1 report

The const keyword specifies that a variable's value is constant tells the compiler to prevent anything from modifying it (read-only)

const double pi = 3.14;

const int light\_speed = 299792458;

these may be a few examples of variables you do not want to be changed or altered at all you only want them to be read-only

Namespace - provides a solution for preventing name conflicts in large projects. Each entity needs a unique name. A namespace allows for identically named entities as long as the namespaces are different.

namespace first{

int x = 0;

}

int main () {

using namespace first;

std::cout << x;

} answer 0

typedef – reserved keyword used to create an additional name (alias) for another data type. New identifier for an existing type. Helps with readability and reduces typos.

#include <iostream>

#include <vector>

//typedef std::vector<std::pair<std::string, int>> pairlist\_t;

typedef std::string text\_t;  //using text\_t = std::string;

typedef int number\_t;  //using number\_t = int;

int main(){

//  pairlist\_t pairlist;

    text\_t firstname = "Yulia"; //equal string

    number\_t age = 21;

    return 0;

}

Arithmetic operators – return the result of a specific arithmetic operation (+ - \* /). Program doesn’t need parenthesis, it does it like we, first division and multiplication, then addition and subtraction.

#include <iostream>

int main(){

    int students = 20;

    //addition

    students = students + 1;

    students += 1;

    students ++;

    //subtraction

    students = students - 1;

    students -= 1;

    students --;

    //multiplication

    students = students \* 2;

    students \*= 2;

    //division

    students = students / 2;

    students /= 2;

    return 0;

}

type conversion – conversion a value of one data type to another. Implicit (неявное) – automatic. Explicit (явное) – Precede value with new data type (int).

#include <iostream>

int main(){

    //implicit

    double pi = (int) 3.14;

    //ans 3

    //explicit

    int correct = 8;

    int questions = 10;

    double score = correct / (double)questions \* 100;

    //ans 80%

    return 0;

}

Cin - std::getline (std::cin, name)

Useful math related functions – the most popular functions are here. But all functions you can see on website [www.cplusplus.com/reference/cmath](http://www.cplusplus.com/reference/cmath)

#include <iostream>

#include <cmath>

int main(){

    double x = 3.14;

    double y = 4;

    double z;

    z = std::max(x, y); //максимальное значение из двух

    z = std::min(x, y); //минимальное значение из двух

    z = pow(2, 4); //возведение в степень

    z = sqrt(9); //корень

    z = abs(-3); //модуль

    z = round(x); //округление по правилам математики

    z = ceil(x); //округление вверх

    z = floor(x); //округление вниз

    return 0;

}

If statements – do something if a condition is true. If not, then don’t do it.

#include <iostream>

using namespace std;

int main(){

    int age;

    cout << "Enter your age: ";

    cin >> age;

    if (age >= 18){

        cout << "Welcome to the site!";

    }else if(age < 0){

        cout << "You are not born yet!";

    }else{

        cout << "You are not old enough to enter!";

    }

    return 0;

}

Switch – alternative to using many “else if” statements compare one value against matching cases.

#include <iostream>

using namespace std;

int main(){

    int month;

    cout << "Enter num of month: ";

    cin >> month;

    switch(month){

        case 1:

            std::cout << "It is January";

            break;

        case 2:

            std::cout << "It is February";

            break;

        case 3:

            std::cout << "It is March";

            break;

        case 4:

            std::cout << "It is April";

            break;

        case 5:

            std::cout << "It is May";

            break;

        case 6:

            std::cout << "It is June";

            break;

        case 7:

            std::cout << "It is July";

            break;

        case 8:

            std::cout << "It is August";

            break;

        case 9:

            std::cout << "It is September";

            break;

        case 10:

            std::cout << "It is October";

            break;

        case 11:

            std::cout << "It is November";

            break;

        case 12:

            std::cout << "It is December";

            break;

        default:

            std::cout << "It is not a month";

    }

    return 0;

}

Ternary operator – replacement to an if/else statement. Condition – expression1 : expression2.

#include <iostream>

using namespace std;

int main(){

    int grade = 75;

    grade >= 60 ? std::cout << "You pass!" : std::cout << "You fail!";

    return 0;

}

&& - check if two conditions are true.

|| - check if at least one of two conditions is true.

! – reserves the logical state of its operand

2 report.

Using string methods. Use [www.cplusplus.com/reference/string/string](http://www.cplusplus.com/reference/string/string) for more.

#include <iostream>

int main(){

    std::string name;

    std::cout << "Enter your name!";

    std::cin >> name;

    name.length();

    name.empty();

    name.clear();

    name.append("@gmail.com"); //yulia -> yulia@gmail.com

    name.at(0); //what is the first letter (y)

    name.insert(0, "@"); //yulia -> @yulia

    name.find('l'); //yulia -> ans=2(index)

    name.erase(0, 2); //yulia -> lia (delete)

    return 0;

}

While loops.

#include <iostream>

int main(){

    std::string name;

    while(name.empty()){

        std::cout << "Enter your name!";

        std::cin >> name;

    }

    std::cout <<"Hello, " << name;

    return 0;

}

Do while loop. – do some block of code first, then repeat again if condition is true.

#include <iostream>

int main(){

    int number;

    do{

        std::cout << "Enter a positive number: ";

        std::cin >> number;

    }while(number < 0);

    std::cout << "The number is: " << number;

    return 0;

}

For loop - a loop that will execute a block of code a specified amount of times.

#include <iostream>

int main(){

    for (int i = 0; i <= 10; i++){

        std::cout << i << '\n';

    }

    std::cout << "Happy new year!";

    return 0;

}

Break – break out of a loop. Continue – skip current iteration.

#include <iostream>

using namespace std;

int main(){

    for (int i = 1; i <= 5; i++){

        if (i == 3){

            continue;

        }

        cout << i << '\n';

    }  //ans - 1 2 4 5

    for (int x = 1; x <= 10; x++){

        if (x == 7){

            break;

        }

        cout << x << '\n';

    } //ans - 1 2 3 4 5 6

    return 0;

}

Nested loops – loop in loop.

#include <iostream>

using namespace std;

int main(){

    int rows, columns;

    cin >> rows >> columns;

    for (int i = 1; i <= rows; i++){

        for(int j = 1; j <= columns; j++){

            cout << j << ' ';

        }

        cout <<'\n';

    }

    /\*

    1 2 3 4 5 6 7 8 9 10

    1 2 3 4 5 6 7 8 9 10

    1 2 3 4 5 6 7 8 9 10

    \*/

    return 0;

}

Random number generator.

#include <iostream>

using namespace std;

int main(){

    srand(time(NULL));

    int num = (rand()% 15)+1; //between 1, 15

    cout<< num;

    return 0;

}

Random event generator.

#include <iostream>

using namespace std;

int main(){

    srand(time(0));

    int num = (rand()% 5)+1; //between 1, 5

    switch (num){

        case 1: cout << "You win a bumper sticker!"; break;

        case 2: cout << "You win a t-shirt!"; break;

        case 3: cout << "You win a free lunch!"; break;

        case 4: cout << "You win a gift card!"; break;

        case 5: cout << "You win concert tickets!"; break;

    }

    return 0;

}

Number guessing game.

#include <iostream>

using namespace std;

int main(){

    int num, guess, tries;

    srand(time(NULL));

    num = (rand() % 100) + 1; //between 1, 5

    do{

        cout << "Enter a guess between (1-100): ";

        cin >> guess;

        tries++;

        if (guess > num){

            cout << "too high! \n";

        }else if(guess < num){

            cout << "too low! \n";

        }else{

            cout <<"Correct! "<< tries <<" of tries. \n";

        }

    }while(guess != num);

    return 0;

}

User defined (определяемая пользователем функция). Function – a block of reusable code.

#include <iostream>

#include <vector>

using namespace std;

void happyBirthday(std::string name, int age);

int main(){

    std::string name = "Bro";

    int age = 20;

    happyBirthday(name, age);

    return 0;

}

void happyBirthday(std::string name, int age){

    std::cout << "Happy Birthday to " << name << '\n';

    std::cout << "Happy Birthday to " << name << '\n';

    std::cout << "Happy Birthday dear " << name << '\n';

    std::cout << "Happy Birthday to " << name << '\n';

    std::cout << "You are " << age << " years old!" <<'\n';

}

Return keyword

#include <iostream>

double square (double length);

int main(){

    double length = 6.0;

    double area = square (length);

    std::cout << "Area: " << area << "cm^2\n";

    return 0;

}

double square (double length){

    return length \* length;

}

#include <iostream>

std::string concatStrings(std::string string1, std::string string2);

int main(){

    std::string firstName = "Yulia";

    std::string secondName = "Kravtsova";

    std::string fullName = concatStrings(firstName, secondName);

    std::cout << "Hello " << fullName;

    return 0;

}

std::string concatStrings(std::string string1, std::string string2){

    return string1 + ' ' + string2;

}

Overload function.

#include <iostream>

using namespace std;

void pizza();

void pizza(string topping1);

void pizza(string topping1, string topping2);

int main(){

    pizza("pepperoni", "mushrooms");

    return 0;

}

void pizza(){

    cout << "Here is your pizza!\n";

}

void pizza(string topping1){

    cout << "Here is your " << topping1 << " pizza!\n";

}

void pizza(string topping1, string topping2){

    cout << "Here is your " << topping1 << " and " << topping2 << " pizza!\n";

}

Variable scope. Local variables – declared inside a function or block.

#include <iostream>

using namespace std;

void num();

int main(){

    num();

    return 0;

}

void num(){

    int myNum = 2;

    cout << myNum;

}//2

Global variables – declared outside of all functions.

#include <iostream>

using namespace std;

int myNum = 3;

void num();

int main(){

    num();

    cout <<myNum;

    return 0;

}

void num(){

    cout << myNum;

}

//33

3 report.

Array – a data structure that can hold multiple values. Values are accessed by an index number.

#include <iostream>

using namespace std;

int main(){

    string car[] = {"Corvette", "Mustang", "Camry"};

    cout << car[0] << ' ' << car[1] << ' ' << car[2] << '\n';

    car[0] = "Camaro";

    cout << car[0] << ' ' << car[1] << ' ' << car[2] << '\n';

    string cars[3];

    cars[0] = "BMW";

    cars[1] = "Mustang";

    cars[2] = "Camry";

    cout << cars[0] << ' ' << cars[1] << ' ' << cars[2];

    return 0;

}

Sizeof() – determines the size in bytes of a: variable, data type, class, objects, etc.

#include <iostream>

using namespace std;

int main(){

    string name = "Yulia";

    double gpa = 2.5;

    char grade = 'A';

    bool students = true;

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

    cout << sizeof(name) << ' ' << sizeof(gpa) << ' ' << sizeof(grade) << ' ' << sizeof(students) << ' ' << sizeof(grades) <<'\n';

    cout << sizeof(grades)/sizeof(char) << " elements\n";

    return 0;

}

Iterate over an array.

#include <iostream>

using namespace std;

int main(){

    string name = "Yulia";

    double gpa = 2.5;

    char grade = 'A';

    bool students = true;

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

    cout << sizeof(name) << ' ' << sizeof(gpa) << ' ' << sizeof(grade) << ' ' << sizeof(students) << ' ' << sizeof(grades) <<'\n';

    cout << sizeof(grades)/sizeof(char) << " elements\n";

    return 0;

}

For each loop – loop that eases the traversal over an iterable data set.

#include <iostream>

using namespace std;

int main(){

    string students[]= {"Spongebob", "Patrick", "Squidward", "Sandy"};

    for (string student : students){

        cout << student <<'\n';

    }

    return 0;

}

Pass array to a function.

#include <iostream>

using namespace std;

double getTotal(double prices[], int size);

int main(){

    double prices[] = {49.99, 15.05, 75, 9.99};

    long long size = sizeof(prices)/sizeof(prices[0]);

    double total = getTotal(prices, size);

    cout << "$" << total;

    return 0;

}

double getTotal(double prices[], int size){

    double total = 0;

    for (int i = 0; i < size; i++){

        total+= prices[i];

    }

    return total;

}

Search an array for an element.

#include <iostream>

using namespace std;

int searchArray(int array[], int size, int element);

int main(){

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

    int size = sizeof(numbers)/sizeof(numbers[0]);

    int index, mynum;

    cout << "Enter an element to search for: " <<'\n';

    cin >> mynum;

    index = searchArray(numbers, size, mynum);

    if (index != -1){

        cout << mynum << " is on index " << index;

    }else{

        cout << mynum << " is not in the array";

    }

    return 0;

}

int searchArray(int array[], int size, int element){

    for (int i = 0; i < size; i++){

        if (array[i] == element){

            return i;

        }

    }

    return -1;

}

Sort an array. Descending order.

#include <iostream>

using namespace std;

void sort(int array[], int size);

int main(){

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

    int size = sizeof(array)/sizeof(array[0]);

    sort(array, size);

    for(int element : array){

        cout << element <<' ';

    }

    return 0;

}

void sort(int array[], int size){

    int temp;

    for(int i = 0; i < size - 1; i++){

        for(int j = 0; j < size - i - 1; j++){

            if(array[j] > array[j + 1]){ //<

                temp = array[j];

                array[j] = array[j + 1];

                array[j + 1] = temp;

            }

        }

    }

}

Fill() function – fills a range of elements with a specified value. Fill (begin, end, value)

#include <iostream>

using namespace std;

int main(){

    const int size = 100;

    string food[size];

    fill(food, food + (size/2), "pizza");

    fill(food + (size/2), food, "hamburger");

    for(string foods : food){

        cout << foods <<'\n';

    }

    return 0;

}

Multidimensional arrays.

#include <iostream>

using namespace std;

int main(){

    string cars[][3] = {{"Mustang", "Escape", "F-150"},

                        {"Corvette", "Equinox", "Silverado"},

                        {"Challenger", "Durango", "Ram 1500"}};

    cout << cars[1][2] << '\n';

    int rows = sizeof(cars)/sizeof(cars[0]);

    int columns = sizeof(cars[0])/sizeof(cars[0][0]);

    for(int i = 0; i < rows; i++){

        for(int j = 0; j < columns; j++){

            cout << cars[i][j] << ' ';

        }

        cout << '\n';

    }

    return 0;

}

Memory address – a location in memory where data is stored. A memory address can accessed with (address-of operator).

#include <iostream>

using namespace std;

int main(){

    string name = "yulia";

    int age = 21;

    bool student = true;

    cout << &name << ' ' << &age << ' ' << &student;

    return 0;

}

//0x8aa91ff7f0 0x8aa91ff7ec 0x8aa91ff7eb

Const parameter – parameter that is effectively read-only code is more secure and conveys intent useful for references and pointers.

#include <iostream>

using namespace std;

void func (const string &name, const int &age);

int main(){

    string name = "yulia";

    int age = 21;

    func(name, age);

    return 0;

}

void func (const string &name, const int &age){

    //if we want to change parameters (name = "kravtsova", age 18),

    // there will be an error. so, we use const to avoid changes.

    cout << name << ' ' << age;

}

Pointers – variable that stores a memory address of another variable. Sometimes it’s easer to work with an address. & address – of operator. \* dereference operator.

#include <iostream>

using namespace std;

int main(){

    string name = "yulia";

    int age = 21;

    string \*pname = &name;

    int \*page = &age;

    cout << \*pname << ' ' << \*page;

    return 0;

}

Null value – 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 represents a null pointer literal. Nullptr are helpful when determining if an address was successfully assigned to a pointer.

#include <iostream>

using namespace std;

int main(){

    int \*pointer = nullptr;

    int x = 123;

    pointer = &x;

    if (pointer == nullptr){

        cout << "addresss was not assigned";

    } else{

        cout << "address was assigned" << '\n';

        cout << \*pointer;

    }

    return 0;

}

4 report.

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 stack. Useful when we don’t know how much memory we will need. Makes our programs more flexible, especially when accepting user input.

#include <iostream>

using namespace std;

int main(){

    char \*pgrades = NULL;

    int size;

    cin >> size;

    pgrades = new char[size];

    for(int i = 0; i < size; i++){

        cout << "Enter grade # " << i + 1 << ' ';

        cin >> pgrades[i];

    }

    for(int i = 0; i < size; i++){

        cout << pgrades[i] << ' ';

    }

    delete[] pgrades;

    return 0;

}

Recursion – a programming technique where a function invoke itself from within break a complex into a repeatable single step. (iterative vs recursion).

#include <iostream>

using namespace std;

//iteration

int factorial(int num);

int main(){

    cout << factorial(10);

    return 0;

}

int factorial(int num){

    int res = 1;

    for(int i = 1; i <= num; i++){

        res \*=i;

    }

    return res;

}

//recursion

int factorial(int num);

int main(){

    cout << factorial(10);

    return 0;

}

int factorial(int num){

    if(num > 1){

        return num \* factorial(num -1);

    }else{

        return 1;

    }

}

Function template – describes what a function looks like. Can be used to generate as many overloaded functions as needed, each using different data types.

#include <iostream>

using namespace std;

//there you should put all data types, otherwise it will be an error

int max(int x, int y){

    return (x > y) ? x : y;

}

double max(double x, double y){

    return (x > y) ? x : y;

}

char max(char x, char y){

    return (x > y) ? x : y;

}

int main(){

    cout << max(1.1, 2.1);

    return 0;

}

//works with different data types. but equal.

template <typename Thing>

Thing maxim (Thing x, Thing y){

    return (x > y) ? x : y;

}

int main(){

    cout << maxim(1.1, 2.1) <<'\n';

    return 0;

}

Struct – a structure that group related variables under one name. structs can contain many different data types (string, int, double, etc). variable in a struct are known as “members”. Members can be access with “class member access operator.

#include <iostream>

using namespace std;

struct student{

    string name;

    double gpa;

    bool enrolled;

};

int main(){

    student student1;

    student1.name = "yulia";

    student1.gpa = 5;

    student1.enrolled = true;

    cout << student1.name << ' ' << student1.gpa << ' ' << student1.enrolled;

    return 0;

}

Pass structs as arguments.

#include <iostream>

using namespace std;

struct car{

    string model;

    int year;

    string color;

};

void printcar(car &car);

void paintcar(car &car, string color);

int main(){

    car car1;

    car car2;

    car1.model = "mustang";

    car1.year = 2023;

    car1.color = "red";

    car2.model = "corvette";

    car2.year = 2024;

    car2.color = "blue";

    paintcar(car1, "gold");

    printcar(car1);

    printcar(car2);

    return 0;

}

void printcar(car &car){

    cout << &car << '\n';

    cout << car.model << ' ' << car.year << ' ' << car.color << '\n';

}

void paintcar(car &car, string color){

    car.color = color;

}

Enums – a user-defined data type that consist of paired named-integer constants. GREAT if you have a set of potential options.

#include <iostream>

using namespace std;

enum day {sunday = 0, monday = 1, tuesday = 2, wednesday = 3,

         thursday = 4, friday = 5, saturday = 6};

int main(){

    day today = friday;

    switch(today){

        case 0: cout << "It's sunday!\n"; break;

        case 1: cout << "It's monday!\n"; break;

        case 2: cout << "It's tuesday!\n"; break;

        case 3: cout << "It's wednesday!\n"; break;

        case 4: cout << "It's thursday!\n"; break;

        case 5: cout << "It's friday!\n"; break;

        case 6: cout << "It's saturday!\n"; break;

    }

    return 0;

}

Object – a collection of attributes and methods. They can have characteristics and could perform actions. Can be used to mimic real world items (ex. Phone, Book, Dog). Created from a class which acts as a ‘blue-print’.

#include <iostream>

using namespace std;

class human {

    public:

        string name;

        string job;

        int age;

    void eat(){

        cout << "this person is eating\n";

    }

    void drink(){

        cout << "this person is drinking\n";

    }

    void sleep(){

        cout << "this person is sleeping\n";

    }

};

int main(){

    human human1;

    human1.name = "Rick";

    human1.job = "teacher";

    human1.age = 54;

    cout << human1.name <<'\n';

    cout << human1.job <<'\n';

    cout << human1.age <<'\n';

    human1.eat();

    human1.drink();

    human1.sleep();

    return 0;

}

Constructor – special method that is automatically called when an object is instantiated useful for assigning values to attributes as arguments.

#include <iostream>

using namespace std;

class student {

    public:

        string name;

        int age;

        double gpa;

    student(string name, int age, double gpa){

        this -> name = name;

        this -> age = age;

        this -> gpa = gpa;

        }

    /\*

     student(string x, int y, double z){

        name = x;

        age = y;

        gpa = z;

    }\*/

};

int main(){

    student student1 ("yulia", 20, 5);

    cout << student1.name << '\n';

    cout << student1.age << '\n';

    cout << student1.gpa << '\n';

    return 0;

}

Overloaded constructors – multiple constructors w/ same name but different parameters aloows for varying arguments when instantiating an object.

#include <iostream>

using namespace std;

class pizza {

    public:

        string topping1;

        string topping2;

    pizza(){

    }

    pizza(string topping1){

        this -> topping1 = topping1;

    }

    pizza(string topping1, string topping2){

        this -> topping1 = topping1;

        this -> topping2 = topping2;

    }

};

int main(){

    pizza pizza1("pepperoni");

    pizza pizza2("pepperoni", "mushrooms");

    pizza pizza3();

    cout << pizza1.topping1 << '\n';

    cout << pizza2.topping2 << '\n';

    cout << pizza3 << '\n';

    return 0;

}

Abstraction – hiding unnecessary data from outside a class. Getter – function that makes a private attribute READABLE. Setter – function that makes a private attribute WRITABLE.

#include <iostream>

using namespace std;

 //method setter

class Stove {

    private:

        int temp = 120;

    public:

    int getTemp (){

        return temp;

    }

    void setTemp(int temp){

        if(temp < 0){

            this -> temp = 0;

        }else if(temp >= 10){

            this -> temp = 10;

        }else{

            this -> temp = temp;

        }

    }

};

int main(){

    Stove stove;

    stove.setTemp(5);

    cout << stove.getTemp();

    return 0;

}

//method getter

class Stove {

    private:

        int temp = 120;

    public:

    int getTemp (){

        return temp;

    }

};

int main(){

    Stove stove;

    cout << stove.getTemp();

    return 0;

}

Inheritance – a class can receive attributes and methods from another class. Children classes inherit from a Parent class. Helps to reuse similar code found within multiple classes.

#include <iostream>

using namespace std;

class Animal {

    public:

    bool alive = true;

    void eat(){

        cout << "this dog is eating!\n";

    }

};

class Dog : public Animal{

    public:

    void bark(){

        cout << "this dog goes woof!\n";

    }

};

int main(){

    Dog dog;

    cout<< dog.alive << '\n';

    dog.eat();

    dog.bark();

    return 0;

}