Introduction to Computing

MCS1101B

Lecture 3

Recap

- Expressions
  - Arithmetic
  - Assignment
  - Logical
- Special operators
  - SizeOf
  - AddressOf (&)

- Typecasting
- **Statements**
  - Declaration
  - Assignment
  - Control
    - Branching
    - Looping
  - Input /Output

Statements in a C Program

- Parts of C program that tell the computer what to do
- Types of statements
  - Declaration statements – Declares variables etc.
  - Assignment statement – Assignment expression, followed by a ';'
  - Control statements – For branching and looping
    - Branching - if-else, switch
    - Looping - for, while, do while
  - Input/Output – Read/print, like printf/scanf

Statements (contd.)

- Compound statements
  - A sequence of statements enclosed within **{** and **}**
  - Each statement can be an assignment statement, control statement, input/output statement, or another compound statement
  - We will also call it block of statements sometimes informally

Conditional Statements

- Allow different sets of instructions to be executed depending on truth or falsity of a logical condition

aka. Branching

How do we specify conditions?

- Using expressions
  - ***non-zero value*** means the condition is ***true***
  - ***zero value*** means the condition is ***false***
- Usually logical expressions, but can be any expression
  - The value of the expression will be used

The **if** Statement

```
if (expression)
    statement;
```

*The condition to be tested is any **expression** enclosed in parentheses. The **expression** is evaluated, and if its value is **non-zero**, the statement is executed.*

```
if (expression)
    compound statement;
```

```
if (expression)
{
    statement 1;
    …
    statement n;
}
```

if else Statement

```
if (expression)
        statement/compound statement;
else
        statement/compound statement;
```

Example:

Grade Computation

Find the larger of two numbers

Find the largest of three numbers

```
if (expression)
        statement/compound statement;
else if (expression)
        statement/compound statement;
else
        statement/compound statement;
```

Nested if else

- It is not necessary for all if statements to have an else part
- Every else gets matched to the *closest preceding unmatched* if statement
- It's very easy to create *confusion* while writing a nested if-else
- So it is always a good idea to use parentheses to avoid any ambiguity

Ambiguous statement

```
if (expression)
if (expression)
        statement;
else
        statement;
```

The **conditional** operator **?:**

Another way of writing if else statement

<condition> **?** <expression1> **:**
<expression2>;

- If condition is ***true*** then expression1 is executed
- If condition is ***false*** then expression2 is executed

…used for convenience

```
int x = 10, y = 20, max;
if (x > y)
        max = x;
else
        max = y;
```

Using conditional operator…

```
max = (x > y) ? x : y;
```

The **switch** statement

- This statement can be used *instead of* writing lot of if else statement
- You can provide statements for different cases
- switch statement will ***match*** the ***value*** *of expression* with the ***case number*** and *execute statements from* ***that point onwards***

i.e. All statements below a matched case is executed

```
switch (<expression>)
{
    case <const-expr> : <statements>
    case <const-expr> : <statements>
    …
    case <const-expr> : <statements>
    default :  <statements>
}
```

Example: Evaluation of expressions

The **break** statement

- The **break** statement takes the sequence of execution out of the block
  - *Works with looping as well*
- switch-case ***does not work exactly like*** a if else if else if…
- We *use break statements to mimic the behaviour*

```
switch (<integer_value>)
{
        case <integer> : <statements>
                         break;
        case <integer> : <statements>
                         break;
        …
        case <integer> : <statements>
                         break;
        default :        <statements>
}
```

Looping Statements

- *Group of statements* that are *executed repeatedly* **while** *some condition remains true*
- **Each execution** of the *group of statements* is called an **iteration** of the loop

Examples:

- Read 5 integers and display their sum
- Find the smallest number among 100 integers
- Grade computation for entire class
- Calculate factorial of a number

The **while** statement

- The condition to be tested is any expression enclosed in parentheses
- The expression is evaluated, and if its value is non-zero, the statement is executed
- Then the expression is evaluated again and the same thing repeats
- The loop terminates when the expression evaluates to 0

```
while (expression)
    statement;

while (expression)
    <Compound statement>
```

The **while** statement (contd.)

Examples

- Sum of the first N natural numbers
- Sum of the squares of the first N natural numbers
- Compute GCD of two numbers
- Calculate maximum of many positive numbers
- Compute the sum of digits of a number

The **for** statement

```
for ( expr1; expr2; expr3)
    statement;

for ( expr1; expr2; expr3)
    <Compound statement>
```

- expr1 (init) : initialize parameter(s)
- expr2 (test): test condition, *loop continues* **if** *expression is non-zero*
- expr3 (update): used to alter the value of the parameter(s) *after each iteration*
- **statement** (body): body of loop

The **for** statement (contd.)

Example: Computing Factorial

```
int i, n=10, result;
for(i=1, result=1; i <= n; i++)
        result *= i;
printf("%d", result);
```

Try for yourself → Sum of N natural numbers

*Equivalence of **for** and **while*** ⇒

```
for ( expr1; expr2; expr3)
    <statement(s)>
```

```
expr1;
while (expr2)
{
    <statement(s)>
    expr3;
 }
```

The **do while** statement

- *Another way* of doing looping
- *Used for convenience*

Example:

Decimal to binary conversion ⇒

```
do {
statement/compound statement;
}while (expression);
```

```
int decimal=10, binary=0, rem, i=1;
do {
rem = decimal%2;
binary = (rem * i) + binary;
decimal = decimal/2;
i = i*10;
} while (decimal != 0);
```

Illustrative Example

Infinite loops and the break statement

- while (1)

  { statements }

- for (; ;)

  { statements }

- do

  { statements } while (1);

- Use break statement to come out of the loop body
  - can be used with while, do while, for, switch
  - does not work with if, else
- Causes immediate exit from a while, do/while, for or switch structure
- Program execution continues with the first statement after the structure

The **continue** statement

- Skips the remaining statements in the body of a while, for or do/while structure
  - Proceeds with the next iteration of the loop
- while and do/while loop
  - Loop-continuation test is evaluated immediately after the continue statement is executed
- for loop
  - expr3 is evaluated, then expr2 is evaluated

In The Next Class…

- Nested Loops
- You will learn about array and pointers
- You will learn more about functions