```c
#include <stdio.h>
//standard I/O

#include <stdbool.h>
//boolean data type

#include <string.h>
//methods upon string

#include <math.h>
//math function

#include <stdlib.h>
//C programming standard library

/**
    This is a notebook of C-programming basics
*/

//call by reference
void increment(int*);

//call by reference -- array
int sumOfArray(int*, int);


//C structure: a group of variables that may have different data types
struct myFirstStructure {
    char myLastInitial;
    char fullName[20];
    int myAge;
    double myHeight;


};
//end the structure with a semicolon



//the main method
int main() {
    printf("Hello World!\n");
    //standard output


    int variable = 4;
    double height = 4.576;
    int h = (int)height;
    //casting


    char letter = 'C';
    printf("my %d variable is %d feet\n", variable, h);
    //formating output
    int num = 9;
    const int ten = 10;
    //constant

    bool isFound = false;
    //notice: boolean is not built-in data in C
    //#include<stdbool.h>


    printf("%d",isFound);
    //boolean returns as integers (0 or 1)

    printf("%lu\n", sizeof(variable));
    //sizeof: always return memory size


    int arr[] = {3,5,8};
    printf("%d", arr[1]);
    //array
    int fixedArr[20]= {3,2,7};
    //array of fixed size
    //set aside 20 slots with 3 filled

    char myString[] = "What the heck?";
    //C programming does not have String
    //instead, we use an array of chars
    //the only way to declare a String in C programming
    //in C programming, string has to be terminated by a null ('\0') character
    //always needs one extra space for '\0'

    printf("%s\n", myString);
    //but in terms of output, we can still use "%s"
    printf("%c\n", myString[0]);
    //The first character


    printf("%d\n", strlen(myString));
    //the length of the string

    char string2[] = "Here you go!";
    strcat(myString, string2);
    printf("%s", myString);
    //string concatenation


    char string3[] = "Have a good one";
    strcpy(string3, myString);//copy myString to string3
    printf("%s\n",string3);
    //string copying

    printf("%d\n", strcmp(string2,myString));
    //string comparing
    //return 0 if two strings are equal
    //return non-zero if two strings are different

    int userInput;
    scanf("%d",&userInput);
    char ch;
    scanf("%c", &ch);
    //take integer/char as input


    scanf("%d %c", &userInput,&ch);
    printf("%d        %c", userInput,ch);
    //take multiple inputs at the same time

    char string4[20];
    scanf("%s", string4);
    printf("%s", string4);
    //caveat: to take a string input
    //1. the size must be specified;
    //2. no need for reference operator
    //"scanf" can only take one word

    char string5[19];
    fgets(string5, sizeof(string5), stdin);
    printf("%s", string5);
    //"fgets" can take an entire line as input

    //reference operator "&" -- return the memory address

    int myAge = 19;
    int* ptr = &myAge; //data type*: create a pointer variable
    //alternative: int *ptr = &myAge;
    //printf("%p", ptr);
    //pointer variable (in terms of hexadecimal)

    int myNumbers[4] = {25, 50, 75, 100};

    printf("%p\n", myNumbers);
    //in C, array's name is actually a pointer variable
    //a pointer variable that points to the first element of the array
    //"myNumbers" is equivalent to "&myNumbers[0]" (base address)
    //For array: regular notation + pointer notation
    //address:       &arr[i] or  (arr + i)
    //value:          arr[i] or *(arr + i)
    //takeaway: arrays and pointers are different data types;
    //But they are used in a similar manner.
    //However, array is not applicable for arithmetic like arr++ (NO)

    //*: dereference operator -- return the actual element
    printf("%d\n", *myNumbers);
    //first element
    printf("%d\n", *(myNumbers+1));
    //second element
    printf("%d\n", *(myNumbers+2));
    //third element

    printf("%f\n", sqrt(16));
    //square root

    printf("%d\n", -20);
    //absolute-value

    printf("%f\n", pow(4, 3));
    //power function

    printf("%f\n", ceil(1.4));
    //round up

    printf("%f\n", floor(1.4));
    //round down
    int number = 3;
    increment(&number);
    //pointer as function arguments
    //pass in the address as arguments
    printf("%d\n", number);

    int myArray[] = {3,2,4,5,8};
    int size = sizeof(myArray)/sizeof(myArray[0]);
    //the standard way to compute the size of the array
    int sum = sumOfArray(myArray, size);
    //parameters: array ("pointer") and size ("int")
    printf("%d\n", sum);

    doubleIt(myArray, size);
    for(int i = 0; i < size; i++){
        printf("%d ", *(myArray+i));
    }
    printf("\n");


    //dynamic memory allocation: (use heap for memory)
    //the
    int a;//this variable goes onto stack


    int* p = (int*)malloc(sizeof(int));
    //malloc will return a void pointer
    //typecast it to integer pointer
    *p = 10;
    //modify the value by dereferencing it

    free(p);
    //clear unnecessary memory on the heap

    int* p2 = (int*)malloc(20*sizeof(int));
    //allocate an array of size 20 on the heap
    //no initialization is done
    //caveat:
    //if the size of the array is not fixed (depends on the runtime)
    //we cannot declare an array like int arr[10]
    //we have to use dynamic memory allocation (like malloc)

    free(p2);

    int* p3 = (int*)calloc(20, sizeof(int));
    //alternative: calloc: also initialize all to be zero

    int* p4 = (int*) realloc(p3,21*sizeof(int));
    //realloc: the previous pointer + new size
    //copy values from the previous pointer
    //automatically de-allocated the previous pointer

    printf("%d\n", *(p3+2));

    //file I/O:
    FILE* fptr;
    fptr = fopen("filename", w);
    //write to a file
    fptr = fopen("filename", a);
    //append new data to a file
    fprintf(fptr,"some text");
    //applicable for "w" (writing) and "a" (append)


    fptr = fopen("filename", r);
    //read from a file

    char myLine[100];
    if(fptr != NULL){
        while(fgets(myLine, sizeof(myLine),fptr)){
            printf("%s", myLine);
        }
    }else{
        printf("Not able to open the file.");
    }

    fclose(fptr);
    //best practice:
    //close the file


    struct myFirstStructure s1;
    //create a structure
    //we can also create multiple structures

    s1.myAge = 19;
    s1.myLastInitial = 'P';
    s1.myHeight = 6.3;
    //initialize values
    strcpy(s1.fullName, "Yusen Peng");
    //to work with strings, we can only use "strcpy" command

    return 0;

}


int addTwoNum(int a, int b){
    return a+b;
}


//pass by reference
void increment(int* p){
    //pointer parameter:
    //pass in the address of the integer variable
    *p = *p + 1;
    //dereference:
    //change integer value in place

}


//array -- pass by reference
//The size of the array must be passed as a parameter
int sumOfArray(int* arr, int arrSize){
    //arr: a pointer variable
    int sum = 0;
    for(int i = 0; i < arrSize; i++){
        sum += *(arr+i);
    }
    return sum;
}

//modify elements in an array -- pass by reference
void doubleIt(int* arr, int size){
    //arr: a pointer variable
    for(int i = 0; i < size; i++){
        *(arr+i) = 2*(*(arr + i));
    }
}

//pointers as function returns
//best practice:
//when we use pointers as function returns
//always use dynamic memory allocation
int* add(int* a, int* b){
    int* sum = (int*)malloc(sizeof(int));
    //use heap (dynamic memory allocation) instead of stack
    //because memory at stack will be automatically de-allocated
    //once the function finishes executing
    //however, the memory at heap will not be de-allocated
    //unless we use "free()" command explicitly

    *sum = *a + *b;

    return sum;
}

/**
 * Leetcode#1: two sum
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* twoSum(int* nums, int numsSize, int target, int* returnSize){
    *returnSize = 2;
    //dereference the return size to be 2

    int* result = (int*) malloc(2*(sizeof(int)));
    //use dynamic memory allocation as required

    for(int i=0;i < numsSize;i++){
        for(int j=i+1;j < numsSize;j++){
            if(*(nums + i) + *(nums + j) == target){
                *result = i;
                *(result+1) = j;
            }
        }
    }

    return result;
}
```

```cpp
#include <iostream>

#include <string>
//to use string in C++, we need to include string library

#include <cmath>
//library for more available math function

#include <fstream>
//file I/O stream

#include <vector>
using namespace std;

//create a class
class Car{
    public:
    //public members
        string brand;
        string model;
        int year;
        //some attributes

        Car(string b, string m, int y){
            brand = b;
            model = m;
            year = y;
        }
        //constructor


        void printCar(){
            cout << brand << model << year;
        }
        //some methods (method defined inside the class)

        int carYear();//getter
        //alternative way: method header inside the class

    private:
    //private members (by default, members are private)
        int days;

};

int Car::carYear(){
    return Car::year;
}
//finish the method implementation outside of the class



//inheritance
//class subClass: public superClass{...};
class truck: public Car{
    public:
        double truck_size;
} ;


class wheel{

};


//multiple inheritance
```

```cpp
class bike: public Car, public wheel{

};
//separate by commas

//recursion
int sumOfDigits(int number = 0){
    //default parameter value "  = *some value* "
    int sum = 0;
    if(number < 10){
        sum = number;
    }else{
        int onesDigit = number % 10;
        sum = onesDigit + sumOfDigits(number/10);
    }
}

//pass by reference
int sumOfArray(int* arr, int arrSize){
    int sum = 0;
    for(int i = 0; i < arrSize; i++){
        sum += *(arr + i);
    }
    return sum;
}

struct structureTemplate {
    int myAge;
    string myName;
    char myInitial;
};
//named structure
//declare a "structure type" outside of the main
//treat this kind of structure as "a new data type"

//for vector: pass by reference
void print_vector(vector<int> &vec){
    for(int i = 0; i < vec.size(); i++){
        cout << vec[i] << " ";
    }
}


int main() {
    cout << "Hello World!";
    //standard output

    const int myAge = 10;
    //a constant

    bool isVisited = false;
    //return value: 0 or 1 (logic value)

    cout << "I am " << myAge << " years old!";
    //For output stream concatenation (use "<<" to concatenate instead of "+")
    //for string concatenation (simply use "+")

    string userInput;
    //string type: lowercase "s"

    //cin >> userInput;
    //take user input, but only take the first token
    //in order to take an entire line, we need to do the following:
    //getLine(cin,userInput);


    string last_name = "Peng";
    string first_name = "Yusen";
    string fullName = first_name.append(last_name);
    //an alternative way to concatenate strings

    int pos = 0;
    cout << fullName[pos];
    //access strings just like arrays


    //C++ Math
    min(3,2);
    max(5,9);
    //these two functions is independent of the <cmath> library

    sqrt(16);
    round(2.3);
    log(10);//natural log
    pow(2,5);
    abs(-2);

    //C++ arrays
    int arr[10] = {1,2,3,4,5,6,7,8,9,10};
    //very similar to C language
    //the array's size can be omitted

    int arr_size = sizeof(arr) / sizeof(int);
    //determine the array's size -- very similar to C
    cout << sumOfArray(arr,arr_size);

    struct {
        int myAge;
        string myName;
        char myInitial;

    } myFirstStructure;
    //directly declare a structure in main and manipulate it
    myFirstStructure.myAge = 19;

    structureTemplate anInstance;
    //create an instance of "named structure" type
    anInstance.myAge = 21;
    anInstance.myInitial = 'P';

    cout << sumOfDigits(12345);

    ofstream my_file("filename.txt");
    my_file << "some text to write into file";
    my_file.close();
    //create and write a file


    string text;
    ifstream his_file("filename.txt");
    //while(getLine(his_file, text)){
    //    cout << text;
    //}
    his_file.close();


    //dynamic memory allocation
    //keyword: new
    //For a single variable
    int* ptr_int = new int;
    *ptr_int = 34;
    //store the value 5 in the heap
    printf("%d", *ptr_int);
    delete ptr_int;
    //explicitly delete

    //dynamic memory allocation for array
    int* anotherArray = new int[4];
    for(int i = 0; i < 4; i++){
        *(anotherArray + i) = 2*i;
        cout << *(anotherArray+i);
    }
    delete anotherArray;

    //vector
    vector<int> my_vector;
    my_vector.push_back(12);
    //add element at the very end

    cout << my_vector[my_vector.size()-1];

    my_vector.pop_back();
    //remove the last element


  return 0;
}


```