Printing

```
cout << "Hello" << endl;
cout << "World";
cout << "!" << endl;
```

Variables and Data Types

```
/*
Names are case-sensitive and may begin with:
    letters, _
After, may include
    letters, numbers, _
Convention says
    Start with a lowercase word, then additional words are capitalized
    ex. myFirstVariable
*/
string name = "Mike";    // string of characters, not primitive
char testGrade = 'A';    // single 8-bit character.

// you can make them unsigned by adding "unsigned" prefix
short age0 = 10;        // atleast 16-bits signed integer
int age1 = 20;          // atleast 16-bits signed integer (not smaller than short)
long age2 = 30;         // atleast 32-bits signed integer
long long age3 = 40;    // atleast 64-bits signed integer

float gpa0 = 2.5f;      // single percision floating point
double gpa1 = 3.5;      // double-precision floating point
long double gpa2 = 3.5; // extended-precision floating point

bool isTall;            // 1 bit -> true/false
isTall = true;

name = "John";

cout << "Your name is " << name << endl;
```

Casting and Converting

```
cout <<  (int)3.14 << endl;
cout << (double)3 / 2 << endl;
```

Pointers

```
int num = 10;
cout << &num << endl;

int *pNum = &num;
cout << pNum << endl;
cout << *pNum << endl;
```

Strings

```
#include <string>
string greeting = "Hello";
//      indexes:   01234

cout << greeting.length();
cout << greeting[0] << endl;
cout << greeting.find("llo") << endl;
cout << greeting.substr(2) << endl;
cout << greeting.substr(1, 3) << endl;
```

Numbers

```
cout << 2 * 3 << endl;      // Basic Arithmetic: +, -, /, *
cout << 10 % 3 << endl;     // Modulus Op. : returns remainder of 10/3
cout << 1 + 2 * 3 << endl;  // order of operations
cout << 10 / 3.0 << endl;   // int's and doubles


int num = 10;
num += 100; // +=, -=, /=, *=
cout << num << endl;

num++;
cout << num << endl;

// Math class has useful math methods
#import <cmath>
cout << pow(2, 3) << endl;
cout << sqrt(144) << endl;
cout << round(2.7) << endl;
```

User Input

```
string name;
cout << "Enter your name: ";
cin >> name;
cout << "Hello " << name << endl;

int num1, num2;
cout << "Enter first num: ";
cin >> num1;
cout << "Enter second num: ";
cin >> num2;
cout << "Answer: " << num1 + num2 << endl;
```

Arrays

```
// int luckyNumbers[6];
int luckyNumbers[] = {4, 8, 15, 16, 23, 42};
//       indexes:   0  1  2   3   4   5
luckyNumbers[0] = 90;
cout << luckyNumbers[0] << endl;
cout << luckyNumbers[1] << endl;
```

2 Dimensional Arrays

```
// int  numberGrid[2][3];
int numberGrid[2][3] = { {1, 2, 3}, {4, 5, 6} };
numberGrid[0][1] = 99;

cout << numberGrid[0][0] << endl;
cout << numberGrid[0][1] << endl;
```

Vectors

```
// #include <vector>
vector<string> friends;
friends.push_back("Oscar");
friends.push_back("Angela");
friends.push_back("Kevin");
friends.insert(friends.begin() + 1, "Jim");

// friends.erase(friends.begin() + 1);
cout << friends.at(0) << endl;
cout << friends.at(1) << endl;
cout << friends.at(2) << endl;
cout << friends.size() << endl;
```

Functions

```
int addNumbers(int num1, int num2);

int main()
{
    int sum = addNumbers(4, 60);
    cout << sum << endl;

    return 0;
}

int addNumbers(int num1, int num2){
    return num1 + num2;
}
```

If Statements

```
bool isStudent = false;
bool isSmart = false;

if(isStudent && isSmart){
    cout << "You are a student" << endl;
} else if(isStudent && !isSmart){
    cout << "You are not a smart student" << endl;
} else {
    cout << "You are not a student and not smart" << endl;
}

// >, <, >=, <=, !=, ==
```

```
if(1 > 3){
    cout << "number omparison was true" << endl;
}

if('a' > 'b'){
    cout << "character comparison was true" << endl;
}

string myString = "cat";
if(myString.compare("cat") != 0){
    cout << "string comparison was true" << endl;
}
```

Switch Statements

```
char myGrade = 'A';
switch(myGrade){
    case 'A':
        cout << "You Pass" << endl;
        break;
    case 'F':
        cout << "You fail" << endl;
        break;
    default:
        cout << "Invalid grade" << endl;
}
```

While Loops

```
int index = 1;
while(index <= 5){
    cout << index << endl;
    index++;
}

do{
    cout << index << endl;
    index++;
}while(index <= 5);
```

For Loops

```
for(int i = 0; i < 5; i++){
    cout << i << endl;
}
```

Exception Catching

```
double division(int a, int b) {
  if( b == 0 ) {
    throw "Division by zero error!";
  }
  return (a/b);
}

int main(){
    try {
      division(10, 0);
    } catch (const char* msg) {
     cerr << msg << endl;
    }
    return 0;
}
```

Classes and Objects

```
class Book{
    public:
        string title;
        string author;

        void readBook(){
            cout << "Reading " + this->title + " by " + this->author << endl;
        }
};

Book book1;
book1.title = "Harry Potter";
book1.author = "JK Rowling";

book1.readBook();
cout << book1.title << endl;

Book book2;
book2.title = "Lord of the Rings";
book2.author = "JRR Tolkien";

book2.readBook();
cout << book2.title << endl;
```

Constructors

```
class Book{
    public:
        string title;
        string author;

        Book(string title, string author){
            this->title = title;
            this->author = author;
        }

        void readBook(){
            cout << "Reading " + this->title + " by " + this->author << endl;
        }
};

Book book1("Harry Potter", "JK Rowling");
cout << book1.title << endl;

Book book2("Lord of the Rings", "JRR Tolkien");
cout << book2.title << endl;
```

Getters and Setters

```
class Book{
    private:
        string title;
        string author;
    public:
        Book(string title, string author){
            this->setTitle(title);
            this->setAuthor(author);
        }

        string getTitle(){
            return this->title;
        }
        void setTitle(string title){
            this->title = title;
        }
        string getAuthor(){
            return this->author;
        }
        void setAuthor(string author){
            this->author = author;
```

```
    }

    void readBook(){
        cout << "Reading " + this->title + " by " + this->author << endl;
    }
};

Book book1("Harry Potter", "JK Rowling");
cout << book1.getTitle() << endl;

Book book2("Lord of the Rings", "JRR Tolkien");
cout << book2.getTitle() << endl;
```

Inheritance

```
class Chef{
    public:

    string name;
    int age;

    Chef(string name, int age){
        this->name = name;
        this->age = age;
    }

    void makeChicken(){
        cout << "The chef makes chicken" << endl;
    }

    void makeSalad(){
        cout << "The chef makes salad" << endl;
    }

    void makeSpecialDish(){
        cout << "The chef makes a special dish" << endl;
    }
};

class ItalianChef : public Chef{
    public:

    string countryOfOrigin;

    ItalianChef(string name, int age, string countryOfOrigin) : Chef(name, age){
        this->countryOfOrigin = countryOfOrigin;
    }
```

```
        void makePasta(){
            cout << "The chef makes pasta" << endl;
        }

        // override
        void makeSpecialDish(){
            cout << "The chef makes chicken parm" << endl;
        }
};

Chef myChef("Gordon Ramsay", 50);
myChef.makeChicken();

ItalianChef myItalianChef("Massimo Bottura", 55, "Italy");
myItalianChef.makeChicken();
cout << myItalianChef.age << endl;
```

Abstract Classes and Methods

```
class Vehicle{
    public:
        virtual void move() = 0;
        void getDescription(){
            cout << "Vehicles are used for transportation" << endl;
        }
};

class Bicycle : public Vehicle{
    public:
        void move(){
            cout << "The bicycle pedals forward" << endl;
        }
};

class Plane : public Vehicle{
    public:
        virtual void move(){
            cout << "The plane flys through the sky" << endl;
        }
};

Plane myPlane;
myPlane.move();
```