Quiz

Determine the output of the following program:

```cpp
#include <iostream>
using namespace std;

int main()
{
  int x = 3;
  int y = 9;
  int z = 77;
  if (x == (y / 3))
    cout << "H";
  if (z != 77)
    cout << "a";
  if (z == 77)
    cout << "e";
  if (z + y + x < 0)
    cout << "o";
  if (y == (x * x))
    cout << "11";
  cout << "0!" << endl;
} // end main
```
Lecture Contents:

- Loops
  - While
  - do-while
  - for
  - Nesting loops
  - Break & continue

Iterate

- A program loop is a form of iteration. A computer can be instructed to repeat instructions under certain conditions.
While loops

```
while loop:
    if True:
    else:
```

While Loops Syntax

A while STATEMENT WITH A SINGLE STATEMENT BODY
```
while (Boolean_Expression)
    Statement
```

A while STATEMENT WITH A MULTISTATEMENT BODY
```
while (Boolean_Expression)
    [ Statement_1
    Statement_2
    ...
    Statement_Last
]
### while Loop Example

Consider:

```cpp
int count = 0;
cin>>count;      // Initialization

while (count > 0)      // Loop Condition
{
    cout << "Hi ";     // Loop Body
    count--;          // Update expression
}
```

- Loop body executes how many times?
- Use sentinel values for non-determined iterations

### Example: Payroll

Develop a C++ program to determine the gross pay for a number of employees. The company pays “straight time” for the first 40 hours worked by each employee and pays “time-and-a-half” for all hours worked in excess of 40 hours. Your program should input the number of hours each employee worked last week and the hourly rate of each employee. It then should determine and display the employee's gross pay.

**Hint:**
- Use -1 as a sentinel value
do loops

do while:

---

C

Do-while Loop Syntax

A do-while STATEMENT WITH A SINGLE-STATEMENT BODY

do
    Statement
while (Boolean_Expression);

A do-while STATEMENT WITH A MULTISTATEMENT BODY

do
    {
        Statement_1
        Statement_2
        .
        .
        .
        Statement_Last
    } while (Boolean_Expression);

---
**do-while Loop Example**

```cpp
int count = 0;
cin>>cout; // Initialization
do
{
    cout << "Hi "; // Loop Body
    count--; // Update expression
} while (count > 0); // Loop Condition
```

- Loop body executes how many times?
- do-while loops always execute body at least once!

**Example: Cash Register**

Write a C++ program that computes the total amount of money owed by a customer at a cash register. Notice that the customer must pay 7% tax for all the items he buys.
while vs. do-while

- Very similar, but...
  - One important difference
    - Issue is "WHEN" boolean expression is checked
      - while: checks BEFORE body is executed
      - do-while: checked AFTER body is executed
  - After this difference, they're essentially identical!
  - while is more common, due to it's ultimate "flexibility"

for loops

```
for (Init; Bool_Exp; Update)
  Body_Statement
```

```
Initialization
Condition
update
true
false
```
for Loop Example

- for (count=0; count<3; count++)
  {
    cout << "Hi "; // Loop Body
  }

- How many times does loop body execute?

- Initialization, loop condition and update all "built into" the for-loop structure!

- A natural "counting" loop

Example: odd numbers

Write a C++ program to print the odd numbers between 1 and 100.
It’s break time!!  
See you.....

Loops

3 Types of loops in C++

- while
  - Most flexible
  - No "restrictions"

- do-while
  - Least flexible
  - Always executes loop body at least once

- for
  - Natural "counting" loop
Loop Issues

- Loop's condition expression can be ANY boolean expression

Examples:

```c
while (count<3 && done!=0)
{
    // Do something
}
```

```c
for (index=0;index<10 && entry!=-99)
{
    // Do something
}
```

Loop Pitfalls: Misplaced ;

- Watch the misplaced ; (semicolon)

  Example:
  ```c
  while (response != 0) ;
  {
    cout << "Enter val: ";
    cin >> response;
  }
  ```

  Notice the ";" after the while condition!

- Result here: INFINITE LOOP!
Loop Pitfalls: Infinite Loops

- Loop condition must evaluate to false at some iteration through loop
  - If not → infinite loop.
  - Example:
    ```cpp
    while (1)
    {
        cout << "Hello ";
    }
    ```
    - A perfectly legal C++ loop → always infinite!

- Infinite loops can be desirable
  - e.g., "Embedded Systems"

The break and continue

- Flow of Control
  - Recall how loops provide "graceful" and clear flow of control in and out
  - In RARE instances, can alter natural flow
  - `break;`
    - Forces loop to exit immediately.
  - `continue;`
    - Skips rest of loop body
  - These statements violate natural flow
    - Only used when absolutely necessary!
Example: more loops

Write an algorithm that asks the user to input a sequence of integers and do each of the following, separately:
- prints the maximum number of them.
- prints the average of them.
- Counts the odd numbers.
- Counts the even numbers.
- Counts the numbers which are > than 25 and less than 50 or > than 75 and less than 100.

Note: Use a sentinel value to end the sequence and skip an endless loop.

Nested Loops

- Recall: ANY valid C++ statements can be inside body of loop
- This includes additional loop statements!
  - Called "nested loops"
- Requires careful indenting:
  ```cpp
  for (outer=0; outer<5; outer++)
    for (inner=7; inner>2; inner--)
      cout << outer << inner;
  ```
  - Notice no {} since each body is one statement
  - Good style dictates we use {} anyway
Example: Primes

Write a program that finds and prints all of the prime numbers between 3 and 100. A prime number is a number such that one and itself are the only numbers that evenly divide it (e.g., 3, 5, 7, 11, 13, 17, …).

```cpp
for (n=3; n<=100; n++)
{
    // Assume n is prime
    isprime = true;
    // Inner loop tests whether or not n is prime
    for (i=2; i<= n-1; i++)
    {
        // n is not prime if n/i has no remainder
        if ((n % i)==0)
        {
            isprime = false;
        }
    }
    // Check if flag isprime is still true
    if (isprime)
    // Output that n is prime
        cout << n << " is a prime number." << endl;
}
```
Comma Operator

- Evaluate list of expressions, returning value of the last expression
- Most often used in a for-loop
- Example:
  first = (first = 2, second = first + 1);
  - first gets assigned the value 3
  - second gets assigned the value 3
- No guarantee what order expressions will be evaluated.

That’s all for today !!

Thanks.....