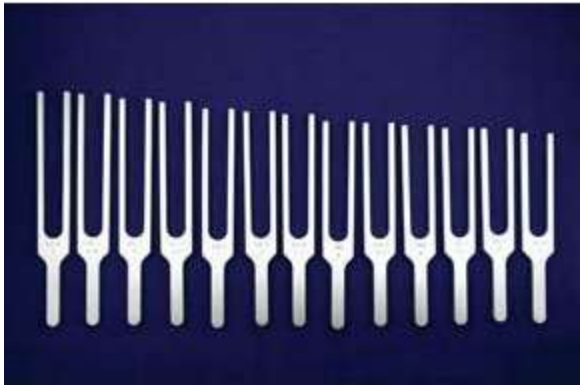
Code Tuning



Deliverables

Logic Constructs

Various techniques to optimize code

Shortcuts in programming

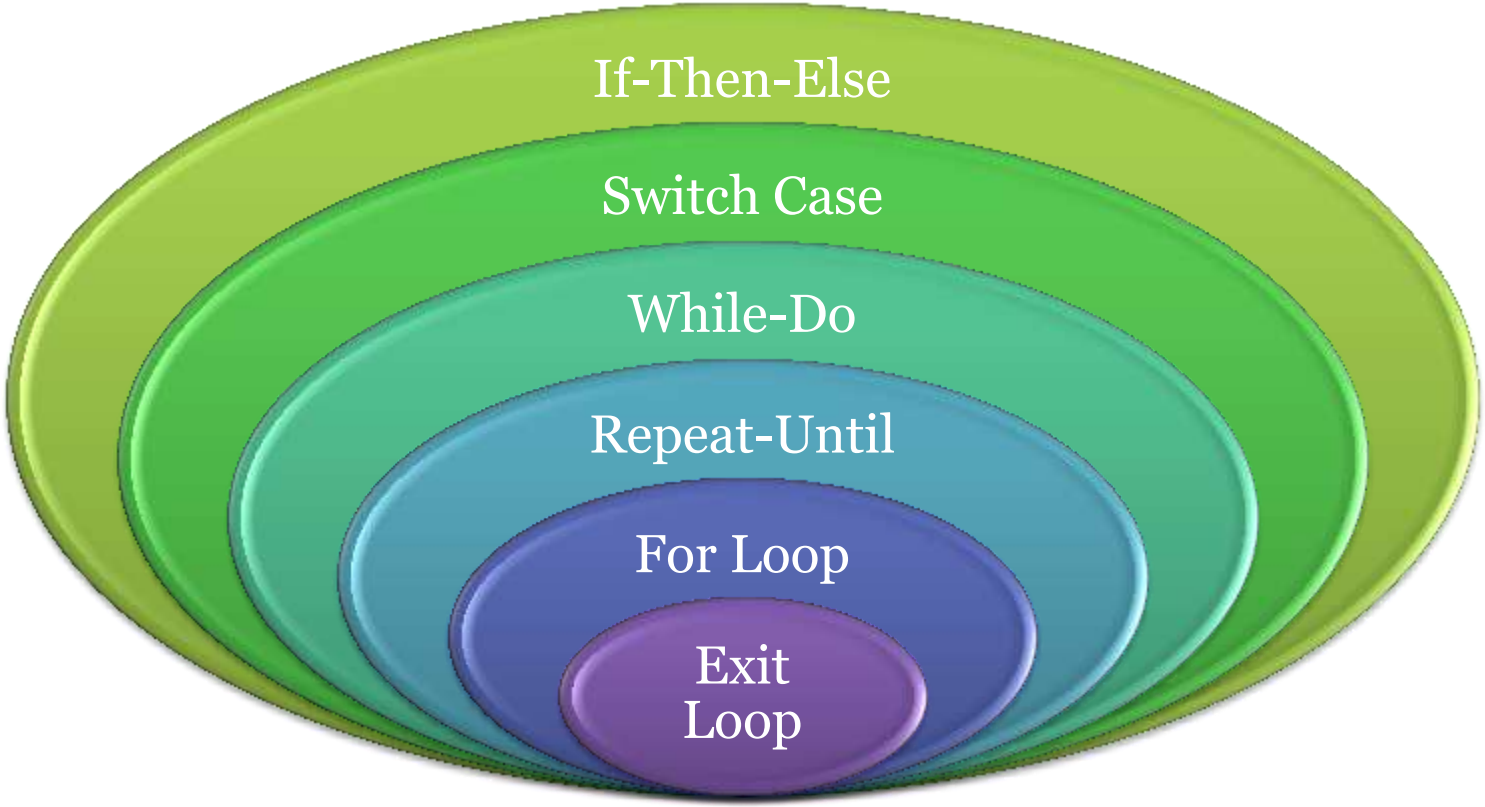
Code Tuning-Introduction

Used to write better code

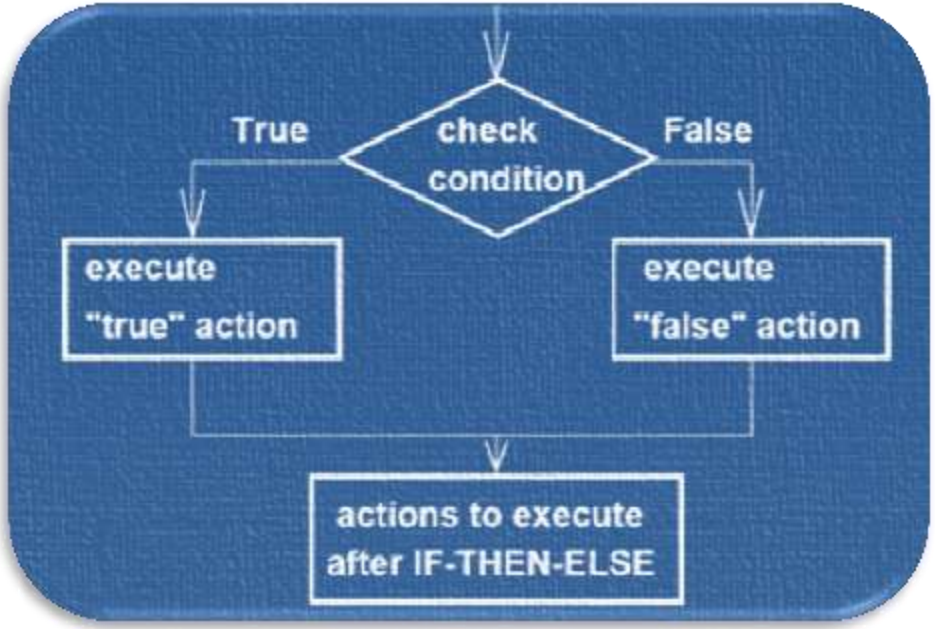
Needs better understanding of the programming language and its compiler

Equivalent of code optimization at higher language level

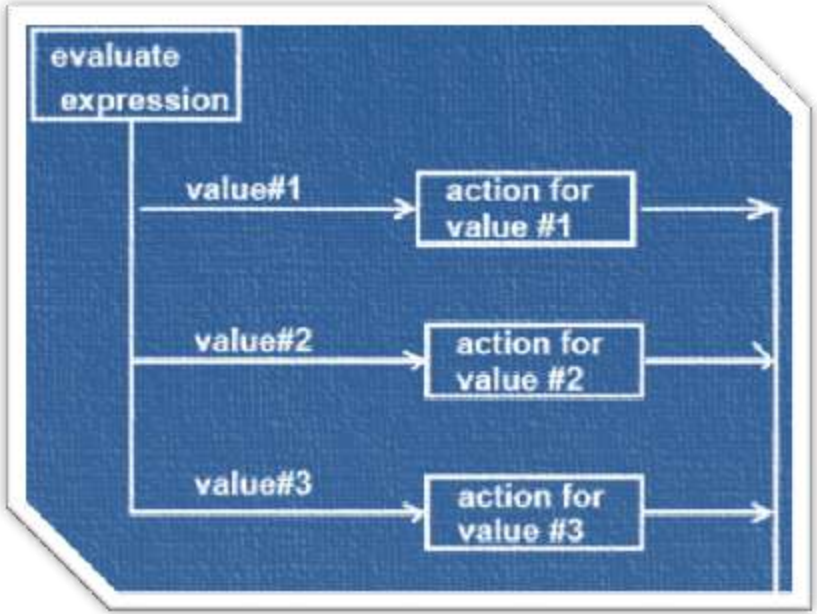
Important Constructs of Logic formulation



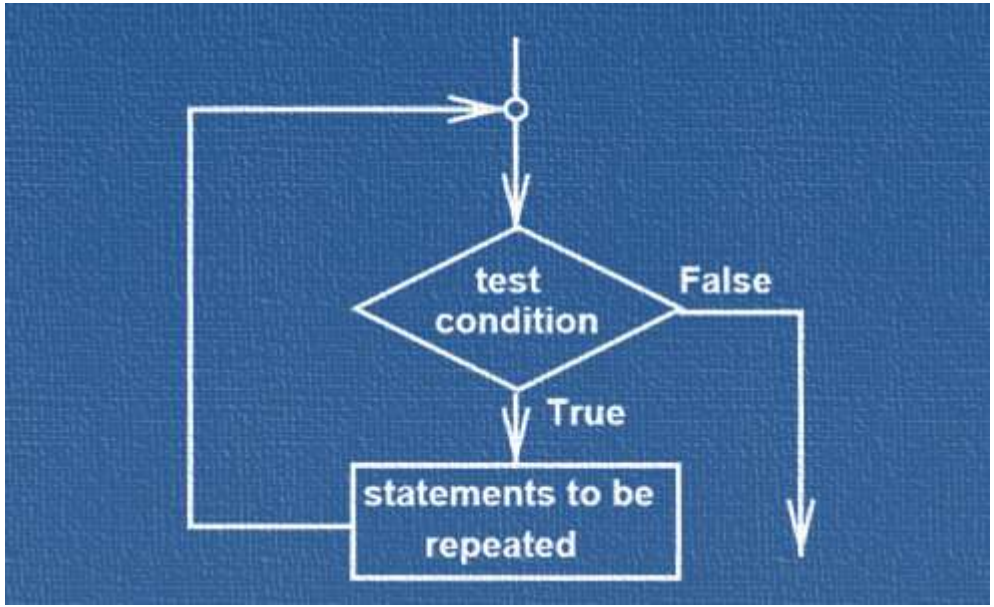
If Then Else



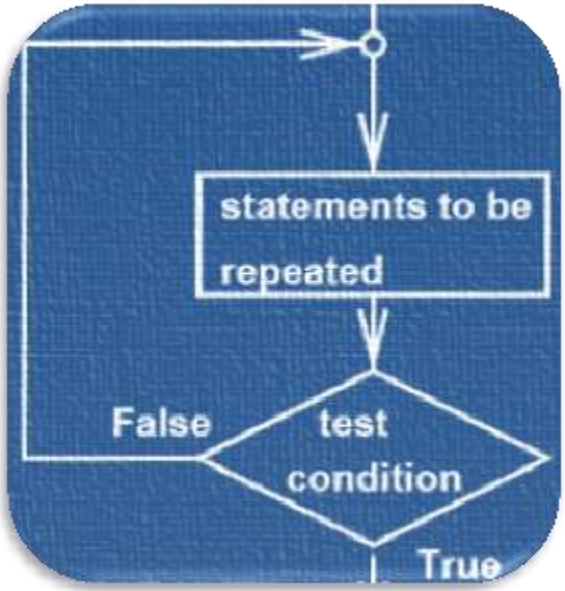
Switch-Case



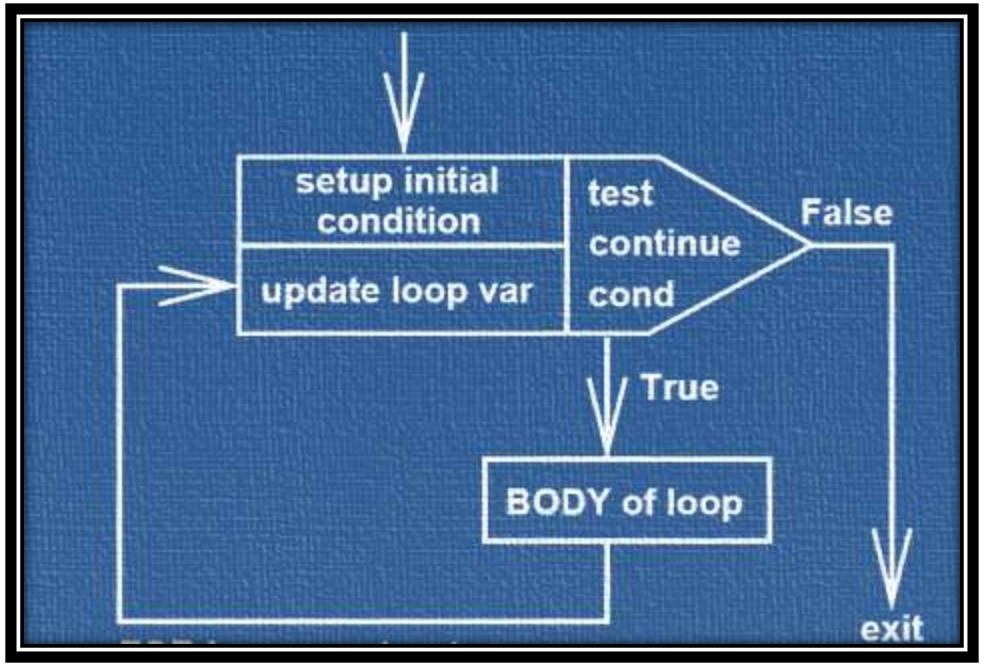
While-Do



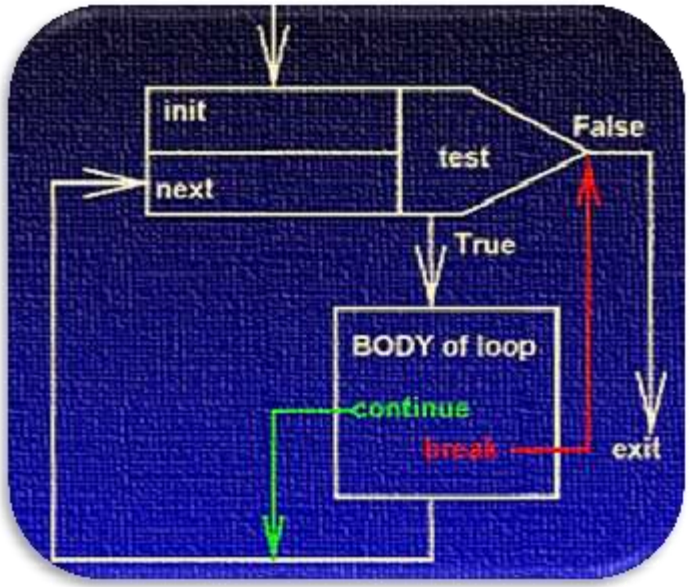
Repeat Until



For Loop



Exit



Proving Correctness of Loops

Loop invariant: A statement that is always true about the loop.

To prove correctness proposition about a loop:

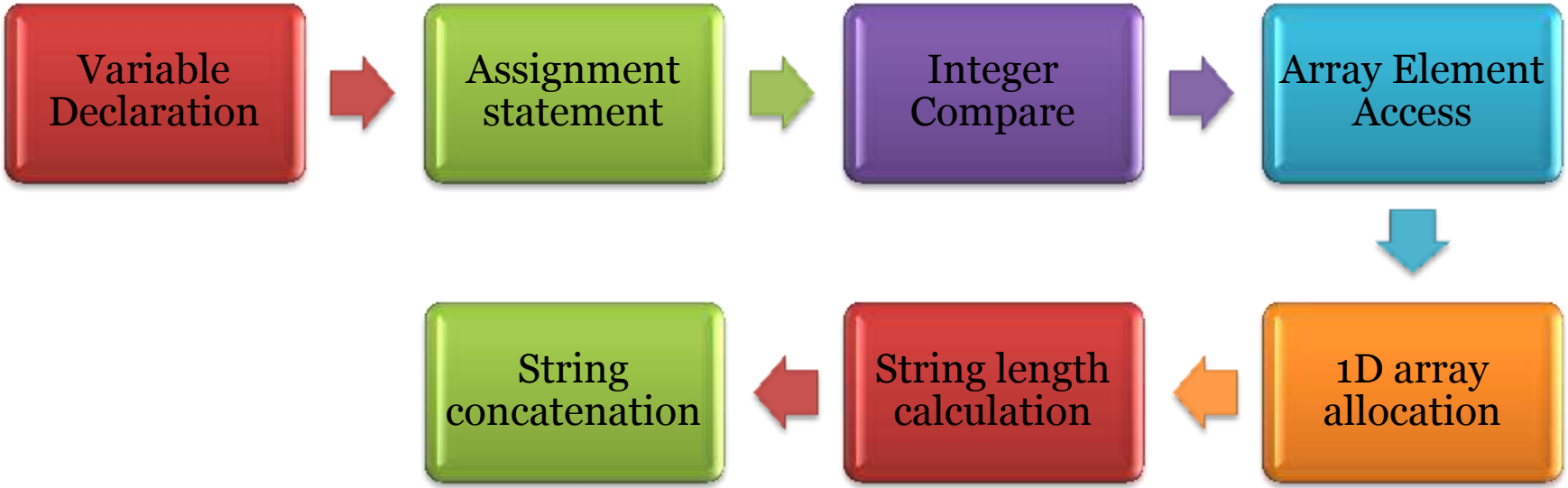
1. Identify loop invariant P.
2. Prove that initialization establishes truth of P.
3. Prove that execution of the loop body preserves the truth of P.
4. Prove that P and falsehood of loop condition, imply post-condition.
5. Prove that loop terminates.

Steps 1-4 ensure partial correctness of loop. 5 ensures total correctness.

Comparative Analysis of various operations

Operation	Expression	Time
Integer add	X+Y	1 ns
Integer Multiply	X*Y	1 ns
Integer Divide	X/Y	2 ns
Floating point add	X+Y	1.5 ns
Floating point Multiply	X*Y	1.5 ns
Floating point Divide	X/Y	6 ns
cos	Math. Cos(theta)	100 ns
…	…	…

Increasing cost of operations



Time calculation

Total Time to run our program will be

$c_1A+c_2B+c_3C+c_4D+c_5E+c_6F...$

Where ci is the cost of a particular operation which will vary from machine to machine and A,B... is the number of times that operation appears in program under analysis.

Space Requirement for common operations

Data type	Bytes
Boolean	1
Byte	1
Char	2
Int	4
Float	4
Long	8
Double	8
1D array Overhead	16 Bytes
Object overhead	8 bytes
Pointer or reference overhead	4 bytes

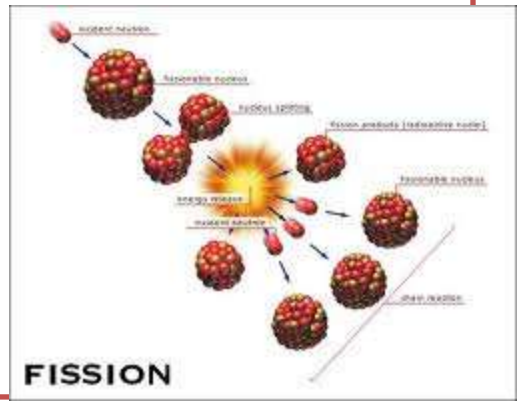
Loop Fusion and Loop Fission

After Loop fission

```
int i, a[100], b[100];
for (i = 0; i < 100; i++)
{
    a[i] = 1;
}
for (i = 0; i < 100; i++)
{
    b[i] = 2;
}
```

After Loop fusion

```
int i, a[100], b[100];
for (i = 0; i < 100; i++)
{
    a[i] = 1;
    b[i] = 2;
}
```



Loop Unrolling/Unwinding

Normal Code

```
for (x = 0; x < 100; x++)
{
print(x);
}
```

Code after unrolling

```
for (x = 0; x < 100; x += 2)
{
print(x);
print(x+1);
}
```

Loop invariant code motion

Normal Code

```
for (i = 0; i < n; ++i)
{
    x = y + z;
    a[i] = 6 * i + x * x;
}
```

Improved code

```
x = y + z;
t1 = x * x;
for (i = 0; i < n; ++i)
{
    a[i] = 6 * i + t1;
}
```

Strength Reduction

Normal Code

```
c = 8;
for (i = 0; i < N; i++)
{
    y[i] = c * i;
}
```

Improved Code

```
c = 8;
k = 0;
for (i = 0; i < N; i++)
{
y[i] = k;
k = k + c;
}
```

Minimize work inside loop

Normal code

```
for (i = 1;i<n/2;i++)
{
...
}
```

Need to compute n/2 times in all iterations

Improved code

```
m = n/2;
for (i = 1;i< m; i++)
{
...
}
```

Loop Peeling

Normal Code

```
int p = 10;
for (int i=0; i<10; ++i)
{
    y[i] = x[i] + x[p];
    p = i;
}
```

Improved Code

```
y[0] = x[0] + x[10];
for (int i=1; i<10; ++i)
{
        y[i] = x[i] + x[i-1];
    }
```

Loop Unswitching

Normal Code

```
int i, w, x[1000], y[1000];
for (i = 0; i < 1000; i++)
{
        x[i] = x[i] + y[i];
        if (w)=1
        y[i] = 0;
}
```

```
int i, w, x[1000], y[1000];
if (w)
{
for (i = 0; i < 1000; i++)
{       x[i] = x[i] + y[i];
        y[i] = 0;
}}
else
{ for (i = 0; i < 1000; i++)
        x[i] = x[i] + y[i];
}
```

Use of sentinel Value

To find the x in array a

```
while(i<n) and (x<>a[i])
{
    i=i+1;
}
if (i<n)
    print number at position i
else
    print number not present
```

Improved Code

```
a[n+1]=x;
while(x<>a[i])
{
    i=i+1;
}
if i==(n+1)
    print number not present
else
    print position of number i
```

Order Condition Testing

Normal Code

```
read(RegnNo)
case Grade(student)
1: {……}
2: {……}
3: {……}
4: {……}
endcase
```

Improved Code

```
read(RegnNo)
case Grade(student)
2: {……}
3: {……}
1: {……}
4: {……}
endcase
```

Applicable for if-else also

Common expression elimination

Normal Code

```
a = b * c + g;
d = b * c * d;
```

Improved Code

```
tmp = b * c;
a = tmp + g;
d = tmp * d;
```

Use constants of correct type

```
float x;
x=5;
```

convert 5 to 5.0 and then stores into x

```
int i;
i=3.14;
```

convert 3.14 to 3 and store into i

Precompute Results

Normal code

```
for(i=0;i<100;i++)
{
    Y=log(i)/log(2)
    B=log(ni^a)/log(2)
}
```

Improved Code

```
twolog=log(2)
for(i=0;i<100;i++)
{
    Y=log(i)/ twolog
    B=log(ni^a)/ twolog
}
```

Dead Code elimination

```
int xyz()
{
    int a = 24;
    int b = 25; /*dead variable
    int c;
    c = a- 2;
    return c;
    b = 24; /* Unreachable
    return 0;
 }
```

Improved Code

```
int xyz()
{
    int a = 24;
    int c;
    c = a- 2;
    return c;
}
```

Lazy computations

Normal code

```
n=x*x+2*y+z
if q>10 then
{
......
}
else q > n then
{.....}
else
{...}
```

Improved code

```
if q>10 then
{......}
else
{
    n=x*x+2*y+z
if q > n then
{.....}
else
{.....}
}
```

Exploit algebraic identities

- Algebraic identities can be used to replace costlier operations by cheaper ones
- To find whether sqrt(x) <sqrt(y), we can use the algebraic identity which says x < y only when x < y. So it is enough to check if x < y in this case.
- not (A or B) is cheaper than not A and not B

Short circuiting and reordering

A particular condition may have the probability of being true or false

```
if (a>b) && (c>d) && (e>f)
{
.....
}
```

Minimizing array references

Normal code

```
for( a=0;a<5;a++)
{
for (b=0;b<10;b++)
{
Total[b]=total[b]*sum[a]
}
}
```

Improved code

```
for( a=0;a<5;a++)
{
Sum_t=sum[a]
for (b=0;b<10;b++)
{
Total[b]=total[b]*sum_t;
}}
```

Locality of Reference

Row major, cache-friendly, use this method to access all items in array sequentially.

```
for (i=0; i<numRow; i++) //
    RowFirst = Effective
for (j=0; j<numCol; j++)
    array[i][j] = 0;
```

Column major, NOT cache-friendly, very poor performance. because computer access array data in row major.

```
for (j=0; j<numCol; j++) //
    ColFirst = Ineffective
for (i=0; i<numRow; i++)
    array[i][j] = 0;
```

Questions and Suggestions and Comments

Question 1

What will be the output of the following program

```
main()
{
int count, i, x;
for(count=1,x=0,i=0; count<=4;count++,i++)
{
x=i%2;
if(x==0)
continue;
else
{
printf("%d\n", x);
continue;
}
}
printf("%d", count);
}
```

Question 2

For calculating the sum of 100 numbers stored in the array we have following code

```
for (int x = 0; x < 100; x++)
{
S=S+a[x];
}
```

Above code can be replaced by one of the following, which will involve less operations

A)

```
x=0;
axvalue=A[x];
for (int x = 0; x < 100; x++)
{
S=S+axvalue;
}
```

B)

```
for (int x = 0; x < 100; x=x+2)
{
S=S+a[x]+(x+1);
}
```

C)

```
for (int x = 0; x < 100; x=x+3)
{
S=S+a[x]+a[x+1]+a[x+2];
}
```

D)

```
for (int x = 0; x < 100; x=x+2)
{
S=S+a[x]+a[x+1];
}
```

Question 3

Which One is a better data structure in context of locality of reference or cache performance.

A) Array

B) Link List

C) Stack

D) Queue