Hello World

```cpp
#include <iostream>

int main(){
  std::cout << "Hello World";
  return 0;
}
```

Results: `Hello World` ____ # Newline

```cpp
#include <iostream>

int main(){
  std::cout << "Something" << "\n";
  std::cout << "Else" << std::endl;
  return 0;
}
```

Results:

```
Something
Else
```

- `\n` is great for performance
- `std::endl` is good for flushing out the bufffer ____ # Comments

```cpp
// Single Line Comment
/*
multi
line
comment
*/
```

Variables

```cpp
#include <iostream>

int main(){
  // declaration
  int age;

  // assignment
  age = 19;

  // declaration + assignment
  int something = 20;
```

```
  // truncation
  int days = 6.7;
  std::cout << days << std::endl;
}
```

Results: `6`

```
#include <iostream>

int main(){
  int length = 20;
  double price = 9.99;
  char grade = 'A';
  bool student = true;
  std::string name = "Someone idk";
  return 0;
}
```

Const

```
#include <iostream>

int main(){
  const double PI = 3.14;
  int radius = 20;

  std::cout << "Circumference: " << 2 * PI * radius << std::endl;
  return 0;
}
```

Results: `Circumference: 125.6`

- Makes the variable value read-only. Value cannot be changed after declaration
- Naming Convention : All Caps _____ # Namespaces
- We would get an error if we defined a variable more than once because each entity needs a unique name
- However, we can namespace to define indentically named entities as long as the namespaces are different

```
#include <iostream>

namespace first{
```

```
  int x = 1;
}
namespace second{
  int x = 2;
}

int main(){
  int x = 0;
  std::cout << x << "\n";
  std::cout << first::x << "\n";
  std::cout << second::x << "\n";
  return 0;
}
```

Results:

```
0
1
2
```

- "Using Keyword"

```
#include <iostream>

namespace first{
  int x = 1;
}
namespace second{
  int x = 2;
}

int main(){
  using namespace first;
  std::cout << x << "\n";
  std::cout << first::x << "\n";
  std::cout << second::x << "\n";
  return 0;
}
```

Results:

```
1
1
2
```

- `::` is "Scope Resolution Operator" ____ # Dangerous Territory
- Despicable Stuff (NOT ALLOWED 440V WARNING)

```
#include <iostream>
```

```
int main(){
  using namespace std;
  cout << "Something" << endl;
  return 0;
}
```

Results: `Something`

- A bit better, but why tho?

```
#include <iostream>

int main(){
  using std::cout;
  using std::endl;
  cout << "Something else" << endl;
  return 0;
}
```

Results: `Something else` _____ # Typedef

- Typedef is a reserved keyword used to create additional alias for another data type
- Helps with readability and reduces type
- Naming Convention: <type>_t

```
#include <iostream>

typedef std::string str_t;

int main(){
  str_t name = "someone";
  std::cout << name << std::endl;
  return 0;
}
```

Results: `someone`

- "Using" keyword

```
#include <iostream>

using string_t = std::string;

int main(){

  string_t name = "Aditya";
  std::cout << name << std::endl;
```

```
  return 0;
}
```

Results: `Aditya` ____ # Arithmetic Operators

```
#include <iostream>

int main(){

  int age = 18;
  age += 2;
  std::cout << age << std::endl;
  age ++;
  std::cout << age << std::endl;
  age -= 2;
  std::cout << age << std::endl;
  age --;
  std::cout << age << std::endl;

  age *= 2;
  std::cout << age << std::endl;
  age /= 2;
  std::cout << age << std::endl;

  return 0;
}
```

Results:

```
20
21
19
18
36
18
```

- Integer division will floor the output if its not perfectly divisible

```
#include <iostream>

int main(){

  double num = 33;
  num /= 5;
  std::cout << num << std::endl;

  int num2 = 33;
  num2 /= 5;
  std::cout << num2 << std::endl;
```

```
  return 0;
}
```

Results:

```
6.6
6
```

Type Conversion

- Type Conversion means converting a value of one data type to another

type	info
Implicit	Automatic
Explicit	Precede value with the data type

- Here, it Implicitly converts float to int

```
#include <iostream>

int main(){

  int x = 3.14;
  std::cout << x << std::endl;

  return 0;
}
```

Results: `3`

- Explicitly changing double value to int and then storing it in a double var

```
#include <iostream>

int main(){

  double x = (int) 3.14;
  std::cout << x << std::endl;

  return 0;
}
```

Results: `3`

- Implicitly changed int value to char using ASCII values

```cpp
#include <iostream>

int main(){

  char x = 65;
  std::cout << x << std::endl;

  return 0;
}
```

Results: `A`

- Explicitly changed int to char

```cpp
#include <iostream>

int main(){

  std::cout << (char) 100 << std::endl;

  return 0;
}
```

Results: `d`

- A great scenario in which explicit type conversion is very useful

```cpp
#include <iostream>

int main(){

  int correct = 8;
  int total = 10;

  std::cout << correct / total * 100 << "%" << std::endl;

  return 0;
}
```

Results: `0%`

- Here we are getting 0% as the result instead of 80% because int division is being carried out
- However, if we change any or both of the vars to double, we will get the intended output

```cpp
#include <iostream>

int main(){
```

```
  int correct = 8;
  int total = 10;

  std::cout << correct / (double) total * 100 << "%" << std::endl;
  std::cout << (double) correct / total * 100 << "%" << std::endl;
  std::cout << (double) correct / (double) total * 100 << "%" << std::endl;

  return 0;
}
```

Results:

```
80%
80%
80%
```

Input

```
#include <iostream>

int main(){


  std::string name;

  std::cout << "Type your name: ";
  std::cin >> name;

  return 0;
}
```

- However, this creates a problem when we input multiple words

Getline

```
#include <iostream>

int main(){

  std::string fullname;

  std::cout << "Type your name: ";
  std::getline(std::cin, fullname);

  return 0;
}
```

- This will as well give an error if there is a `std::cin` anywhere before it because `cin` creates a `\n` after input
- To prevent that, we use `std::ws` in getline in order to get rid of any white spaces or `\n` before input

```
#include <iostream>

int main(){

  std::string fullname;

  std::cout << "Type your name: ";
  std::getline(std::cin >> std::ws, fullname);

  return 0;
}
```

Math Functions

Min and Max

```
#include <iostream>

int main(){

  double x = 4;
  double y = 5;

  std::cout << std::min(x,y) << std::endl;
  std::cout << std::max(x,y) << std::endl;

  return 0;
}
```

Results:

```
4
5
```

CMATH

Pow

```
#include <iostream>
#include <cmath>

int main(){
```

```
  std::cout << pow(2,3) << std::endl;

  return 0;
}
```

Results: 8

Sqrt

```
#include <iostream>
#include <cmath>

int main(){

  std::cout << sqrt(16) << std::endl;

  return 0;
}
```

Results: 4

Absolute

```
#include <iostream>
#include <cmath>

int main(){

  std::cout << abs(4.8) << std::endl;

  return 0;
}
```

Results: 4

Round

```
#include <iostream>
#include <cmath>

int main(){

  std::cout << round(4.2) << std::endl;

  return 0;
}
```

Results: 4

Ceil

```
#include <iostream>
#include <cmath>

int main(){

  std::cout << ceil(4.01) << std::endl;

  return 0;
}
```

Results: 5

Floor

```
#include <iostream>
#include <cmath>

int main(){

  std::cout << floor(2.99) << std::endl;

  return 0;
}
```

Results: 2

If else statement

```
#include <iostream>

int main(){

  int age;
  age = 18;

  // std::cout << "Enter your age: ";
  // std::cin >> age;

  if ( age < 18 ){
    std::cout << "You are a CHOILD RIP BOZO" << std::endl;
  }
  else if ( age == 18 ){
    std::cout << "Damnnnn BIGG BOIIII" << std::endl;
  }
  else if ( age == 60 ){
```

```
    std::cout << "DOBBY IS A FREEEE ELFFFFFFFF" << std::endl;
  }
  else{
    std::cout << "bro damn" << std::endl;
  }

  return 0;
}
```

Results: Damnnnn BIGG BOIIII

Switch Case

```
#include <iostream>

int main(){

  int month;
  month = 2;

  // std::cout << "Enter month: (in number) ";
  // std::cin >> month;

  switch(month){
    case 1:
      std::cout << "January" << std::endl;
      break;
    case 2:
      std::cout << "February" << std::endl;
      break;
    case 3:
      std::cout << "March" << std::endl;
      break;
    case 4:
      std::cout << "April" << std::endl;
      break;
    case 5:
      std::cout << "May" << std::endl;
      break;
    case 6:
      std::cout << "June" << std::endl;
      break;
    case 7:
      std::cout << "July" << std::endl;
      break;
```

```
    case 8:
      std::cout << "August" << std::endl;
      break;
    case 9:
      std::cout << "September" << std::endl;
      break;
    case 10:
      std::cout << "October" << std::endl;
      break;
    case 11:
      std::cout << "November" << std::endl;
      break;
    case 12:
      std::cout << "December" << std::endl;
      break;
    default:
      std::cout << "Enter a number from 1-12" << std::endl;
  }

  return 0;
}
```

Results: `February`

Ternary Operator

- Replacement to if-else statement
- Condition ? expression 1 : expression 2

```
#include <iostream>

int main(){

  int grade = 75;

  grade < 60 ? std::cout << "Damn bro, you got a back" << "\n" : std::cout << "Nice" << "\n'

  return 0;
}
```

Results: `Nice`

```
#include <iostream>

int main(){

  bool student = true;
```

```
  std::cout << ( student ? "Padh le bhai" : "rehne de bro" );

return 0;
}
```

Results: `Padh le bhai`

Logical Operators

Operator	Symbol
Add	&&
Or	||
Not	!

String methods

- length
- empty
- clear
- apppend
- at
- insert
- find
- erase

```
#include <iostream>
#include <string>

int main(){

  std::string a = "something";
  std::string empt;
  std::string mail = "adityascottish27gmail.com";

  std::cout << a.length() << std::endl;
  std::cout << a.empty() << std::endl;
  std::cout << empt.empty() << std::endl;

  a.clear();
  std::cout << a.empty() << std::endl;

  std::string name = "Aditya";
  name.append(" Gautam");
  std::cout << name << std::endl;
```

```
  std::cout << name.at(7) << std::endl;
  std::cout << name[7] << std::endl;

  mail.insert(16,"@");
  std::cout << mail << std::endl;

  std::cout << name.find("a") << std::endl;

  std::cout << name << std::endl;
  name.erase(5,5);
  std::cout << name << std::endl;

  return 0;
}
```

Results:

```
9
0
1
1
Aditya Gautam
G
G
adityascottish27@gmail.com
5
Aditya Gautam
Aditytam
```

- ***NOTE***: <string>.erase(x,y) will erase `y` characters from `x` index, including the `x`'th character

While Loop

```
#include <iostream>

int main(){

  int count = 0;

  while ( count < 10 ){
    count ++;
    std::cout << count << std::endl;
  }
```

```
  return 0;
}
```

Results:

```
1
2
3
4
5
6
7
8
9
10
```

For Each Loop

```
#include <iostream>

int main(){

  char grades[] = { 'A', 'B', 'C', 'D' };

  for ( char eachGrade : grades ){
    std::cout << eachGrade << std::endl;
  }

  return 0;
}
```

Results:

```
A
B
C
D
```

Pass Array to Function

- When passing an array to a function as an argument, its shrinked / decays to a pointer and therefore its size cannot be determined inside the function. In order to combat that, pass size along with the array as an argument

Sorting an array

- Bubble Sort

```
#include <iostream>

int main(){

  int arr[] = { 10, 1, 9, 2, 8, 3, 7, 4, 6, 5 };

  for ( int i = 0; i < sizeof(arr)/sizeof(arr[0]); i++ ){
    int j = i + 1;
    while ( j < sizeof(arr)/sizeof(arr[0]) ){
      if ( arr[i] > arr[j] ){
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
      }
      j++;
    }
  }

  for ( int something : arr ){
    std::cout << something << std::endl;
  }

  return 0;
}
```

Results:

```
1
2
3
4
5
6
7
8
9
10
```

Fill function

- `fill()` = Fills a range of elements with a specified value
- fill(begin, end, value)

```cpp
#include <iostream>

int main(){

  const int SIZE = 10;
  std::string something[SIZE];

  fill( something, something + SIZE, "something" );

  for( std::string each : something ){
    std::cout << each << std::endl;
  }

  return 0;
}
```

Results:

```
something
something
something
something
something
something
something
something
something
something
```

- Eg: The first half should have “one”, second half should have “two” and third half should have “three”

```cpp
#include <iostream>

int main(){

  const int SIZE = 12;

  std::string arr[SIZE];

  fill(arr, arr + (SIZE/3), "one");
  fill(arr + (SIZE/3), arr + (SIZE/3)*2, "two");
  fill(arr + (SIZE/3)*2, arr + SIZE, "three");

  for ( std::string something : arr ){
    std::cout << something << std::endl;
  }
```

```
  return 0;
}
```

Results:

```
one
one
one
one
two
two
two
two
three
three
three
three
```

Memory Address

```
#include <iostream>

int main(){

  std::string name = "something";
  int age = 18;
  bool student = true;

  std::cout << &name << "\n" << &age << "\n" << &student << std::endl;

  return 0;
}
```

Results:

```
0x7ffd6e074e98
0x7ffd6e074e8c
0x7ffd6e074e8b
```

Pass by value VS Pass by reference

```
#include <iostream>

void swap(int x, int y);

int main(){
```

```
  int x = 1;
  int y = 2;

  swap(x,y);

  std::cout << x << std::endl;
  std::cout << y << std::endl;

  return 0;
}

void swap(int x, int y){
  int temp;

  temp = x;
  x = y;
  y = temp;
}
```

Results:

```
1
2
```

- This does not work as intended because we pass x and y as values and hence, the x and y inside the function are new (different) x and y
- We could also consider scope in this
- To combat this, we pass variables by reference to their memory addresses

```
#include <iostream>

void swap(int &x, int &y);

int main(){

  int x = 1;
  int y = 2;

  swap(x,y);

  std::cout << x << std::endl;
  std::cout << y << std::endl;

  return 0;
}

void swap(int &x, int &y){
```

```
  int temp;

  temp = x;
  x = y;
  y = temp;
}
```

Results:

```
2
1
```

Check Credit Card Number Program

```
#include <iostream>
#include <string>

void intToArr( long int num, int arr[] );

int main(){

  // long int cardNum;
  // std::cout << "Enter the credit card number: ";
  // std::cin >> cardNum;

  long int cardNum = 6011000990139424;

  int size = std::to_string(cardNum).length();
  int arr[size];
  std::string evenSide;
  std::string oddSide;
  int evenCount = 0;
  int oddCount = 0;

  intToArr( cardNum, arr );

  for ( int i = 0; i < size; i += 2 ){
    evenSide += std::to_string(2*arr[i]);
  }
  for ( int i = 1; i < size; i += 2 ){
    oddSide += std::to_string(arr[i]);
  }

  for ( char num : evenSide ){
    evenCount += (int)num - '0';
  }
```

```
  for ( char num : oddSide ){
    oddCount += (int)num - '0';
  }

  if ( (oddCount+evenCount) % 10 == 0 ){
    std::cout << "Valid Credit Card Number!" << std::endl;
  }
  else{
    std::cout << "Invalid Number! Try Again!" << std::endl;
  }

  return 0;
}

void intToArr( long int num, int arr[] ){
  std::string numStr = std::to_string(num);
  int size = numStr.length();
  for ( int i = 0; i < size; i++ ){
    arr[i] = numStr[i] - '0';
  }
}
```

Results: `Valid Credit Card Number!`

- Some common problems I faced while coding this program so that I do not make those again:
  - In the `intToArr` function, I passed the num argument as an integer instead of a long int even though the credit card number is longer than a normal integer, so it would truncate to 9 digits when passed through the function and hence my calculations were getting wrong because the rest 7 digits were automatically getting stored with random numbers
  - This code is still not organized so, some tips :
    - Think about the problem before starting to write the code
    - Then break the program into functions for repititive tasks and in order to make the code readable
    - Add comments in order to increase readability
  - Be aware about args which need to be passed by values and those need to be passed by reference
  - Pass only those args as reference whose values needs to be changed

Pointers

- Variable that stores a memory address of another variable

Operator	Info
&	address-of operator
*	dereference operator

```
#include <iostream>

int main(){

    std::string name = "Something";
    std::string *pName = &name;

    std::cout << pName << std::endl;
    std::cout << *pName << std::endl;

    return 0;
}
```

Results:

```
0x7ffdeb042198
Something
```

Pointer of an Array

```
#include <iostream>

int main(){

  std::string something[] = { "one", "two", "three" };
  // std::string *pSomething = &something;
  std::string *pSomething = something;

  std::cout << *pSomething << std::endl;

  return 0;
}
```

Results:

```
one
one
```

- If we used the commented line like we used for ints and other data types, it would give an error because the array something is already an address
- And if we try to print the value referenced through that pointer, it will point to the first element

NULL

- A special value that means something has no value
- When a pointer is holding a null value, that pointer is not pointing at anything (null pointer)
- nullptr = keyword representing a null pointer literal
- nullptr are helpful when determining if an address was successfully assigned to a pointer

```
#include <iostream>

int main(){

  int *pointer = nullptr;
  int x = 123;

  pointer = &x;

  if(pointer == nullptr){
    std::cout << "Did not successfully assigned the address of x to the pointer!" << std::en
  }
  else{
    std::cout << "Address of x was assigned to the pointer" << std::endl;
  }

  return 0;
}
```

Results: `Address of x was assigned to the pointer`

- NOTE: Do not dereference a null pointer or pointing to free memory because that will cause undefined behavior

Tic Tac Toe Game

```
#include <iostream>
#include <ctime>

// Function Declarations
void drawBoard( char *spaces );
void playerMove( char *spaces, char player );
void computerMove( char *spaces, char computer );
bool checkWin( char *spaces, char player, char computer, bool &gameStatus );
int main(){

    char spaces[9] = {' ', ' ', ' ',
                                    ' ', ' ', ' ',
```

```
                                    ' ', ' ', ' '};
    char player = 'X';
    char computer = 'O';
    bool gameStatus = true;

    drawBoard( spaces );
    while ( gameStatus ){
        playerMove( spaces, player );
        drawBoard( spaces );
        if (checkWin( spaces, player, computer, gameStatus )){
            gameStatus = false;
            break;
        };
        computerMove( spaces, computer );
        drawBoard( spaces );
        if (checkWin( spaces, player, computer, gameStatus )){
            gameStatus = false;
            break;
        };
    }

    return 0;
}
void drawBoard( char *spaces ){
    std::cout << "\n" << "     " << "|" << "     " << "|" << "     " << "\n";
    std::cout << "  " << spaces[0] << "  |  " << spaces[1] << "  |  " << spaces[2] << "  ";
    std::cout << "\n" << "     " << "|" << "     " << "|" << "     " << "\n";
    std::cout << "-----" << "|" << "-----" << "|" << "-----";
    std::cout << "\n" << "     " << "|" << "     " << "|" << "     " << "\n";
    std::cout << "  " << spaces[3] << "  |  " << spaces[4] << "  |  " << spaces[5] << "  ";
    std::cout << "\n" << "     " << "|" << "     " << "|" << "     " << "\n";
    std::cout << "-----" << "|" << "-----" << "|" << "-----";
    std::cout << "\n" << "     " << "|" << "     " << "|" << "     " << "\n";
    std::cout << "  " << spaces[6] << "  |  " << spaces[7] << "  |  " << spaces[8] << "  ";
    std::cout << "\n" << "     " << "|" << "     " << "|" << "     " << "\n" << std::endl;
}
void playerMove( char *spaces, char player ){
    int number;

    while ( true ){
        std::cout << "Enter the grid where you want to place your marker: (1-9) ";
        std::cin >> number;
        number--;
        if ( number < 0 || number > 8 ){
            std::cout << "Enter a number in range 1-9. Try Again!" << std::endl;
        }
```

```
        else if ( spaces[number] == ' ' ){
            spaces[number] = player;
            break;
        }
        else{
            std::cout << "Grid already filled! Try Again!" << std::endl;
        }
    }
}
void computerMove( char *spaces, char computer ){
    std::cout << "Computer's Turn..." << std::endl;
    while ( true ){
        srand(time(0));
        int num = rand()%9;
        if ( spaces[0]!=' ' && spaces[1]!=' ' && spaces[2]!=' ' && spaces[3]!=' ' && spaces
            break;
        }
        else if ( spaces[num] == ' ' ){
            spaces[num] = computer;
            break;
        }
    }
}
bool checkWin( char *spaces, char player, char computer, bool &gameStatus ){

    // Rows
    if ( spaces[0] != ' ' && spaces[0] == spaces[1] && spaces[1] == spaces[2] ){
        std::cout << ( spaces[0] == player ? "You WIN!" : "You LOSE!" ) << std::endl;
    }
    else if ( spaces[3] != ' ' && spaces[3] == spaces[4] && spaces[4] == spaces[5] ){
        std::cout << ( spaces[3] == player ? "You WIN!" : "You LOSE!" ) << std::endl;
    }
    else if ( spaces[6] != ' ' && spaces[6] == spaces[7] && spaces[7] == spaces[8] ){
        std::cout << ( spaces[6] == player ? "You WIN!" : "You LOSE!" ) << std::endl;
    }

    // Columns
    else if ( spaces[0] != ' ' && spaces[0] == spaces[3] && spaces[3] == spaces[6] ){
        std::cout << ( spaces[0] == player ? "You WIN!" : "You LOSE!" ) << std::endl;
    }
    else if ( spaces[1] != ' ' && spaces[1] == spaces[4] && spaces[4] == spaces[7] ){
        std::cout << ( spaces[1] == player ? "You WIN!" : "You LOSE!" ) << std::endl;
    }
    else if ( spaces[2] != ' ' && spaces[2] == spaces[5] && spaces[5] == spaces[8] ){
        std::cout << ( spaces[2] == player ? "You WIN!" : "You LOSE!" ) << std::endl;
    }
```

```
    // Diagonals
    else if ( spaces[0] != ' ' && spaces[0] == spaces[4] && spaces[4] == spaces[8] ){
        std::cout << ( spaces[0] == player ? "You WIN!" : "You LOSE!" ) << std::endl;
    }
    else if ( spaces[2] != ' ' && spaces[2] == spaces[4] && spaces[4] == spaces[6] ){
        std::cout << ( spaces[2] == player ? "You WIN!" : "You LOSE!" ) << std::endl;
    }

    // Checking for a tie
    else if ( spaces[0]!=' ' && spaces[1]!=' ' && spaces[2]!=' ' && spaces[3]!=' ' && spaces
        std::cout << "TIE!" << std::endl;
    }
    else{
        return false;
    }
    return true;
}
```

Dynamic Memory

- Memory that is allocated after the program is already compiled and running
- Use the `new` operator to allocate memory in the heap rather than the stack
- You must `delete` the pointer after using the `new` operator in order to free up memory at that address and prevent memory leak

```
#include <iostream>

int main(){

  int *pNum = nullptr;
  pNum = new int;

  *pNum = 21;

  std::cout << "Address: " << pNum << std::endl;
  std::cout << "Value: " << *pNum << std::endl;

  return 0;
}
```

Results:

```
Address: 0x5d6bccf2d2b0
Value: 21
```

Dynamically inputting elements into array

```
#include <iostream>

int main(){

  char *pGrades = nullptr;

  int size;
  std::cout << "How many grades to enter?: ";
  std::cin >> size;

  pGrades = new char[size];

  for ( int i = 0; i < size; i++ ){
    std::cout << "Enter element #" << i+1 << ": ";
    std::cin >> pGrades + i;
    /* std::cin >> pGrades[i]; */
  }
  for ( int i = 0; i < size; i++ ){
    std::cout << pGrades[i] << std::endl;
  }
  // std::cout << pGrades << std::endl;

  delete[] pGrades;

  return 0;
}
```

- These statements are same

```
std::cin >> pGrades + i;
std::cin >> pGrades[i];
```

Function Template

- Describes what a function looks like
- Can be used to generate as many overloaded functions as needed, each using different data types
- In order to make a function (lets say, max) for different data types (like, int, long int, float, char, etc), we would have to overload the function i.e. define multiple same max functions but with args of different data types

```
#include <iostream>

int max( int a, int b ){
```

```
  return ( a > b ? a : b );
}
double max( double a, double b ){
  return ( a > b ? a : b );
}
char max( char a, char b ){
  return ( a > b ? a : b );
}

int main(){

  std::cout << max(1,2) << std::endl;
  std::cout << max(1.1,2.2) << std::endl;
  std::cout << max('1','2') << std::endl;

  return 0;
}
```

Results:

```
2
2.2
2
```

- In order to combat this, we use function template

```
#include <iostream>

template <typename T>

T max( T x, T y ){
  return ( x > y ? x : y );
}

int main(){

  std::cout << max(1,2) << std::endl;
  std::cout << max(1.1,2.2) << std::endl;
  std::cout << max('1','2') << std::endl;

  return 0;
}
```

Results:

```
2
2.2
2
```

- Now, what if you need to mix and match the data types of the two variables?
- We use multiple templates then

```
#include <iostream>

template <typename T, typename U>

auto max( T x, U y ){
  return ( x > y ? x : y );
}

int main(){

  std::cout << max(2.2,1) << std::endl;

  return 0;
}
```

Results: `2`

- Note: We use `auto max()` and not `T max()` or `U max()` so that it can automatically deduce which data type to return judging from the arguments
- Advantage: We only have to code the function once and it will be compatible with many data types

Struct

- A structure that group related variables under one name
- Structs can contain many different data types
- Variables in a struct are known as `members`
- Members can be accessed with `.` i.e. `Class Member Access Operator`
- Structs are like (just) attributes of classes in python

```
#include <iostream>

struct student{
  std::string name;
  double gpa;
  bool enrolled = true;
};

int main(){

  student student1;
  student1.name = "Someone";
  student1.gpa = 8.0;
```

```
  student1.enrolled = true;

  std::cout << student1.name << std::endl;
  std::cout << student1.gpa << std::endl;
  std::cout << student1.enrolled << std::endl;

  student student2;
  student2.name = "Else";
  student2.gpa = 4.2;

  std::cout << student2.name << std::endl;
  std::cout << student2.gpa << std::endl;
  std::cout << student2.enrolled << std::endl;

  return 0;
}
```

Results:

```
Someone
8
1
Else
4.2
1
```

- Note: Make sure to put a semicolon after defining a struct
- Withing members, we can set a default value
- i.e. enrolled here

Pass a Struct to a function

```
#include <iostream>

struct Car{
  std::string model;
  int year;
  std::string color;
};
void printCar(Car car);
void paintCar(Car &car, std::string color);

int main(){

  Car car1;
  Car car2;
```

```
  car1.model = "Mustang";
  car1.year = 2023;
  car1.color = "Red";

  car2.model = "Corvette";
  car2.year = 1997;
  car2.color = "Blue";

  printCar(car1);
  printCar(car2);

  paintCar(car1, "Green");
  printCar(car1);

  return 0;
}

void printCar(Car car){
  std::cout << car.model << std::endl;
  std::cout << car.year << std::endl;
  std::cout << car.color << std::endl;
  std::cout << std::endl;
}
void paintCar(Car &car, std::string color){
  car.color = color;
}
```

Results:

```
Mustang
2023
Red

Corvette
1997
Blue

Mustang
2023
Green
```

- For passing Structs to a function, they are passed by value and not by reference by default. I.e. inside the function, it will create a copy of that struct and use that as the argument
- If we need to pass by reference, we can do as usual by passing the address of the struct as the argument

- `void printCar(Car &car);`

Enums

- Enumerations
- A user defined data type that consists of paired named-integer constants
- Great if you have a set of potential options
- Normally, we cannot use strings in switch but we can use enums

```
#include <iostream>

enum Day {
  sunday = 0,
  monday = 1,
  tuesday = 2,
  wednesday = 3,
  thursday = 4,
  friday = 5,
  saturday = 6
};

int main(){

  Day today = wednesday;

  switch(today){
    case sunday:
      std::cout << "It is day number " << today << " of the week" << std::endl;
      break;
    case monday:
      std::cout << "It is day number " << today << " of the week" << std::endl;
      break;
    case tuesday:
      std::cout << "It is day number " << today << " of the week" << std::endl;
      break;
    case wednesday:
      std::cout << "It is day number " << today << " of the week" << std::endl;
      break;
    case thursday:
      std::cout << "It is day number " << today << " of the week" << std::endl;
      break;
    case friday:
      std::cout << "It is day number " << today << " of the week" << std::endl;
      break;
    case saturday:
```

```
      std::cout << "It is day number " << today << " of the week" << std::endl;
      break;
  }
  std::cout << friday << std::endl;

  return 0;
}
```

Results:

```
It is day number 3 of the week
5
```

- Note: We can also do `case 0:` instead of `case sunday:` but the latter is more readable
- The cool thing is if we do not explicitly assign value to enum members, then it automatically assigns int value starting from 0
- eg: `enum Flavor { vanilla, chocolate, strawberry, mint };` will have vanilla at 0, chocolate at 1, etc.

Object

- A collection of attributes and methods
- They can have characteristics and could perform actions
- Created from a class which acts as a "blueprint"
- Something