   printf("I am thinking of a number between 1 and 10\n");
   printf("Enter your guess, please \n");
   scanf("%d",&guess);
   if (guess == number)
   {
      printf("Incredible, you are correct\n");
   }

   return 0;
}
```

If Statement

The "==" is called a relational operator.

Relational operators

==, !-, >, >=, <, and <=, are used to compare two operands.

The program works, but it needs some improvements. If the user enters 5 as a choice, he gets back a nice message, "Incredible, you are correct". But what happens if the user puts in an incorrect choice? Nothing. No message, no suggestions, nothing.

The else statement provides a way to execute one block of code if a condition is true, another if it is false.

If Statement

```
#include <stdio.h>

int main()
{
    int number = 5;
    int guess;

    printf("I am thinking of a number between 1 and 10\n");
    printf("Enter your guess, please\n");
    scanf("%d",&guess);
    if (guess == number)
    {
        printf("Incredible, you are correct\n");
    }
   else
    {
        printf("Sorry, try again\n");
    }
    return 0;
}
```

if / else if Statement

```
#include <stdio.h>
int main()
{
   int number = 5;
   int guess;
   printf("I am thinking of a number between 1 and 10\n");
   printf("Enter your guess, please ");
   scanf("%d",&guess);
   if (guess == number)
   {
      printf("Incredible, you are correct\n");
   }
   else if (guess < number)
   {
      printf("Higher, try again\n");
   }
   else // guess must be too high
   {
      printf("Lower, try again\n");
   }
   return 0;
}
```

Practice

Write a game program that allows a user to guess the day of your birthday. Add logic to your program to limit the guesses between 1 and 31, since the days of a month are in this limit. Print out appropriate error messages if the guess is outside of this range. Add logic and messages to guide the user towards the correct answer, that is, output "Higher" or "Lower" if the guess if wrong.

Solution

```
#include <stdio.h>
int main()
{
    int myBirthday = 13;
    int guess;
    printf("Please guess the day of my birth, from 1 to 31\n");
    printf("Enter your guess, please ");
    scanf("%d",&guess);
    /* Test for out of range guesses first */
    if (guess <= 0)
    {
        printf("Months have at least one day, Einstein\n");
    }
    else if (guess > 31)
    {
        printf("Pretty long month, genius\n");
    }
    else if (guess == myBirthday)
    {
       printf("Incredible, you are correct\n");
    }
    else if (guess < myBirthday)
    {
        printf("Higher, try again\n");
    }
    else /* Guess is in range and not greater than or equal */
    {
        printf("Lower, try again\n");
    }
    return 0;
}
```

Summary of Relational Operators

Operator	Description	Example	Evaluation
==	equal	5 == 4	FALSE
		5 == 5	TRUE
!=	not equal	5 != 4	TRUE
		5 != 5	FALSE
>	greater than	5 > 4	TRUE
		5 > 5	FALSE
>=	greater than or equal	5 >= 4	TRUE
		5 >= 5	TRUE
<	less than	5 < 4	FALSE
		5 < 5	FALSE
<=	less than or equal	5 <= 4	FALSE
		5 <= 5	TRUE

Switch Statement

The <u>switch</u> statement is a construct that is used to replace deeply nested or chained if/else statements. Nested if/else statements arise when there are multiple alternative threads of execution based on some condition.

<u>Example</u> - Suppose that an ice cream store has asked us to write a program that will automate the taking of orders. We will need to present a menu and then based on the customer's choice take an appropriate action.

Switch Statement

Using If/else

This program will work fine, but the if/else block is cumbersome.

```
#include <stdio.h>
int main()
{
    int choice;
    printf("What flavor ice cream do want?\n");
    printf("Enter 1 for chocolate\n");
    printf("Enter 2 for vanilla\n");
    printf("Enter 3 for strawberry\n");
    printf("Enter 4 for green tea flavor\n");
    printf("Enter you choice: ");
    scanf("%d",&choice);
   if (choice == 1) {
        printf("Chocolate, good choice\n");
    }
    else if (choice == 2) {
        printf("Vanillarific\n");
    }
    else if (choice == 3) {
        printf("Berry Good\n");
    }
    else if (choice == 4) {
        printf("Big Mistake\n");
    }
    else {
       printf("We don't have any\n");
        printf("Make another selection \n");
    }
    return 0;
}
```

Switch Statement

Using Switch

```
#include <stdio.h>
int main()
{
   int choice;
   printf("What flavor ice cream do want?\n");
   printf("Enter 1 for chocolate\n");
   printf("Enter 2 for vanilla\n");
   printf("Enter 3 for strawberry\n");
   printf("Enter 4 for green tea flavor, yuck\n");
   printf("Enter you choice: \n");
   scanf("%d",&choice);
   switch (choice) {
   case 1:
      printf("Chocolate, good choice\n");
      break;
   case 2:
      printf("Vanillarific\n");
      break;
   case 3:
      printf("Berry Good\n");
      break;
   case 4:
      printf("Big Mistake\n");
      break;
   default:
      printf("We don't have any\n");
      printf("Make another selection\n");
   }
  return 0;
}
```

Switch Statement

The general form of a switch statement is:

```
switch (variable) {
    case expression1:
        do something 1;
        break;
    case expression2:
        do something 2;
        break;
      ....
    default:
        do default processing;
}
```

Each expression must be a constant. The variable is compared for equality against each expression.

When an expression is found that is equal to the tested variable, execution continues until a break statement is encountered.

Switch Statement

It is possible to have a case without a break. This causes execution to fall through into the next case. This is sometimes very useful. Suppose we need to determine if a letter stored in a variable is a vowel or consonant.

```
switch (myLetter) {
    case 'A':
    case 'E':
    case 'I':
    case 'O':
    case 'U':
        vowelCnt++; /* increments vowel count */
            /* same as, vowelCnt = vowelCnt + 1; */
        break;
    default:
        consonantCnt = consonantCnt + 1;
}
```

Switch Statement

For some coding practice, how could the same logic be implemented using if and else's?

Switch Statement

Implementation without logical operators :

```
if (myLetter == 'A') {
    vowelCnt++;
}
else if (myLetter == 'E') {
    vowelCnt++;
}
else if (myLetter == 'I') {
    vowelCnt++;
}
else if (myLetter == 'O') {
    vowelCnt++;
}
else if (myLetter == 'U') {
    vowelCnt++;
}
else {
    constanantCnt++;
}
```

Switch Statement

Implementation with logical operators:

```
if ((myLetter == 'A') ||
    (myLetter == 'E') ||
    (myLetter == 'I') ||
    (myLetter == 'O') ||
    (myLetter == 'U')) {
    vowelCnt++;
}
else {
    constantantCnt++;
}
```

Logical Operators

C provides several logical operators that allow more complex relational expressions to be formed and evaluated.

Operator	Description	Example	Evaluation
&&	AND	(5 > 3) AND (5 > 6)	FALSE
&&	AND	(5 > 3) AND (5 > 4)	TRUE
||	OR	(5 > 3) OR (5 > 6)	TRUE
||	OR	(5 > 3) OR (5 > 4)	TRUE
!	NOT	!(5 > 3)	FALSE
!	NOT	!(5 > 6)	TRUE

Logical Operators

Relational operators are of higher precedence than the logical and the order of evaluation is from left to right. Here are some examples that illustrate what this means.

if (myChoice == 'A' and myAge < 25) is evaluated as
if ((myChoice == 'A') and (myAge < 25))

Suppose x = 8, y = 49, z = 1.

if (x < 7 && y > 50 || z < 2) is evaluated as
if (((x < 7) && (y > 50)) || (z < 2)) which is TRUE, not as
if ((x < 7)   &&   ((y > 50) || (z < 2)) which is FALSE.

Logical Operators

<u>**Here are a few final points to wrap up this lesson :**</u>

Even if you are sure about the order of precedence of an expression, use explicit parenthesis.

This serves to increase readability and will help avoid errors.