# Outline

# IF STATEMENT

## Indentation is meaningless

Since C++ ignores indentation, multiple statements in an `if` statement must be enclosed in braces to form a block.

## C++ Has No elif

Multiple conditional branches in C++ require nested if statements.

See programs `if1.cpp`, `if2.cpp`, `if3.cpp`, `grades.cpp`

# IF STATEMENT

## Indentation is meaningless

Since C++ ignores indentation, multiple statements in an `if` statement must be enclosed in braces to form a block.

## C++ Has No elif

Multiple conditional branches in C++ require nested if statements.

See programs `if1.cpp`, `if2.cpp`, `if3.cpp`, `grades.cpp`

# Data Types

## Implicit Conversion

Like Python, C++ automatically converts `ints` to `floats` or `doubles` when adding an integer to a float value.

## Explicit Conversion

C++ also performs explicit conversion with the `static_cast` keyword.

See program `typeConvs.cpp`

# Data Types

## Implicit Conversion

Like Python, C++ automatically converts `ints` to `floats` or `doubles` when adding an integer to a float value.

## Explicit Conversion

C++ also performs explicit conversion with the `static_cast` keyword.

See program `typeConvs.cpp`

# Loops

## for Loops Over Different Values of a Single Variable

In Python, a for loop must use a pre-existing sequence of values, like the list returned by the function `range`.

In C++, a variable changes its value by incrementing or decrementing its value for each iteration of the loop.

```
int i;
for (i=0;i<10;i++)
{
   cout << i << endl;
}
```

See a slightly different example: loopExample1.cpp

# Loops

## while Loops

While loops in C++ are exactly like those in Python. They test for a boolean value to be true before they begin each iteration of the loop. When it is false, the loop is exited.

```
int i;
while i < 10
{
   cout << i << endl;
   i += 1; }
```

# Loops

## do-while Loops

Do-while loops in C++ are different than while loops in Python. They only test for the boolean value to be true at the end of each loop iteration. This means that a loop is guaranteed to execute at least once.

```
int i=0;
do {
   cout << i << endl;
   i++;
} while i<10;
```

# Single-Dimension Arrays

## Not As Safe as Python Lists

- No range checking is performed when an index is used.
- Can initialize values when declaring an array.

```
int i,size, a[100];

cout << "Enter size of the array ( < 100 ):";
cin >> size;

for (i=0 ; i<size ; i++) {
   a[i] = i+1;
}
```

# Single-Dimension Arrays

## Not As Safe as Python Lists

- No range checking is performed when an index is used.
- Can initialize values when declaring an array.

```
int i,size = 5, a[] = {1,2,3,4,5};
for (i=0 ; i<size ; i++) {
   cout << a[i] << ", ";
}
```

# Multi-Dimensional Arrays

## Arrays of Arrays

- Any number of dimensions is supported.
- Consecutive addresses are easily found (fast random access via a formula).

```
int i,j,n,m, a[100][100];

cout << "Enter the number of rows ( <100 ):";
   cin >> n;
cout << "Enter the number of columns ( < 100 ):";
   cin >> m;

for(i=0 ; i<n ; i++) {
   for(j=0 ; i<m ; j++) {
      a[i][j] = i+1;
   }
}
```

# Arrays of Characters

## Using Arrays of Characters is Risky

```
char c[20];
cout << "enter your first name:  ";

// this code is a security risk
// a buffer overflow occurs if the user enters
// more than 19 characters
cin >> c;

cout << "Hello " << c << endl;
```

# Arrays of Characters

## C Strings

- `char` arrays with terminating zero in last position.
- Literal string values (in quotes) are zero-terminated strings in C++.
- Convenient library functions for concatenation (`strcat`) etc.
- `#include <string.h>` for library functions

# Arrays of Characters

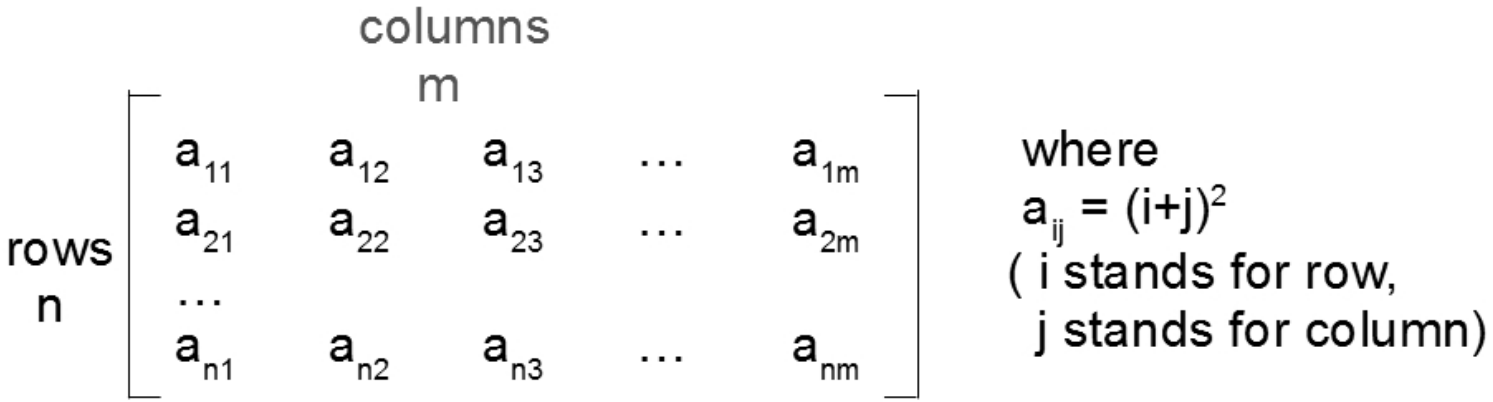
## Better to use the C++ string class.

- `#include <string>`
- An object's capacity increases automatically.
- Convenient member functions for concatenation and other operations.
- Easy to initialize a C++ `string` from a `char` array.

See programs buffer.cpp, str1.cpp

# Arrays of numbers

Build the following matrix of values and display it.




n and m are provided by the user.

# Arrays of numbers

For example, for n=2 and m=3 (2 rows and 3 columns) the matrix will be:

$$A_{2x3} = \begin{bmatrix} 4 & 9 & 16 \\ 9 & 16 & 25 \end{bmatrix}$$