Outline

**Visual Studio**: start project, debug program

**Program Structure**: iostream, std, main, sequential, return, comment

**Integer Variable**: declare, assign, initialize, arithmetic, input/output, round, unsigned, constant.

**Array Variable**: declaration, index, initialization, multi-dimensional array, sparse matrix

**Instruction**: arithmetic/increment, relational/logical, bitwise, if-else, switch, while-loop, for-loop, break

**Other Variables**: float, char, string, enum, vector, structure

**Pointer Variable**: declaration, relation to integer variable, access content, relation to integer array

**Function**: definition, declaration, assignment passing, local/global variable, overload

**Class**: definition, public/private member, constructor/destructor, declaration, memory
1. Float

A float variable can hold real number.

Play with float, similarly to integer.

declare, initialize, array, input/output, arithmetic, if-else, ……

```cpp
int main()
{
    float x=3.14, y[2];
    cin >> y[0];
    y[1] = x * y[0];
    if (y[1] < 5) {
        cout << y[1];
    }
    else {
        cout << y[1] + 1;
    }
    return 0;
}
```

Result remains float.

Check the range of float yourself using <limits>.
Input of a float variable.

```cpp
int main()
{
    float x;
    cin >> x;
    cout << x;
    return 0;
}
```
Output of a float variable.

```cpp
int main()
{
    float x;
    cin >> x;
    cout << x;
    return 0;
}
```

Output the real number.
2. Double

```
int main()
{
    double x = 3.14, y[2];
    cin >> y[0];
    y[1] = x * y[0];
    if (y[1] > 5) {
        cout << y[1];
    } else {
        cout << y[1] + 1;
    }
    return 0;
}
```

Double is similar to float, but covers a larger range. Result remains double. Check the range of double yourself using <limits>.
3. Char

A char variable holds one character.

A character can be anything: letter, number, symbol, etc.
syntax: value in (‘ ’).

```cpp
int main()
{
    char x = 'a', z[2] = {'1', '!'};
    cin >> x;  // Input a character to x.
    cout << z[0] << z[1];  // Output two characters.
    return 0;
}
```

z[0] = ‘1’ and z[1] = ‘!’
Example input of char.

```cpp
int main()
{
    char x = 'a', z[2] = {'1', '!'};
    cin >> x;
    cout << z[0] << z[1];
    return 0;
}
```

Type ‘h’ on keyboard, then press ‘enter’.
Example input of char.

```c
int main()
{
    char x = 'a', z[2] = {'1', '!'};
    cin >> x;
    cout << z[0] << z[1];
    return 0;
}
```

Output two characters, ‘1’ ‘!’
No space between.
Two special characters for output formatting.

```cpp
int main()
{
    char x = 'a', z[2] = {'1','!'};
    cout << z[0] << ' ' << x << '\n' << z[1];
    return 0;
}
```

A space is also a character! This character changes line.
Two special characters for output formatting.

```cpp
int main()
{
    char x = 'a', z[2] = {'1','!'};
    cout << z[0] << ' ' << x << '\n' << z[1];
    return 0;
}
```
int main()
{
    char x;
    cin >> x;
    cout << x;
    return 0;
}

Below is what I typed before pressing ‘enter’.

Take input from keyboard.

Only the first character ‘3’ will be assigned to x.
Only one character to a char variable.

```cpp
int main()
{
    char x;
    cin >> x;
    cout << x;
    return 0;
}
```

Only the first one character is assigned to `x`. Output of `x`.

```
3b#7!k8
3
```
4. String

```cpp
int main()
{
    string x = "I got an A!\n"; // A string is like an "array" of characters, but with a more compact representation.

    cout << x; // syntax: value of a string is in (" ").

    x = "Congratulations!"; // We can assign or reassign a string variable.

    cout << x;

    return 0;
}
```
Output of a string.

```c++
int main()
{
    string x = "I got an A!\n";
    cout << x;
    x = "Congratulations!";
    cout << x;
    return 0;
}
```

Output strings on screen

At the end of the 1st string, there is a change line character `\n`. 
No need to specify size of a string upfront.

```cpp
int main()
{
    string x = "I got an A!\n";
    cout << x;

    x = "Congratulations!";
    cout << x;

    return 0;
}
```

Need not specify size of the string upfront.  
Can assign or reassign a string of any size.

The 1st string has 12 elements. 
The 2nd string has 16 elements.

-- do you know how to count?
Count number of characters in a string.

```java
string x = "I got an A!\n";
```

In total, there are 12 characters in the above string.

A space ' ' or change line '\n' is also a character!
Get size of a string.

```cpp
int main()
{
    string x = "I got an A!\n";
    cout << x.size() << "\n";          // Output 12.

    x = "Congratulations!";
    cout << x.size();                  // Output 16.

    return 0;
}
```

We can use `x.size()` to get the size of `x`. 
Access elements of a string through indexing.

```cpp
int main()
{
    string x = "I got an A!\n";

    cout << x;
    x[9] = 'B';
    cout << x;
    return 0;
}
```

Access an element of string by indexing. x[9] is the 10th element in the string.

Treat an indexed element as a char variable. Assign a new char value ‘B’ to x[9]. (single quote)
Access elements of a string through indexing.

```cpp
int main()
{
    string x = "I got an A!\n";
    cout << x;
    x[9] = 'B';
    cout << x;
    return 0;
}
```
Result of an element update.

```cpp
int main()
{
    string x = "I got an A!\n";
    cout << x;
    x[9] = 'B';
    cout << x;
    return 0;
}
```
Using string to improve readability of outputs.

We often use strings in combination with input and output to improve their readability.

```cpp
int main()
{
    int x;

    cout << "Input an integer: ";

    cin >> x;

    cout << "Your input is " << x;

    return 0;
}
```

This is also a string, just not stored in a variable.

It reminds user he needs to input an integer!

Then take input for x.

Then output x.

This string reminds us what the output is.
Input of one string.

```cpp
int main()
{
    string x;
    cin >> x;
    cout << x;
    return 0;
}
```

Input a sequence of characters and press 'enter'.

Do not include space.
Input of two strings.

```cpp
int main()
{
    string x, y;
    cin >> x >> y;
    cout << "x = " << x << '\n';
    cout << "y = " << y;
    return 0;
}
```

Two input strings should be separated by space.

x will get “data”
y will get “structure”
Output of two strings.

```cpp
int main()
{
    string x, y;
    cin >> x >> y;
    cout << "x = " << x << "\n";
    cout << "y = " << y;
    return 0;
}
```
5. Vector

A vector is similar to array, but more flexible with more pre-defined functions.

```cpp
#include <vector> // include “vector” library

int main()
{
    vector<int> x(3); // a vector of three integers, named x. (initialized by 0’s.)
    x.at(0) = 5; // access the 1st element in the vector.
    cout << x.at(2); // output the 3rd element in the vector.
    return 0;
}
```
Get size of a vector and resize a vector.

```cpp
int main()
{
    vector<int> x(3);
    cout << x.size();  // output 3
    x.resize(10);
    cout << x.size();  // output 10
    return 0;
}
```
Resizing does not change the original elements.

```cpp
int main()
{
    vector<int> x(3);  // Original elements are not affected.
    x.at(2) = 7;       // Initialize new elements to zeros.
    x.resize(6);
    return 0;
}
```
Add a new element to the **end** of a vector; size is automatically increased.

```cpp
int main()
{
    vector<int> x(3);  // 0 0 0
    x.push_back(5);    // 0 0 0 5
    x.push_back(7);    // 0 0 0 5 7
    return 0;
}
```

Size increases automatically.
Copy a vector.

```cpp
int main()
{
    vector<int> x(3), y(3);
    x.at(2) = 7;
    y = x;
    return 0;
}
```


x

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

y

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>
6. Enumeration

```
int main()
{
    enum gender { male, female };
    gender x, y;
    x = male;
    y = female;
    return 0;
}
```

“enum” is used to define a new type of variable and its domain.

The domain should contain a finite number of discrete values.

Values cannot be numbers.
Define a new variable type called “gender”.

This var can take two values: male, female.

```
int main()
{
    enum gender { male, female };
    gender x, y;
    x = male;
    y = female;
    return 0;
}
```

“enum” is used to define a new type of variable and its domain.

The domain should contain a finite number of discrete values.

Values cannot be numbers.
Enumeration: declaration.

int main()
{
  enum gender { male, female };

  gender x, y;

  x = male;
  y = female;

  return 0;
}

Declare two variables x, y of type “gender”.

Similar to declaring x, y as integer variables.

But the type is “gender” instead of “int”, and the domain is self-defined.

“enum” is used to define a new type of variable and its domain.

The domain should contain a finite number of discrete values.

Values cannot be numbers.
Enumeration: assignment.

```
int main()
{
    enum gender { male, female };
    gender x, y;
    x = male;
    y = female;
    return 0;
}
```

“enum” is used to define a new type of variable and its domain.

The domain should contain a finite number of discrete values.

Values cannot be numbers.

Assign “male” to x.

Assign “female” to y.

Similar to assign 5 to x, but value must be self-defined.
Another example of enumeration.

```c
int main()
{
    enum day { Monday, Tuesday, Wednesday, Thursday, Friday };

    day x = Monday;

    day y = Thursday;

    return 0;
}
```
Discussion: why bother with enumeration?

```cpp
int main()
{
    enum day {Monday, Tuesday, Wednesday, Thursday, Friday};

    day x = Monday;

    cout << x;

    return 0;
}
```