More Operators: Increment (++) and Decrement (--) 

- Both of these are unary operators; they operate on a single operand.
- The increment operator causes its operand to be increased by 1.
  - Example: `a++`, `++count` 
- The decrement operator causes its operand to be decreased by 1.
  - Example: `i--`, `--distance`
Pre-increment versus post-increment

- Operator written before the operand (++i, --i))
  - Called pre-increment operator (also sometimes called prefix ++ and prefix --)
  - Operand will be altered in value before it is utilized in the statement

- Operator written after the operand (i++, i--)
  - Called post-increment operator (also sometimes called postfix ++ and postfix --)
  - Operand will be altered in value after it is utilized in the statement
Examples

**Initial values :: a = 10; b = 20;**

\[ x = 50 + ++a; \quad a = 11, \ x = 61 \]
\[ x = 50 + a++; \quad x = 60, \ a = 11 \]
\[ x = a++ + --b; \quad b = 19, \ x = 29, \ a = 11 \]
\[ x = a++ - ++a; \quad ?? \]

Called **side effects** (while calculating some values, something else gets changed)
Precedence among different operators (there are many other operators in C, some of which we will see later)

<table>
<thead>
<tr>
<th>Operator Class</th>
<th>Operators</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>postfix++, --</td>
<td>Left to Right</td>
</tr>
<tr>
<td>Unary</td>
<td>prefix ++, --</td>
<td>Right to Left</td>
</tr>
<tr>
<td>Unary</td>
<td>! &amp;</td>
<td>Right to Left</td>
</tr>
<tr>
<td>Binary</td>
<td>* / %</td>
<td>Left to Right</td>
</tr>
<tr>
<td>Binary</td>
<td>+ -</td>
<td>Left to Right</td>
</tr>
<tr>
<td>Binary</td>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>Left to Right</td>
</tr>
<tr>
<td>Binary</td>
<td>== !=</td>
<td>Left to Right</td>
</tr>
<tr>
<td>Binary</td>
<td>&amp;&amp;</td>
<td>Left to Right</td>
</tr>
<tr>
<td>Binary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>= += -= *= /= %=</td>
<td>Right to Left</td>
</tr>
</tbody>
</table>
Doing More Complex Mathematical Operations

- C provides some mathematical functions to use
  - perform common mathematical calculations
  - Must include a special header file
    ```
    #include <math.h>
    ```
- Example
  ```
  printf("%f", sqrt(900.0));
  ```
  - Calls function `sqrt`, which returns the square root of its argument
- Return values of math functions are of type `double`
- Arguments may be constants, variables, or expressions
- Similar to functions you have seen in school maths
Math Library Functions

double acos(double x) – Compute arc cosine of x.
double asin(double x) – Compute arc sine of x.
double atan(double x) – Compute arc tangent of x.
double atan2(double y, double x) – Compute arc tangent of y/x.
double cos(double x) – Compute cosine of angle in radians.
double cosh(double x) – Compute the hyperbolic cosine of x.
double sin(double x) – Compute sine of angle in radians.
double sinh(double x) – Compute the hyperbolic sine of x.
double tan(double x) – Compute tangent of angle in radians.
double tanh(double x) – Compute the hyperbolic tangent of x.
Math Library Functions

double ceil(double x) — Get smallest integral value that exceeds x.
double floor(double x) — Get largest integral value less than x.
double exp(double x) — Compute exponential of x.
double fabs (double x) — Compute absolute value of x.
double log(double x) — Compute log to the base e of x.
double log10 (double x) — Compute log to the base 10 of x.
double pow (double x, double y) — Compute x raised to the power y.
double sqrt(double x) — Compute the square root of x.
Computing distance between two points

```c
#include <stdio.h>
#include <math.h>
int main()
{
    int x1, y1, x2, y2;
    double dist;
    printf(“Enter coordinates of first point: “);
    scanf(“%d%d”, &x1, &y1);
    printf(“Enter coordinates of second point: “);
    scanf(“%d%d”, &x2, &y2);
    dist = sqrt(pow(x1 – x2, 2) + pow(y1 – y2, 2));
    printf(“Distance = %lf
”, dist);
    return 0;
}
```

Enter coordinates of first point: 3  4
Enter coordinates of second point: 2  7
Distance = 3.162278

Output
Practice Problems

1. Read in three integers and print their average

2. Read in four integers a, b, c, d. Compute and print the value of the expression
   \[ a + \frac{b}{c} \cdot d \cdot 10 - 5 - b + 20 \cdot \frac{d}{c} \]
   - Explain to yourself the value printed based on precedence of operators taught
   - Repeat by putting parenthesis around different parts (you choose) and first do
     by hand what should be printed, and then run the program to verify if you got it
     right
   - Repeat similar thing for the expression \( a \&\& b \mid\mid c \&\& d > a \mid\mid c <= b \)

3. Read in the coordinates (real numbers) of three points in 2-d plane, and print the
   area of the triangle formed by them

4. Read in the principal amount P, interest rate I, and number of years N, and print
   the compound interest (compounded annually) earned by P after N years
Conditional Statements
Why Are Conditional Statements Required?

1. Sometimes execution of an instruction depends on the outcome of testing a condition.
   - Whether Outcome TRUE or FALSE.
   - Example: divide a by b, if b is non-zero.
   - This is also called branching.

2. Sometimes a set of instructions need to be executed repeatedly:
   - This is called looping
   - Involves branching.
   - Example: For each student compute grade
Statements in a C program

Parts of C program that tell the computer what to do

Different types

- **Declaration statements**
  - Declares variables etc.

- **Assignment statement**
  - Assignment expression, followed by a ;

- **Control statements**
  - For branching and looping, like if-else, for, while, do-while (to be seen later)

- **Input/Output**
  - Read/print, like printf/scanf

```c
int a, b, larger;
scanf("%d %d", &a, &b);
larger = b;
if (a > b){
    larger = a;
}
printf("Larger number is %d\n", larger);
```
Example

```c
int a, b, larger;
scanf("%d %d", &a, &b);
larger = b;
if (a > b)
larger = a;
printf("Larger number is %d\n", larger);
```
Compound Statements

- A sequence of statements enclosed within { and }
- Also called a **block of statements**
- Each statement in a block can be an assignment statement, control statement, input/output statement, or another compound statement
- There may be only one statement inside a block also

```c
int a, b, larger;
scanf("%d %d", &a, &b);
larger = b;
if (a > b){
    larger = a;
}
printf("Larger number is %d\n", larger);
```
Example

```c
int n;
scanf("%d", &n);
while(1) {
    if (n > 0) break;
    Compound statement
    scanf("%d", &n);
}
```
Conditional Statements

- Allow different sets of instructions to be executed depending on truth or falsity of a logical condition
- Also called **Branching**
- How do we specify conditions?
  - Using expressions
    - non-zero value means condition is **true**
    - value 0 means condition is **false**
  - Usually logical expressions, but can be any expression
    - The value of the expression will be used
- Example: if(mark>=80) grade='A'

```c
int a, b, larger;
scanf(“%d %d”, &a, &b);
larger = b;
if (a > b){
    larger = a;
}
printf(“Larger number is %d\n”, larger);
```
Branching: if Statement

if (expression)
    statement;

if (expression) {
    Block of statements;
}
Branching: if Statement

if (expression)  
statement;

if (expression) {  
Block of statements;
}

The condition to be tested is any expression enclosed in parentheses. The expression is evaluated, and if its value is non-zero, the statement/block of statements is executed.
Flow Chart Representation: if Statement

- If marks are greater than or equal to 40 (true), print "Passed"
- If marks are less than 40 (false), print "Good luck"
A decision can be made on any expression.

- Zero - False
- Nonzero - True

marks >= 40

true
print "Passed"
print "Good luck"

false
A decision can be made on any expression.

- zero - false
- nonzero - true

if (marks >= 40) {
    printf("Passed \n");
    printf("Good luck\n");
}
printf ("End\n") ;

true

false
Simple “if” statement

```java
if (aNumber != 1000)
    countA++;```

“if” with a block of statements

```c
if (aValue <= 10)
{
    printf("Answer is %8.2f\n", aValue);
    countB++;
}
// End if
```
Branching: **if-else** Statement

```java
if (expression) {
    Block of statements;
}
else {
    Block of statements;
}
```

```java
if (expression) {
    Block of statements;
}
else if (expression) {
    Block of statements;
}
else {
    Block of statements;
}
```
Quiz

- Write code for the following:
  - If CGPA within 0 and 10, increment validGrade, else display error message
  - Variables: int cgpa, validGrade;

```c
If(cgpa>=0 && cgpa<=10) 
    validGrade++;
else printf("Error\n");
```
int main() {
    int marks;
    scanf("%d", &marks);
    if (marks >= 80)
        printf ("Grade= A") ;
    else if (marks >= 70)
        printf (" Grade= B") ;
    else if (marks >= 60)
        printf (" Grade= C") ;
    else printf ("Failed");
    return 0;
}
int main () {
    int marks;
    scanf ("%d", &marks);
    if (marks >= 80) {
        printf ("A: ");
        printf ("Good Job!"');
    }
    else if (marks >= 70) printf ("B ");
    else if (marks >= 60) printf ("C ");
    else {
        printf ("Failed: ");
        printf ("Study hard!");
    }
    return 0;
}

Outputs for different inputs

90
A: Good Job!

65
C

50
Failed: Study hard!
Find the larger of two numbers

START

READ X, Y

IS X > Y?

YES

OUTPUT X

STOP

NO

OUTPUT Y

STOP
Find the larger of two numbers

```c
int main () {
    int x, y;
    scanf ("%d%d", &x, &y);
    if (x > y)
        printf ("%d\n", x);
    else
        printf ("%d\n", y);
    return 0;
}
```
Largest of three numbers

START

READ X, Y, Z

YES

IS X > Y?

Max = X

NO

Max = Y

YES

IS Max > Z?

OUTPUT Max

NO

OUTPUT Z

STOP

STOP
int main () {
    int x, y, z, max;
    scanf ("%d%d%d", &x, &y, &z);
    if (x > y) {
        max = x;
        if (max > z)
            printf ("%d", max);
        else
            printf ("%d", z);
    } else {
        max = y;
        if (max > z)
            printf ("%d", max);
        else
            printf ("%d", z);
    }
    return 0;
}
int main() {
    int a, b, c;
    scanf("%d%d%d", &a, &b, &c);
    if ((a >= b) && (a >= c))
        printf("\n The largest number is: %d", a);
    if ((b >= a) && (b >= c))
        printf("\n The largest number is: %d", b);
    if ((c >= a) && (c >= b))
        printf("\n The largest number is: %d", c);
    return 0;
}

Exercise

- Read three integers and display the integer that is neither the largest nor the smallest.
int main() {
    int i, j, k;

    scanf("%d %d %d", &i, &j, &k);

    if((i < j && i > k) || (i > j && i < k)) printf("Median=%d\n", i);
    if((j < i && j > k) || (j > i && j < k)) printf("Median=%d\n", j);
    if((k < j && k > i) || (k > j && k < i)) printf("Median=%d\n", k);

    return 0;
}

Solution
Assignment

Read principal and number of months of deposit

- Compute interest,
- less than 1 month 4%,
- Less than 1 year but more than a month 7%
- Less than 2 years but more than 1 year 8%
- Less than 5 years but more than 2 year 9%
- More than 5 years 8%
Confusing Equality (==) and Assignment (=) Operators

- **Dangerous error!**
  - Does not ordinarily cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are true, zero values are false

- **Example:**
  ```c
  if ( payCode = 4 )
      printf( "You get a bonus!\n" );
  WRONG! Will always print the line
  ```
Nesting of if-else Structures

- It is possible to nest if-else statements, one within another

- All “if” statements may not be having the “else” part

```
if (exp1) if (exp2) stmta else stmtb
```

- Confusion??

- Rule to be remembered:

```
An “else” clause is associated with the closest preceding unmatched “if”
```
Dangling else problem

Which one is the correct interpretation?

Give braces explicitly in your programs to match the else with the correct if to remove any ambiguity.
More Examples

if e1 s1
else if e2 s2

if e1 s1
else if e2 s2
else s3

if e1 if e2 s1
else s2
else s3
Lecture 5
Dangling else problem

if (exp1) if (exp2) stmta else stmtb

if (exp1) {
    if (exp2)
        stmta
    else
        stmtb
}

OR

if (exp1) {
    if (exp2)
        stmta
}
else
    stmtb

Which one is the correct interpretation?

Give braces explicitly in your programs to match the else with the correct if to remove any ambiguity.
If $e_1 s_1$
else if $e_2 s_2$
else $s_3$

If $e_1 s_1$
else if $e_2 s_2$
else $s_3$

If $e_1$ if $e_2 s_1$
else $s_2$
else $s_3$

While programming, it is always good to explicitly give the { and } to avoid any mistakes.
int main()
{
    int x;
    scanf("%d", &x);
    if (x >= 0)
        if (x <= 100)
            printf("ABC\n");
        else
            printf("XYZ\n");
    else
        printf("XYZ\n");
    return 0;
}
Example

```c
int main()
{
    int x;
    scanf("%d", &x);
    if (x >= 0)
    {
        if (x <= 100)
            printf("ABC\n");
        else
            printf("XYZ\n");
    }
    return 0;
}
```

Print “ABC” if a number is between 0 and 100, or “XYZ” if it is –ve. Do not print anything in other cases.

Outputs for different inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>XYZ</td>
<td>XYZ</td>
</tr>
<tr>
<td>-20</td>
<td>XYZ</td>
<td>XYZ</td>
</tr>
</tbody>
</table>

Not what we want, should not have printed anything

Not what we want, should have printed XYZ
int main()
{
    int x;
    scanf("%d", &x);
    if (x >= 0)
    {
        if (x <= 100)
            printf("ABC\n");
    }
    else
        printf("XYZ\n");
    return 0;
}

Correct Program

Outputs for different inputs

150

-20

XYZ
The Conditional Operator `?:`

- Cryptic if-then-else, but sometimes elegant

- Example: instead of writing

  ```java
  if (balance > 5000)
      interest = balance * 0.2;
  else interest = balance * 0.1;
  ```

  We can just write

  ```java
  interest = (balance > 5000) ? balance * 0.2 : balance * 0.1;
  ```
Ternary conditional operator (? : )

- Takes three arguments (condition, value if true, value if false)
  - Returns the evaluated value.

  (marks >= 60) ? printf( "Passed\n") : printf( "Failed\n" );

Example:
  interest = (balance > 5000) ? balance * 0.2 : balance * 0.1;

Returns a value
Express Using Ternary Operator

- if (((a > 10) && (b < 5))
  x = a + b;
else x = 0;

\[ x = ((a > 10) \&\& (b < 5)) ? a + b : 0 \]

- if (marks >= 60)
  printf("Passed \n");
else printf("Failed \n");

\( (marks \geq 60) ? \text{printf("Passed\n")} : \text{printf("Failed\n")}; \)
Exercise

- Write a ternary conditional expression to express the following:
  - If class attendance is 100% add 5 to the marks else add 3 to marks.
  - Assume following variable declaration:
    - int marks;
    - int percent_attendance;
The **switch** Statement

- An alternative to writing lots of if-else in some special cases
- This causes a particular group of statements to be chosen from several available groups based on equality tests only
- Uses **switch** statement and **case** labels
Syntax

- **expression** is any integer-valued expression
- **const-expr-1, const-expr-2,…** are any constant integer-valued expressions
  - Values must be distinct
- **S-1, S-2, …, S-m, S** are statements/compound statements
- Default is optional, and can come anywhere (not necessarily at the end as shown)

```c
switch (expression) {
    case const-expr-1: S-1
    case const-expr-2: S-2
    :
    case const-expr-m: S-m
    default: S
}
```
Behavior of **switch**

- **expression** is first evaluated
- It is then compared with **const-expr-1**, **const-expr-2**, ... for equality in order
- If it matches any one, all statements from that point till the end of the switch are executed (including statements for default, if present)
  - Use **break** statements if you do not want this (see example)
- Statements corresponding to **default**, if present, are executed if no other expression matches

```c
switch (expression) {
    case const-expr-1: S-1
    case const-expr-2: S-2
    case const-expr-m: S-m
    default: S
}
```
Example

```c
int main()
{
    int x;
    scanf("%d", &x);
    switch (x) {
        case 1: printf("One\n");
        case 2: printf("Two\n");
        default: printf("Not one or two\n");
    }
}
```

If x = 1 is entered, this will print

One
Two
Not one or two

Not what we want
int main()
{
    int x;
    scanf("%d", &x);
    switch (x) {
        case 1: printf("One\n");
              break;
        case 2: printf("Two\n");
              break;
        default: printf("Not one or two\n");
    }
}
Rounding a Digit

switch (digit) {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
        result = 0; printf("Round down\n"); break;
    case 5:
    case 6:
    case 7:
    case 8:
    case 9:
        result = 10; printf("Round up\n"); break;
}

Since there isn’t a break statement here, the control passes to the next statement without checking the next condition.

It will come here if digit is any of 0 to 4. Round to 0, then break as done.
The **break** Statement

- Used to exit from a switch or terminate from a loop
- With respect to “switch”, the “break” statement causes a transfer of control out of the entire “switch” statement, to the first statement following the “switch” statement
- Can be used with other statements also …(will discuss later)
Switch Statement: Exercise

- Read a number between 0 and 6 and display the corresponding day of the week: Sunday, Monday, … , Saturday.
switch (day){
    case 0:  printf ("Sunday\n") ;
            break ;
    case 1:  printf ("Monday\n") ;
            break ;
    case 2:  printf ("Tuesday\n") ;
            break ;
    case 3:  printf ("Wednesday\n") ;
            break ;
    case 4:  printf ("Thursday\n") ;
            break ;
    case 5:  printf ("Friday\n") ;
            break ;
    case 6:  printf ("Saturday\n") ;
            break ;
    default: printf ("Error -- invalid day.\n") ;
            break ;
}

if (day == 0 ) {
    printf ("Sunday") ;
} else if (day == 1 ) {
    printf ("Monday") ;
} else if (day == 2) {
    printf ("Tuesday") ;
} else if (day == 3) {
    printf ("Wednesday") ;
} else if (day == 4) {
    printf ("Thursday") ;
} else if (day == 5) {
    printf ("Friday") ;
} else if (day == 6) {
    printf ("Saturday") ;
} else {
    printf ("Error - invalid day.\n") ;
}
Homework 1

- Write a program that prompts the user to input the boiling point in degree Celsius.
- The program should output the substance corresponding to the boiling point listed in the table.
- The program should output the message “substance unknown” when it does not match any substance.

<table>
<thead>
<tr>
<th>Substance</th>
<th>Boiling point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water</td>
<td>100°C</td>
</tr>
<tr>
<td>Mercury</td>
<td>357°C</td>
</tr>
<tr>
<td>Copper</td>
<td>1187°C</td>
</tr>
<tr>
<td>Silver</td>
<td>2193°C</td>
</tr>
<tr>
<td>Gold</td>
<td>2660°C</td>
</tr>
</tbody>
</table>
- Read student mark out of 100.
- Print grade to be awarded as per the rule shown.

<table>
<thead>
<tr>
<th>Mark</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 35</td>
<td>F</td>
</tr>
<tr>
<td>35 to 50</td>
<td>P</td>
</tr>
<tr>
<td>50 to 60</td>
<td>D</td>
</tr>
<tr>
<td>60 to 70</td>
<td>C</td>
</tr>
<tr>
<td>70 to 80</td>
<td>B</td>
</tr>
<tr>
<td>80 to 90</td>
<td>A</td>
</tr>
<tr>
<td>90 to 100</td>
<td>EX</td>
</tr>
</tbody>
</table>
Why Use a switch Statement?

- A nested if-else structure is just as efficient as a switch statement.
  - A switch statement is easier to read.
  - Also, it is easier to add new cases to a switch statement than to a nested if-else structure.
The **break** Statement

- Used to exit from a switch or terminate from a loop
- With respect to “switch”, the “break” statement causes a transfer of control out of the entire “switch” statement, to the first statement following the “switch” statement
- Can be used with other statements also ...(will show later)
More on Data Types
More Data Types in C

Some of the basic data types can be augmented by using certain data type qualifiers:

- **short**
- **long**
- **signed**
- **unsigned**

Typical examples:

- **short int** (usually 2 bytes)
- **long int** (usually 4 bytes)
- **unsigned int** (usually 4 bytes, but cannot store + or -)
Some typical sizes (some of these can vary depending on type of machine)

<table>
<thead>
<tr>
<th>Integer data type</th>
<th>#Bits</th>
<th>Minimum value</th>
<th>Maximum value</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>8</td>
<td>$-2^7 = -128$</td>
<td>$2^7 - 1 = 127$</td>
</tr>
<tr>
<td>short int</td>
<td>16</td>
<td>$-2^{15} = -32768$</td>
<td>$2^{15} - 1 = 32767$</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>$-2^{31} = -2147483648$</td>
<td>$2^{31} - 1 = 2147483647$</td>
</tr>
<tr>
<td>long int</td>
<td>32</td>
<td>$-2^{31} = -2147483648$</td>
<td>$2^{31} - 1 = 2147483647$</td>
</tr>
<tr>
<td>long long int</td>
<td>64</td>
<td>$-2^{63} = -9223372036854775808$</td>
<td>$2^{63} - 1 = 9223372036854775807$</td>
</tr>
<tr>
<td>unsigned char</td>
<td>8</td>
<td>0</td>
<td>$2^8 - 1 = 255$</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>16</td>
<td>0</td>
<td>$2^{16} - 1 = 65535$</td>
</tr>
<tr>
<td>unsigned int</td>
<td>32</td>
<td>0</td>
<td>$2^{32} - 1 = 4294967295$</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>32</td>
<td>0</td>
<td>$2^{32} - 1 = 4294967295$</td>
</tr>
<tr>
<td>unsigned long long int</td>
<td>64</td>
<td>0</td>
<td>$2^{64} - 1 = 18446744073709551615$</td>
</tr>
</tbody>
</table>
More on the **char** type

- Is actually stored as an integer internally
- Each character has an integer code associated with it (**ASCII** code value)
- Internally, storing a character means storing its integer code
- All operators on int are allowed on char
  - $32 + \text{‘a’}$ will evaluate to $32 + 97$ (the integer ascii code of the character ‘a’) = 129
  - Same for other operators
- Can switch on chars constants in **switch**, as they are integer constants
Another example

```c
int a;
a = 'c' * 3 + 5;
printf("%d", a);
```
Will print 302 (99*3 + 5)
(ASCII code of ‘c’ = 99)

```c
char c = 'A';
printf("%c = %d", c, c);
```
Will print A = 65
(ASCII code of ‘A’ = 65)

Assigning char to int is fine. But other way round is dangerous, as size of int is larger
ASCII Code

- Each character is assigned a unique integer value (code) between 32 and 127.

- The code of a character is represented by an 8-bit unit.
  - Since an 8-bit unit can hold a total of $2^8 = 256$ values and the computer character set is much smaller than that, some values of this 8-bit unit do not correspond to visible characters.

- But not a good idea to remember exact ASCII codes while programming. Use the facts that
  - C stores characters as integers
  - Ascii codes of some important characters are contiguous (digits, lowercase alphabets, uppercase alphabets)
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
<th>Character</th>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>20</td>
<td>00100000</td>
<td>SPACE</td>
<td>80</td>
<td>50</td>
<td>01010000</td>
<td>P</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>00100001</td>
<td>!</td>
<td>81</td>
<td>51</td>
<td>01010001</td>
<td>Q</td>
</tr>
<tr>
<td>34</td>
<td>22</td>
<td>00100010</td>
<td>&quot;</td>
<td>82</td>
<td>52</td>
<td>01010010</td>
<td>R</td>
</tr>
<tr>
<td>35</td>
<td>23</td>
<td>00100011</td>
<td>#</td>
<td>83</td>
<td>53</td>
<td>01010011</td>
<td>S</td>
</tr>
<tr>
<td>36</td>
<td>24</td>
<td>00100100</td>
<td>$</td>
<td>84</td>
<td>54</td>
<td>01010100</td>
<td>T</td>
</tr>
<tr>
<td>37</td>
<td>25</td>
<td>00100101</td>
<td>%</td>
<td>85</td>
<td>55</td>
<td>01010101</td>
<td>U</td>
</tr>
<tr>
<td>38</td>
<td>26</td>
<td>00100110</td>
<td>&amp;</td>
<td>86</td>
<td>56</td>
<td>01010110</td>
<td>V</td>
</tr>
<tr>
<td>39</td>
<td>27</td>
<td>00100111</td>
<td>'</td>
<td>87</td>
<td>57</td>
<td>01010111</td>
<td>W</td>
</tr>
<tr>
<td>40</td>
<td>28</td>
<td>00101000</td>
<td>(</td>
<td>88</td>
<td>58</td>
<td>01011000</td>
<td>X</td>
</tr>
<tr>
<td>41</td>
<td>29</td>
<td>00101001</td>
<td>)</td>
<td>89</td>
<td>59</td>
<td>01011001</td>
<td>Y</td>
</tr>
<tr>
<td>42</td>
<td>2a</td>
<td>00101010</td>
<td>*</td>
<td>90</td>
<td>5a</td>
<td>01011010</td>
<td>Z</td>
</tr>
<tr>
<td>43</td>
<td>2b</td>
<td>00101011</td>
<td>+</td>
<td>91</td>
<td>5b</td>
<td>01011011</td>
<td>[</td>
</tr>
<tr>
<td>44</td>
<td>2c</td>
<td>00101100</td>
<td>,</td>
<td>92</td>
<td>5c</td>
<td>01011100</td>
<td>\</td>
</tr>
<tr>
<td>45</td>
<td>2d</td>
<td>00101101</td>
<td>-</td>
<td>93</td>
<td>5d</td>
<td>01011101</td>
<td>]</td>
</tr>
<tr>
<td>46</td>
<td>2e</td>
<td>00101110</td>
<td>.</td>
<td>94</td>
<td>5e</td>
<td>01011110</td>
<td>^</td>
</tr>
<tr>
<td>47</td>
<td>2f</td>
<td>00101111</td>
<td>/</td>
<td>95</td>
<td>5f</td>
<td>01011111</td>
<td>_</td>
</tr>
<tr>
<td>48</td>
<td>30</td>
<td>00110000</td>
<td>0</td>
<td>96</td>
<td>60</td>
<td>01100000</td>
<td>`</td>
</tr>
<tr>
<td>49</td>
<td>31</td>
<td>00110001</td>
<td>1</td>
<td>97</td>
<td>61</td>
<td>01100001</td>
<td>a</td>
</tr>
<tr>
<td>50</td>
<td>32</td>
<td>00110010</td>
<td>2</td>
<td>98</td>
<td>62</td>
<td>01100010</td>
<td>b</td>
</tr>
<tr>
<td>51</td>
<td>33</td>
<td>00110011</td>
<td>3</td>
<td>99</td>
<td>63</td>
<td>01100011</td>
<td>c</td>
</tr>
<tr>
<td>52</td>
<td>34</td>
<td>00110100</td>
<td>4</td>
<td>100</td>
<td>64</td>
<td>01100100</td>
<td>d</td>
</tr>
<tr>
<td>53</td>
<td>35</td>
<td>00110101</td>
<td>5</td>
<td>101</td>
<td>65</td>
<td>01100101</td>
<td>e</td>
</tr>
<tr>
<td>54</td>
<td>36</td>
<td>00110110</td>
<td>6</td>
<td>102</td>
<td>66</td>
<td>01100110</td>
<td>f</td>
</tr>
<tr>
<td>55</td>
<td>37</td>
<td>00110111</td>
<td>7</td>
<td>103</td>
<td>67</td>
<td>01100111</td>
<td>g</td>
</tr>
<tr>
<td>56</td>
<td>38</td>
<td>00111000</td>
<td>8</td>
<td>104</td>
<td>68</td>
<td>01101000</td>
<td>h</td>
</tr>
<tr>
<td>57</td>
<td>39</td>
<td>00111001</td>
<td>9</td>
<td>105</td>
<td>69</td>
<td>01101001</td>
<td>i</td>
</tr>
<tr>
<td>58</td>
<td>3a</td>
<td>00111010</td>
<td>:</td>
<td>106</td>
<td>6a</td>
<td>01101010</td>
<td>j</td>
</tr>
<tr>
<td>59</td>
<td>3b</td>
<td>00111011</td>
<td>;</td>
<td>107</td>
<td>6b</td>
<td>01101011</td>
<td>k</td>
</tr>
<tr>
<td>60</td>
<td>3c</td>
<td>00111100</td>
<td>&lt;</td>
<td>108</td>
<td>6c</td>
<td>01101100</td>
<td>l</td>
</tr>
<tr>
<td>61</td>
<td>3d</td>
<td>00111101</td>
<td>=</td>
<td>109</td>
<td>6d</td>
<td>01101101</td>
<td>m</td>
</tr>
<tr>
<td>62</td>
<td>3e</td>
<td>00111110</td>
<td>&gt;</td>
<td>110</td>
<td>6e</td>
<td>01101110</td>
<td>n</td>
</tr>
<tr>
<td>63</td>
<td>3f</td>
<td>00111111</td>
<td>?</td>
<td>111</td>
<td>6f</td>
<td>01101111</td>
<td>o</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>01000000</td>
<td>@</td>
<td>112</td>
<td>70</td>
<td>01100000</td>
<td>p</td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td>01000001</td>
<td>A</td>
<td>113</td>
<td>71</td>
<td>01100001</td>
<td>q</td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td>01000010</td>
<td>B</td>
<td>114</td>
<td>72</td>
<td>01100010</td>
<td>r</td>
</tr>
<tr>
<td>67</td>
<td>43</td>
<td>01000100</td>
<td>C</td>
<td>115</td>
<td>73</td>
<td>01100100</td>
<td>s</td>
</tr>
<tr>
<td>68</td>
<td>44</td>
<td>01000101</td>
<td>D</td>
<td>116</td>
<td>74</td>
<td>01101000</td>
<td>t</td>
</tr>
<tr>
<td>69</td>
<td>45</td>
<td>01000110</td>
<td>E</td>
<td>117</td>
<td>75</td>
<td>01101010</td>
<td>u</td>
</tr>
<tr>
<td>70</td>
<td>46</td>
<td>01000111</td>
<td>F</td>
<td>118</td>
<td>76</td>
<td>01101100</td>
<td>v</td>
</tr>
<tr>
<td>Code</td>
<td>Value</td>
<td>Hex</td>
<td>Value</td>
<td>Hex</td>
<td>Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>-----</td>
<td>-------</td>
<td>-----</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>47</td>
<td>01000111</td>
<td>G</td>
<td>119</td>
<td>01110111</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>48</td>
<td>01001000</td>
<td>H</td>
<td>120</td>
<td>01111000</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>49</td>
<td>01001001</td>
<td>I</td>
<td>121</td>
<td>01111001</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>4a</td>
<td>01001010</td>
<td>J</td>
<td>122</td>
<td>01111010</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>4b</td>
<td>01001011</td>
<td>K</td>
<td>123</td>
<td>01111011</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>4c</td>
<td>01001100</td>
<td>L</td>
<td>124</td>
<td>01111100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>4d</td>
<td>01001101</td>
<td>M</td>
<td>125</td>
<td>01111101</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>4e</td>
<td>01001110</td>
<td>N</td>
<td>126</td>
<td>01111110</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>4f</td>
<td>01001111</td>
<td>O</td>
<td>127</td>
<td>01111111</td>
<td>DELETE</td>
<td></td>
</tr>
</tbody>
</table>
## Quiz…

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value?</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘9’ &gt;= ‘0’</td>
<td>1 (true)</td>
</tr>
<tr>
<td>‘a’ &lt; ‘e’</td>
<td>1 (true)</td>
</tr>
<tr>
<td>‘Z’ == ‘z’</td>
<td>0 (false)</td>
</tr>
<tr>
<td>‘a’ &lt;= ‘A’</td>
<td>0 (false)</td>
</tr>
</tbody>
</table>
Example: checking if a character is a lowercase alphabet

```c
int main()
{
    /* Read a character and display whether it is lower case or upper case */
    char c1;
    scanf("%c", &c1);
    /* the ascii code of c1 must lie between the
    ascii codes of ‘a’ and ‘z’ */
    if (c1 >= ‘a’ && c1<= ‘z’)
        printf("%c is a lowercase alphabet\n", c1);
    else printf("%c is not a lowercase alphabet\n", c1);
    return 0;
}
```
Example: converting a character from lowercase to uppercase

```c
int main()
{
    char c1;
    scanf("%c", &c1);
    /* convert to uppercase if lowercase, else leave as it is */
    if (c1 >= 'a' && c1<= 'z')
        /* since ascii codes of uppercase letters are contiguous, the
         * uppercase version of c1 will be as far away from the ascii code
         * of ‘A’ as it is from the ascii code of ‘a’ */
        c1 = 'A' + (c1 – 'a');
    printf("The letter is %c\n", c1);
    return 0;
}
```
Exercise

- Write a program that:
  - When the user enters a or A, displays “First letter”
  - When the user enters z or Z, displays “last letter”.
  - For any other letter entered by the user it displays “middle letter”.
Switching with char type

```c
char letter;
scanf("%c", &letter);
switch ( letter ) {
    case 'A':
        printf ("First letter \n");
        break;
    case 'Z':
        printf ("Last letter \n");
        break;
    default :
        printf ("Middle letter \n");
}
```
char letter;
scanf("%c", &letter);
switch ( letter ) {
    case 'A':
        printf ("First letter \n");
        break;
    case 'Z':
        printf ("Last letter \n");
        break;
    default :
        printf ("Middle letter \n");
}

Will print this statement for all letters other than A or Z
switch (choice = getchar()) {
    case 'r':
    case 'R': printf("Red");
        break;
    case 'b':
    case 'B': printf("Blue");
        break;
    case 'g':
    case 'G': printf("Green");
        break;
    default: printf("Black");
}

Another Example

switch (choice = getchar()) { 
    case 'r':
    case 'R': printf("Red");
        break;
    case 'b':
    case 'B': printf("Blue");
        break;
    case 'g':
    case 'G': printf("Green");
        break;
    default: printf("Black");
}

Since there isn’t a break statement here, the control passes to the next statement (printf) without checking the next condition.
```c
int main () {
    int operand1, operand2;
    int result = 0;
    char operation ;

    /* Get the input values */
    printf ("Enter operand1 :");
    scanf("%d",&operand1);
    printf ("Enter operation :");
    scanf("\n%c",&operation);
    printf ("Enter operand 2 :");
    scanf("%d", &operand2);
    switch (operation) {
        case '+':
            result=operand1+operand2;
            break;
        case '-':
            result=operand1-operand2;
            break;
        case '*':
            result=operand1*operand2;
            break;
        case '/':
            if (operand2 !=0)
                result=operand1/operand2;
            else
                printf("Divide by 0 error");
            break;
        default:
            printf("Invalid operation\n");
            return;
    }
    printf ("The answer is %d\n",result);
    return 0;
}```
Practice Problems

1. Read in 3 integers and print a message if any one of them is equal to the sum of the other two.

2. Read in the coordinates of two points and print the equation of the line joining them in \( y = mx + c \) form.

3. Read in the coordinates of 3 points in 2-d plane and check if they are collinear. Print a suitable message.

4. Read in the coordinates of a point, and the center and radius of a circle. Check and print if the point is inside or outside the circle.

5. Read in the coefficients \( a, b, c \) of the quadratic equation \( ax^2 + bx + c = 0 \), and print its roots nicely (for imaginary roots, print in \( x + iy \) form)

6. Suppose the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 are mapped to the lowercase letters a, b, c, d, e, f, g, h, i, j respectively. Read in a single digit integer as a character (using \( %c \) in scanf) and print its corresponding lowercase letter. Do this both using switch and without using switch (two programs). Do not use any ascii code value directly.

7. Suppose that you have to print the grades of a student, with \( \geq 90 \) marks getting EX, 80-89 getting A, 70-79 getting B, 60-69 getting C, 50-59 getting D, 35-49 getting P and \( <30 \) getting F. Read in the marks of a student and print his/her grade.