CS1100
Introduction to Programming

Selection Statements

Decisions with Variables

- Need for taking *logical decisions during problem solving*
  - If b^2 – 4ac negative, we should report that the quadratic has no real roots
- The *if-else* programming construct provides the facility to make logical decisions
- Syntax: *if (condition)*
  { evaluate this part if true}
  else
  { evaluate this part if false}

Conditions

- Specified using relational and equality operators
- Relational: >, <, >=, <=
- Equality: ==, !=
- Usage: for *a*,*b* values or variables
  a > *b*, *a* < *b*, *a* >= *b*, *a* <= *b*, *a* == *b*, *a* != *b*
- A condition is satisfied or true, if the relational operator, or equality is satisfied.
- For *a* = *3*, and *b* = *5*:
  - *a* < *b*, *a* <= *b*, and *a* != *b* are true
  - *a* > *b*, *a* >= *b*, *a* == *b* are false

Completing the program

```
if (discrim < 0)
    {
     printf("no real roots, only complex\n");
     exit(1);
    }
else
    {
     root1 = (−coeff2 + sqrt(discrim))/denom;
     root2 = (−coeff2 − sqrt(discrim))/denom;
    }
```

Terminates execution and returns argument (1)

Statements

Statement: a logical unit of instruction/command

Program : declarations and one or more statements

- assignment statement
- selection statement
- repetitive statements
- function calls etc.

All statements are terminated by semicolon (;)

Note: In C, semi-colon is a statement terminator rather than a separator!

Assignment statement

General Form:

variable " = " *expression* | *constant* ";"

The declared type of the variable should match the type of the result of expression/constant

Multiple Assignment:

var1 = *var2* = *var3* = *expression*;

var1 = (*var2* = (*var3* = *expression*));

Assignment operator associates right-to-left.

Compound Statements

- A group of declarations and statements collected into a single logical unit surrounded by braces
  - a block or a compound statement
- "scope" of the variable declarations
  - part of the program where they are applicable
  - the compound statement
    - variables come into existence just after declaration
    - continue to exist till end of the block
    - unrelated to variables of the same name outside the block
    - block-structured fashion

An Example

```
{
   int i, j, k;
   i = 1; j =2; k =3;
   if ( i > 0) {
      int i, k;
      i = j;
      printf("i = %d\n", i);   // output is 2
   }
   printf("i = %d\n", i);      // output is 1
}
```

This i and k and the previously declared i and k are different. Not a good programming style. But allowed by C.

Note: No semicolon after }

A compound statement can appear wherever a single statement may appear

An Example

```
{
   int i, j, k, s;
   i = 1; j =2; k =3;
   if ( i > 0) {
      int i, k, q;
      i = j;
      printf("i = %d\n", i);        // output is 2
   }                                          Note: No semicolon after }
   int i = k; // Error. Redeclaration of i.
   printf("i = %d %d\n", i, q);     // Error q's scope is not here
}
```

This i and k and the previously declared i and k are different. Not a good programming style. But allowed by C.

Selection Statements

Three forms:

single selection:

if (att < 85) grade = 'W';

no ***then*** reserved word

double selection:

if (marks < 40) passed = 0; /* false = 0 */

else passed = 1; /* true = 1 */

multiple selection:

switch statement - to be discussed later

If Statement

if (<expression>) <stmt1> [***else*** <stmt2>]

if (<expression>) {<stmt1>} [***else*** {<stmt2>}]

Semantics:

Expression evaluates to "true"

– stmt1 will be executed

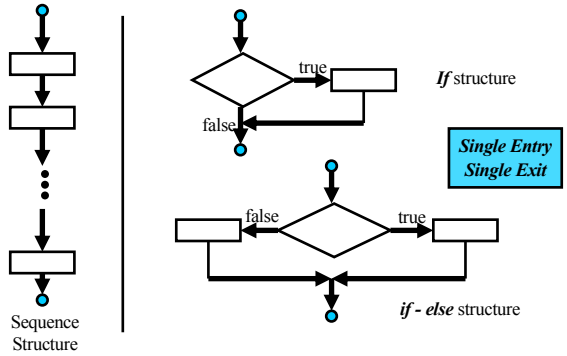
Expression evaluates to "false"

– stmt2 will be executed

Else part is optional

Expression is "true" -- stmt1 is executed; else, no action

Sequence and Selection Flowcharts



Grading Example

Below 50: D; 50 to 59: C ; 60 to 75: B; 75 above: A

```
int marks;
char grade;
…
if  (marks < 50) grade = 'D';
else if (marks <= 59) grade = 'C';
else if (marks <=75) grade = 'B';
else grade = 'A';
  …
```

Note the semicolon before else !

Unless braces are used, an else part goes with the nearest else-less if stmt

Grading Example - 2

Below 50: D; 50 to 59: C ; 60 to 75: B; 75 above: A

```
int marks;
char grade;
…
If marks>75 then grade = 'A'

else if marks>=60 grade=B
  …
```


Grading Example - 2

Below 50: D; 50 to 59: C ; 60 to 75: B; 75 above: A

```
int marks;
char grade;
…
If(marks>75) grade='A';
If( (marks>=60) && (marks<=75)) grade='B';
If( (marks>=50) && (marks<=59)) grade='C';
```


Objective

- Marks >= 40 -> Passed
- Marks < 40 -> Failed

```
if ( marks >= 40)
    printf("you passed ");
else printf("you failed");
```

- Objective 1 is changed to add
  - If Marks > 75, declare Distinction
  - No need to mention Passed

Caution in use of "else"

```
if ( marks  > 40)                          /* WRONG – else goes with second if */
   if ( marks > 75 ) printf("you got distinction");
else printf("you failed");
```

```
if ( marks  > 40) {                        /*RIGHT*/
   if ( marks > 75 ) printf("you got distinction");
   else printf("you passed\n");
}
else printf("you failed");
```


In-class problem

Sl. No.	JEE Rank	JEEADV Rank	SchoolMarks (percentage)	Scholarship
1	1-100	101-200	96-100	10,000
2	1-100	201-400	96-100	5,000
3	101-200	101-200	91-100	3,000
4	101-200	201-300	91-100	1,000
All other cases				0

QUIZ 1 SYLLABUS ENDS HERE

Switch Statement

- A multi-way decision statement
- Syntax:

```
switch ( expression ) {
    case const-expr : statements;
    case const-expr : statements;
        …
    [default: statements;]
}
```

Switch-Case Example

Dice

Roll = 2 – 12

3 – Money back

7 – Double money

11 – Triple Money

12 – Half Money

Switch-Case Code

```
#include<stdio.h>
#include<math.h>

int main()
{
 int roll;
 printf("Enter roll value:");
 scanf("%d", &roll);
 switch (roll)
   {
  case 3: printf("Money back\n");
    break;

  case 7:
    printf("Money double\n");
    break;

  case 11:
    printf("Money triple\n");
    break;

  case 12:
    printf("Money half\n");
    break;

  default:
    printf("No money back!! Ha Ha\n");
    break;
   } // Close Switch
} // Close Main
```


Counting Evens and Odds

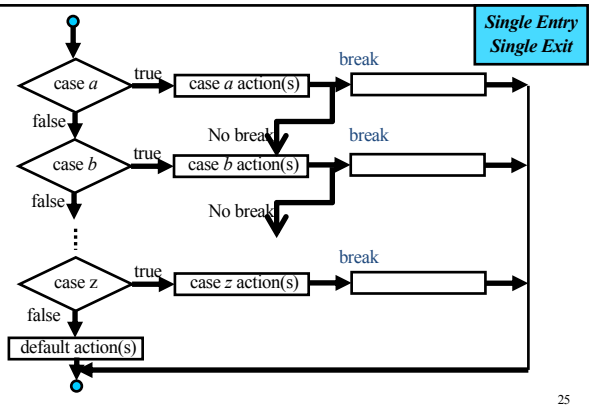
```
int  num, eCount = 0, oCount = 0;
scanf ("%d", &num);
while (num >= 0) {
   switch (num%2) {
    case  0:  eCount++; break;
    case  1:  oCount++; break;
   }
   scanf ("%d", &num);
}
printf( "Even: %d , Odd:  %d\n", eCount, oCount);
```

Counts the number of even and odd integers in the input. Terminated by giving a negative number

Fall Through

- ***Switch*** statement:
  - Execution starts at the matching case and falls through the following ***case*** statements unless prevented explicitly by ***break*** statement
  - Useful for specifying one action for several cases
- ***Break*** statement:
  - Control passes to the first statement after switch
  - A feature requiring exercise of caution

Switch Statement Flowchart



Conditional Operator (?:)

- Syntax

  (*<expression>*)? *<stmt1>*:*<stmt2>*

- Closely related to the *if – else* statement

  if (*<expression>*) *<stmt1>* ***else*** *<stat2>*

- Only ternary operator in C
- E.g.:

  (marks <40)? passed = 0 : passed = 1;

  printf (" passed = %d\n ", (marks<40)?0:1);

Programming Problems

- Write a program to check if a given number is prime.
- Write a program to count the number of digits in a given number. Your answer should contain two parts, number of digits before and after the decimal. (Can you do this only with assignments to variables, and decisions?)