C++ → C + OOPS

Why C++?

C	C++
Top down approach	Bottom up approach
Subset of C++	Superset of C
Procedural language	Procedural and Object-oriented language

C++ can be called a hybrid language.

```c
#include<stdio.h>

int main()
{
     printf("Welcome to FACE");
     return 0;
}
```

```cpp
#include<iostream>

using namespace std;

int main()
{
     cout<<"Welcome to FACE";
     return 0;
}
```

```cpp
#include <iostream>

using namespace std;

int main()
{
	cout<<"Welcome to FACE";
	return 0;
}
```

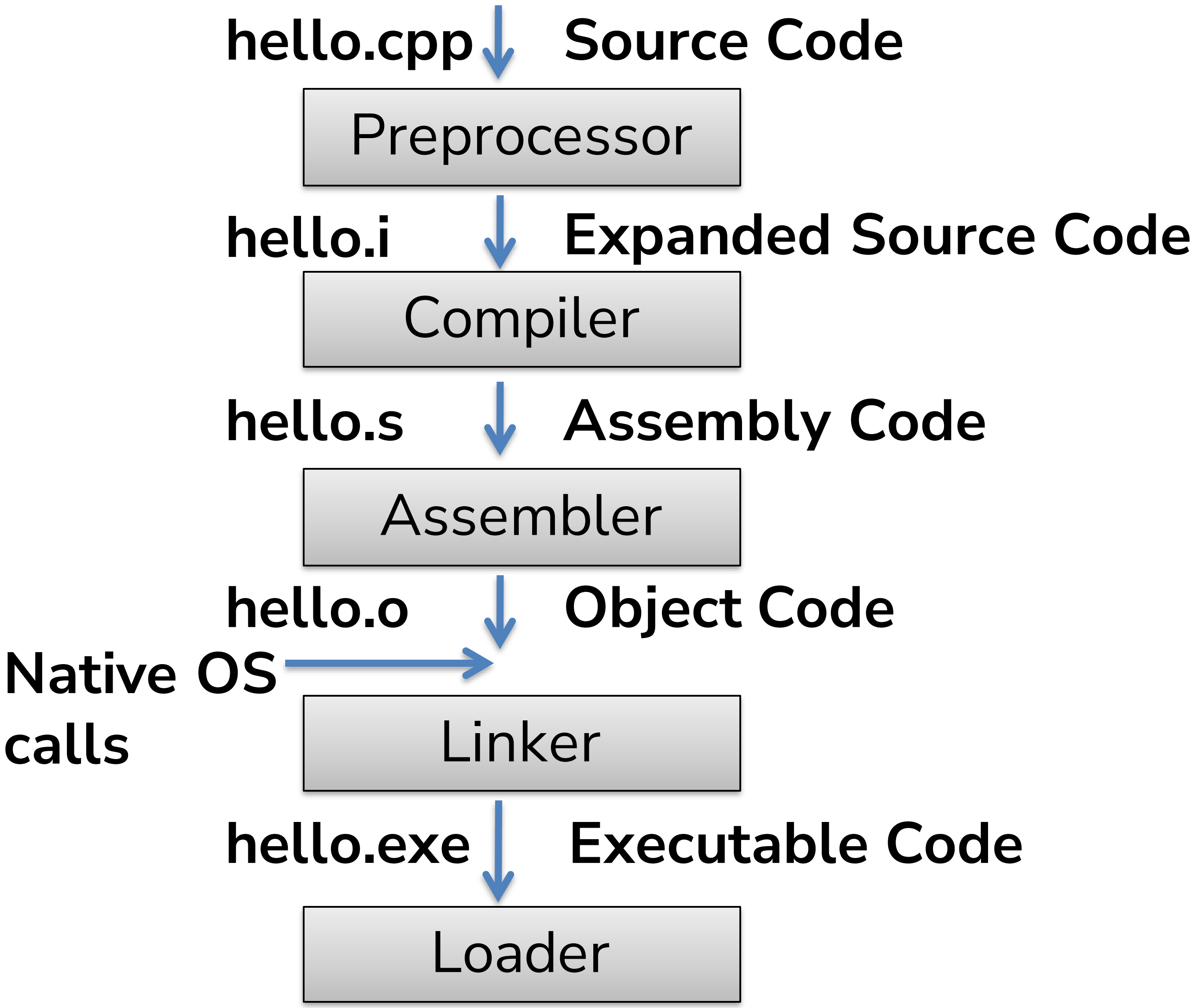
Explanation

Istream – supports input
Ostream – supports output
iostream – both input and output objects are included.

Includes all new standard library

Compilation and Execution

- Preprocessing
- Compilation
- Assembly
- Linking


hello.cpp

Source Code

Preprocessor

hello.i

Expanded Source Code

Compiler

hello.s

Assembly Code

Assembler

hello.o

Object Code

Native OS calls

Linker

hello.exe

Executable Code

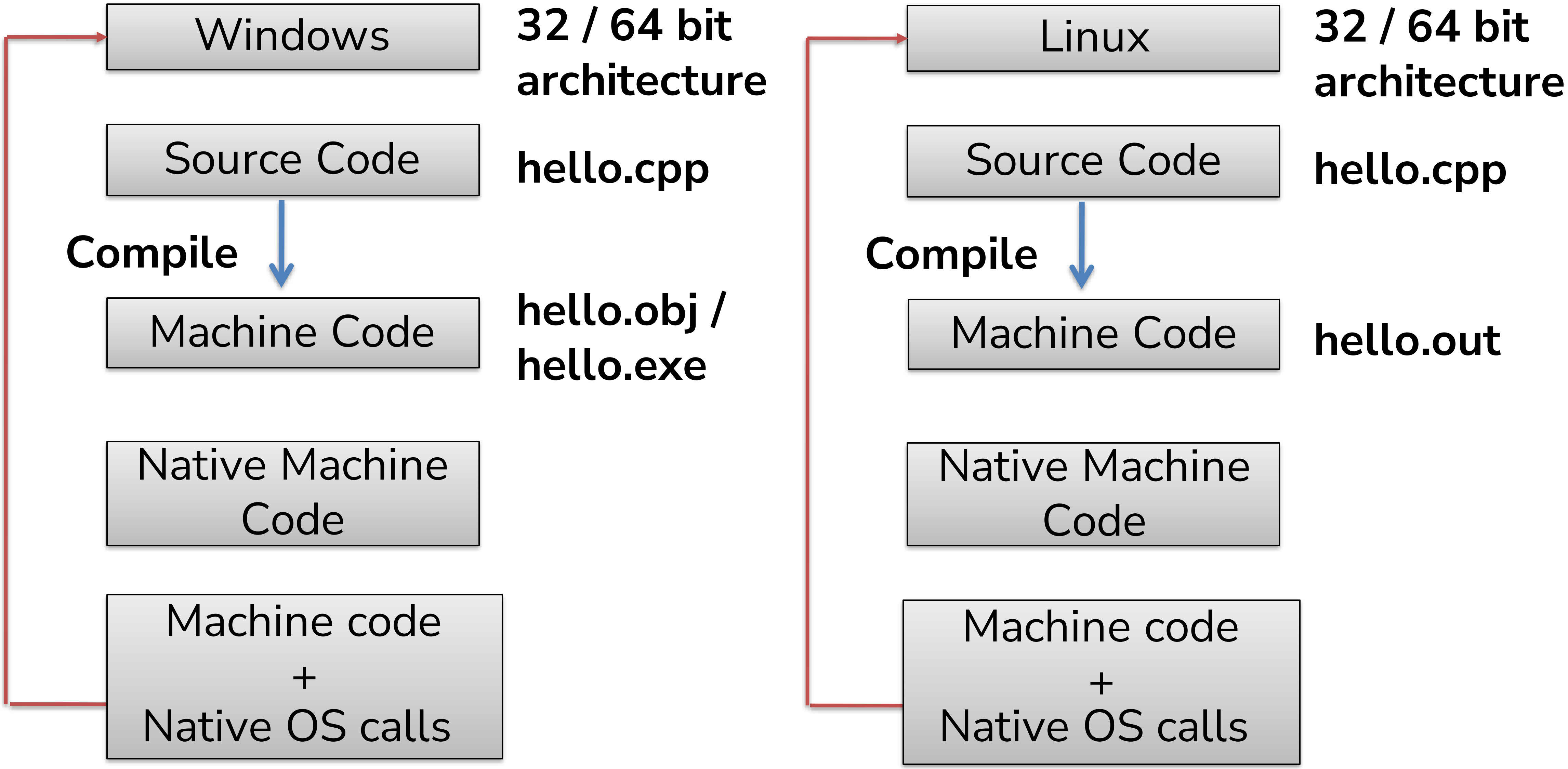
Loader

C++ Language

Platform dependent or Independent?

Dependent!!!

Why??



Windows

32 / 64 bit architecture

Source Code

hello.cpp

Compile

Machine Code

hello.obj / hello.exe

Native Machine Code

Machine code + Native OS calls

Linux

32 / 64 bit architecture

Source Code

hello.cpp

Compile

Machine Code

hello.out

Native Machine Code

Machine code + Native OS calls

Data Types

- Whole numbers → int
- Decimal numbers → float, double
- Character → char
- Logical values → boolean

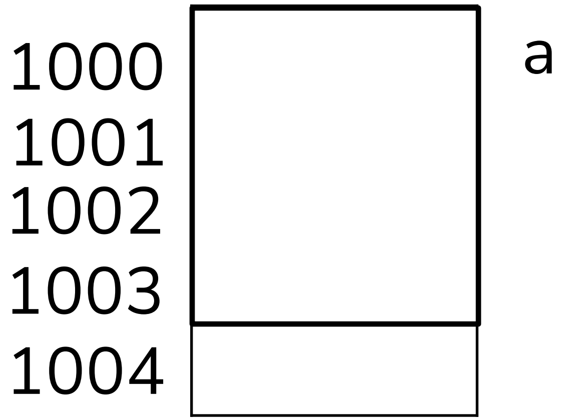
S.No	Data types	Storage size	Range
1.	char	1	-128 to 127
2.	int	4	-2,147,483,648 to 2,147,483,647
3.	float	4	-3.4e38 to +3.4e38
4.	double	8	-1.7e308 to +1.7e308
5.	long double	12	-3.4e38 to +3.4e38
6.	signed char	1	-128 to +127
7.	unsigned char	1	0 to 255

S.No	Data types	Storage size	Range
8.	short signed int	2	-32768 to +32767
9.	unsigned short int	2	0 to 65535
10.	signed int	4	-2147483648 to +2147483647
11.	unsigned int	4	0 to 4294967295
12.	signed long int	4	-2147483647 to +2147483647
13.	unsigned long int	4	0 to 4294967295

Variables

A Variable is a name given to the memory locations.

Eg : int a; // 4 bytes of memory will be allocated



RAM Visualization

Variable name

- int **num;**
- int **NUM;**
- int **Num;**
- **num, NUM, Num** – all three are different variables name

Rules

- Lowercase
- Uppercase
- Lowercase and Uppercase
- C Language is case sensitive

Variable name

- int **_num**; ✓
- int **num_**; ✓
- int **Num_ber** ; ✓
- int **Num.ber** ; ✗
- int **Num ber** ; ✗

Rules

- Underscore can be placed anywhere
- Except Underscore no other special characters are allowed (like dot, white space, etc.,)

Variable name

- int **9num**; ✗
- int **num9**; ✓
- int **Num_7_is** ; ✓
- int **float**; ✗

Rules

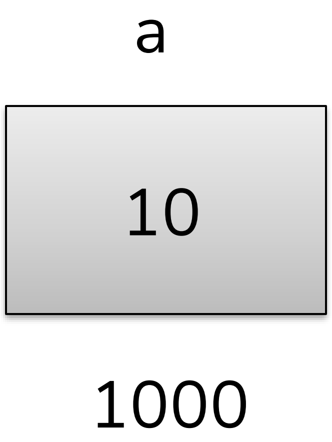
- First character should be alphabet or underscore
- Digits 0 – 9 are allowed
- Variable name should not be a keyword

Variable Declaration :

```
int a;
```

Variable Initialization :

```
int a = 10;
```



int :

4 bytes

Depends on the Compiler

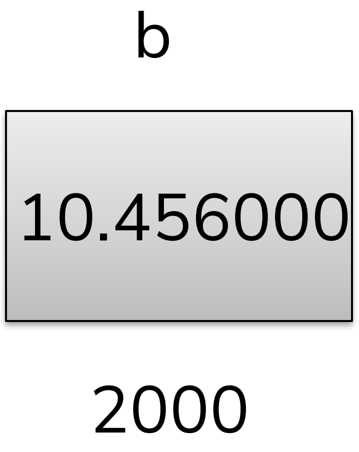
Can take both positive and negative integers

Variable Declaration :

```
float b;
```

Variable Initialization :

```
float b = 10.456;
```



float :

4 bytes

Precision upto 6 decimal places

Can take both positive and negative integers

Different ways of variable initialization

- int a = 10;
- int a = (10);
- int a(10);
- int a = {10};
- int a{10};

- float a = 123.45f;
  - ✓ here a = 123.45
- float a = 123e2f;
  - ✓ here a = 12300
- float a = 123e-2f;
  - ✓ here a = 1.23

Float

- 4 bytes
- Precision 6 decimal places
- Can take both positive and negative Integer values

Double

- 8 bytes
- Precision 15 decimal places
- Can take both positive and negative Integer values

Variable Declaration :

```
char a = ‘S’; // alphabets
char a = ‘?’; // special symbols
char a = ‘3’; // numeric
```

char :

1 byte

S, ?, 3 – anything in single quotes is stored as a character

Decimal	Char
97	a
98	b
99	c
100	d
...	..
...	...
122	z

Decimal	Char
65	A
66	B
67	C
68	D
...	...
...	
90	Z

Decimal	Char
48	0
49	1
50	2
51	3
...	...
...	...
57	9

Scope of Variables

- Local variable
- Global variable

Local variable

- Defined within a function or block
- Anything between ‘{‘ and ‘}’ is said to inside a block

```
#include<iostream>
using namespace std;
void function()
{
    int age=18;
}
int main()
{
    cout<<"Age is: "<<age;
    return 0;
}
```

Error

Local variable

```
#include<iostream>
using namespace std;

void function()
{
    int age=18;
    cout << age;
}

int main()
{
    cout<<"Age is: ";
    function();
    return 0;
}
```

Correct

Global variable

- Can be used throughout the lifetime of the program.
- Declared outside of all the functions and blocks, at the top of the program.

```
#include<iostream>
using namespace std;

int global = 5;

int main()
{
      cout << global;
}
```

Correct

Global variable

```
#include<iostream>
using namespace std;

int global = 5;

void display()
{
    cout<<global<<endl;
}

int main()
{
    display();
    global = 10;
    display();
}
```

Output :

```
5
10
```

How to access local and global variable?

```
#include<iostream>
using namespace std;

int x;

int main()
{
  int x = 10;
  cout << "Value of global x is " << ::x;
  cout<< "\nValue of local x is " << x;
  return 0;
}
```

Output :

```
Value of global x is 0
Value of local x is 10
```

Predict the output

```
#include<iostream>
using namespace std;

int x = 5;

int main()
{
  int x = 10;
  cout << x;
  return 0;
}
```

Output :

10

Predict the output:

```
#include <iostream>
using namespace std;

int main()
{
        int data;
        data = 5;
        float data;
        data = 5.0;
        cout << data;
        return 0;
}
```

Error

Can we name two variables by same name?

Namespace

- A feature added in c++ and not present in c.
- Provide a scope of identifiers within own declarative region
- Used to systematize code in logical groups which prevents naming conflict

Explicit namespace qualifier std::

- Use cout from the std namespace is by explicitly using the std:: prefix
- Safest way to use cout, because there’s no ambiquity about which cout we are referencing

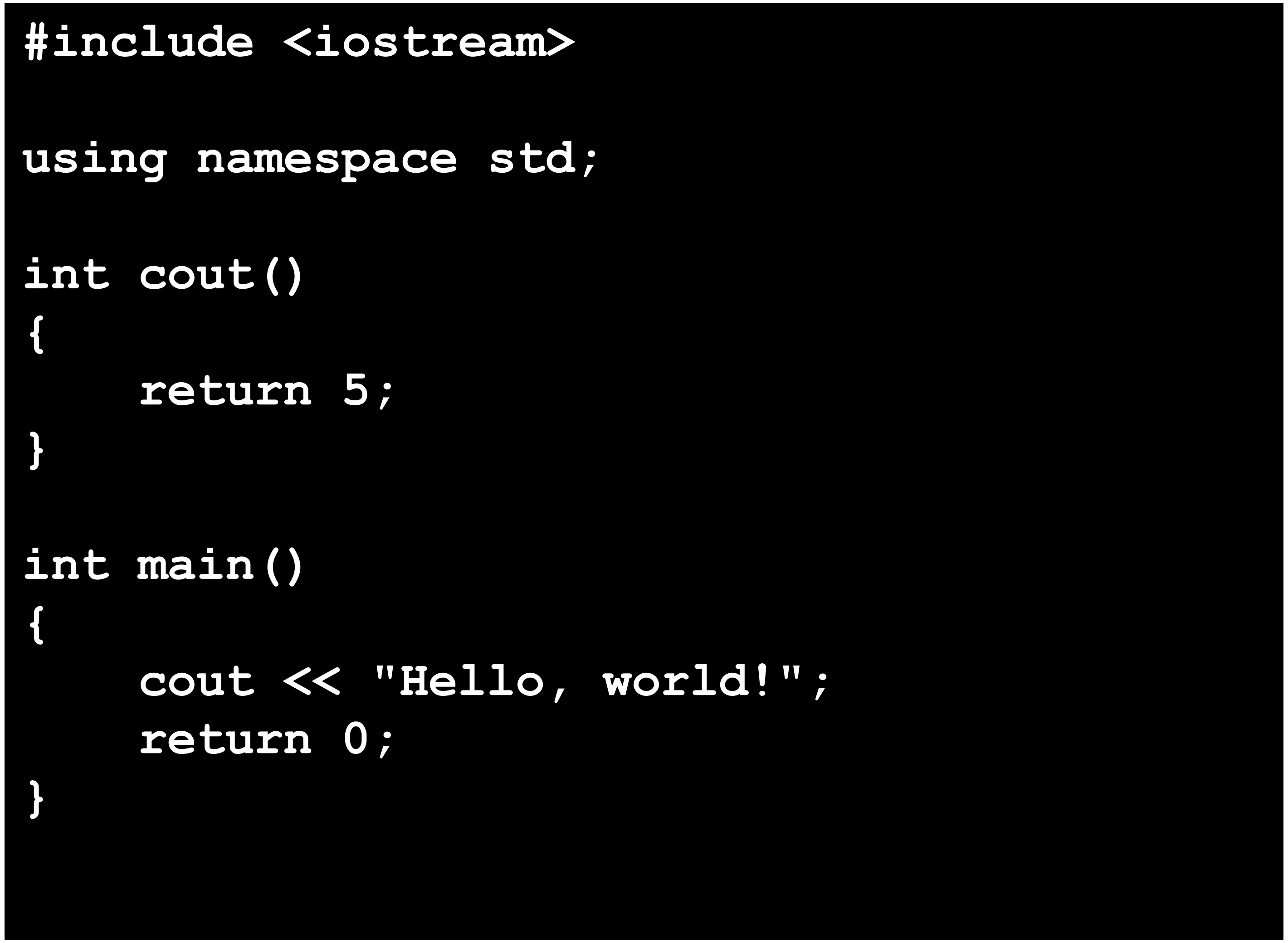
```
#include <iostream>

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

using namespace std

- tells the compiler to check a specified namespace
- when the compiler goes to determine what identifier cout is, it will check both locally and in the std namespace

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello world!";
    return 0;
}
```

using namespace std

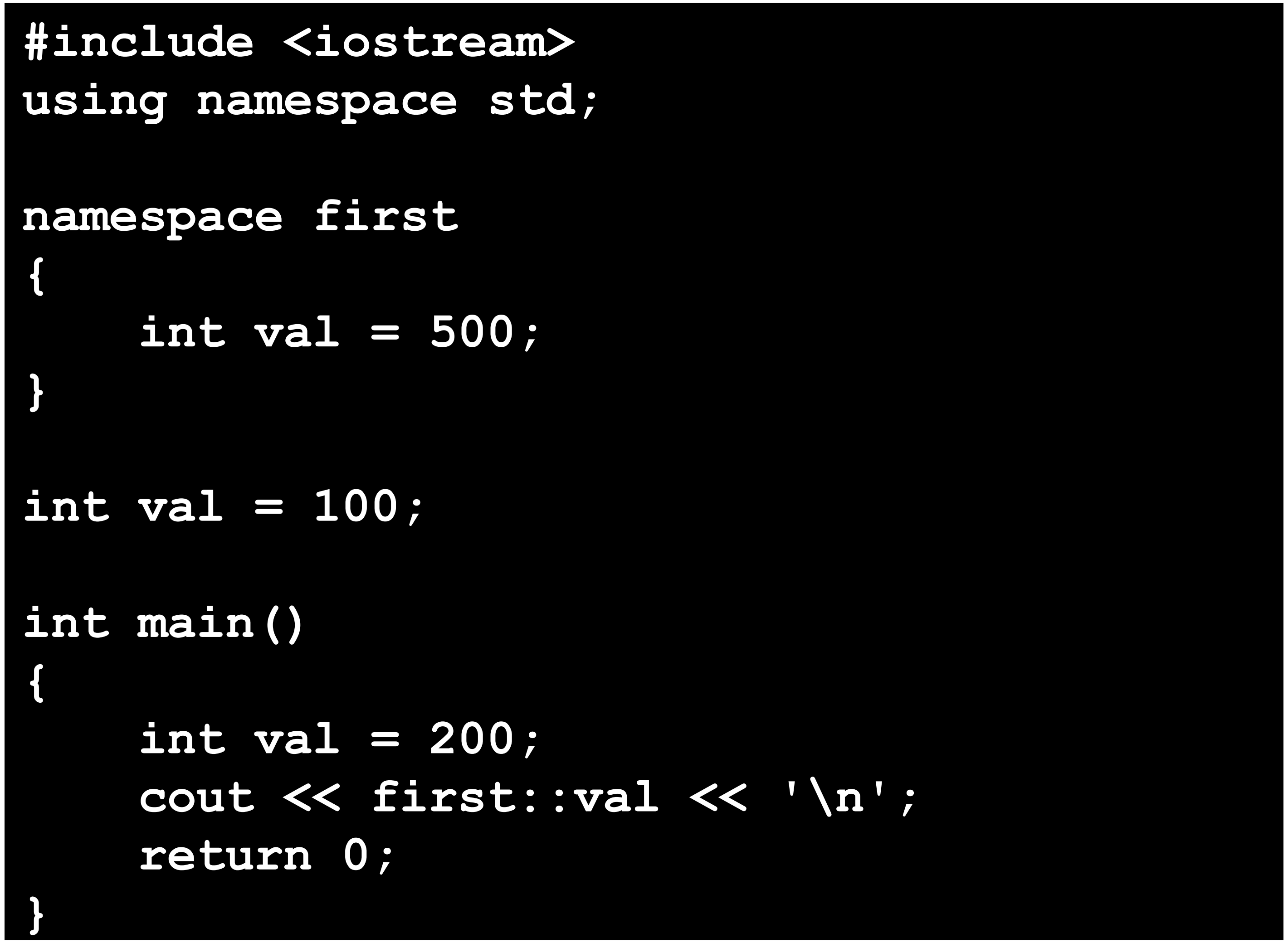


```
#include <iostream>

using namespace std;

int cout()
{
    return 5;
}

int main()
{
    cout << "Hello, world!";
    return 0;
}
```

Output :

Error

Namespace



```cpp
#include <iostream>
using namespace std;

namespace first
{
    int val = 500;
}

int val = 100;

int main()
{
    int val = 200;
    cout << first::val << '\n';
    return 0;
}
```

Output :

```
500
```

Identifiers

Name given to entities such as variables, functions, structures etc.

Example: int sum; float marks; void swap(int a, int b);

sum, marks, swap - Identifiers

int, float - Keywords

Constants

Anything assigned to the variables is called constant

Example: int sum = 10; float marks = 10.456;

10 – integer constant

10.456 – floating point constant

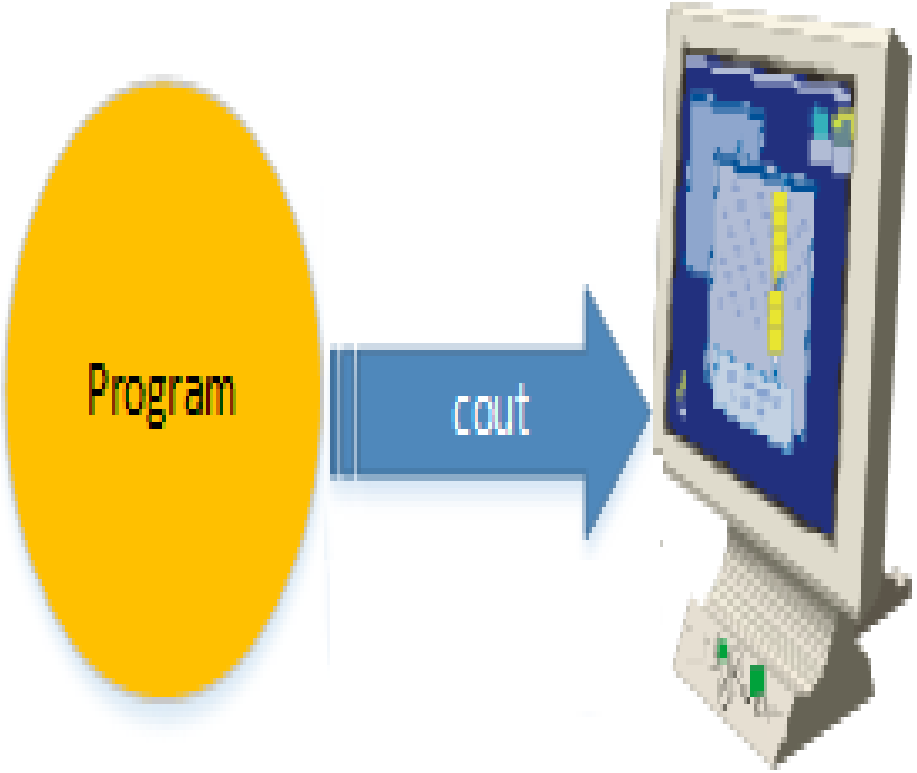
Input & Ouput

I/O occurs in streams, which are sequences of bytes.

- **Input operation.**
- **Output operation.**

Standard Output Stream (cout)

- cout
- << insertion or put to operator
- endl



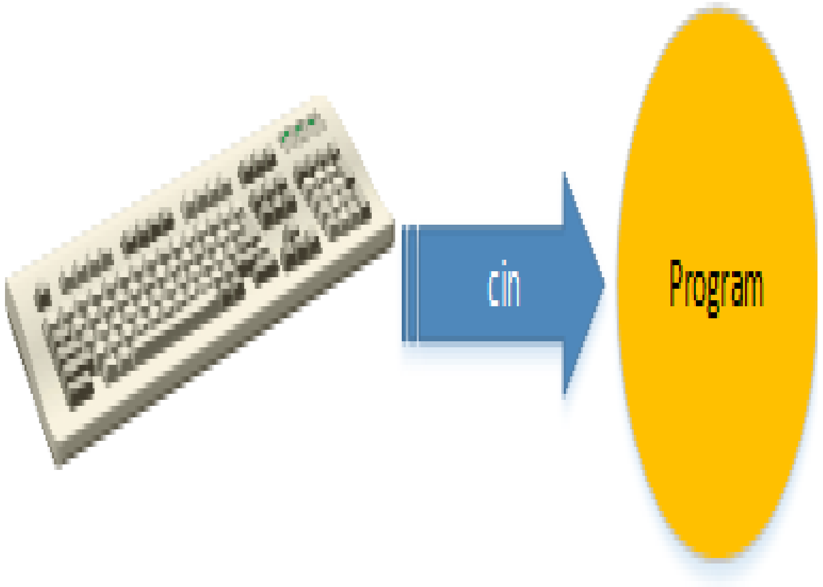
```cpp
#include <iostream>

using namespace std;

int main()
{
    char str[] = "Focus";
    char str1[] = "Academy";
    cout << str << endl << str1;
    return 0;
}
```

Output

```
Focus
Academy
```

Standard Input Stream (cin)

- cin
- >> extraction or get from operator



```cpp
#include <iostream>
using namespace std;

int main()
{
    int value;
    cin >> value;
    cout << value;
    return 0;
}
```

Output

```
5
5
```

Type Conversion

Converting one predefined type into another

- Implicit Type Conversion
- Explicit Type Conversion

Implicit Type Conversion

- Done by compiler on its own
- Takes place in an expression when more than one data type is present
- All the data types of variables are upgraded to data type of variable with largest data type
- **bool -> char -> short int -> int -> unsigned int -> long -> unsigned-> long long -> float -> double -> long double**

```cpp
#include <iostream>
using namespace std;

int main()
{
      int x = 20;
      char y = 'c';
      x = x + y;  // y implicitly converted into int
      float z = x + 1.5;  // x implicitly converted into float
      cout << "x = " << x << endl
           << "y = " << y << endl
           << "z = " << z << endl;

      return 0;
}
```

Output

```
x = 119
y = c
z = 120.5
```

Explicit Type Conversion

User can typecast the result to make it of a particular data type.

- Converting by assignment
- Conversion using cast operator

CONVERTING BY ASSIGNMENT

```cpp
#include <iostream>
using namespace std;

int main()
{
    double x = 7.3;
    int sum = (int)x + 3; // explicit conversion from double to int
    cout << "Sum = " << sum;
    return 0;
}
```

Output

```
Sum = 10
```

CONVERSION USING CAST OPERATOR

```
#include <iostream>
using namespace std;

int main()
{
      float f = 9.5;
      int b = static_cast<int>(f); // using cast operator
      cout << b;
}
```

Output

```
9
```

Operators

- An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulation.
- Operators are used in program to manipulate data and variables.

a + b   a and b -> operands

               + -> operator

C++ Operators are classified into several categories.

Types of Operators

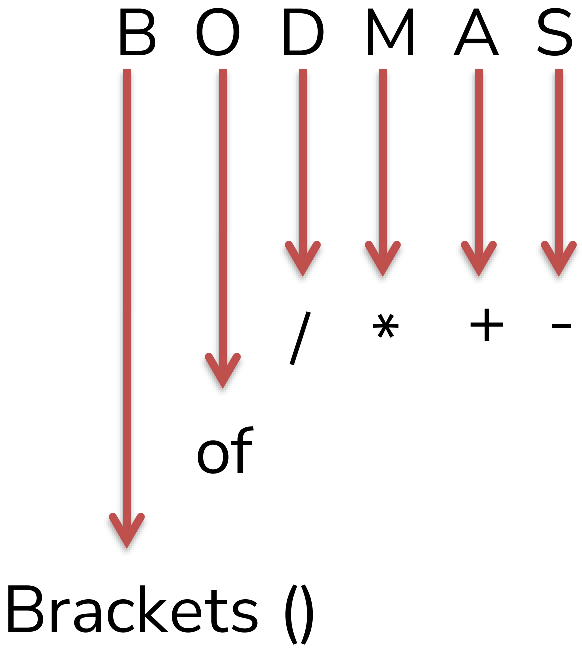
- Arithmetic Operators
- Relational Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Increment and Decrement Operators
- Special Operators

Arithmetic Operators

An arithmetic operator performs mathematical operations such as **addition, subtraction** and **multiplication** on numerical values (constants and variables)

Operators	Example a = 10, b = 5
+	a + b = 15
-	a – b = 5
*	a * b = 50
/	a / b = 2
%	a % b = 0

If you have more than one arithmetic operator in an expression, which operator will execute first?



B O D M A S

/ * + -

of

Brackets ()

What is the difference between % and / operator?

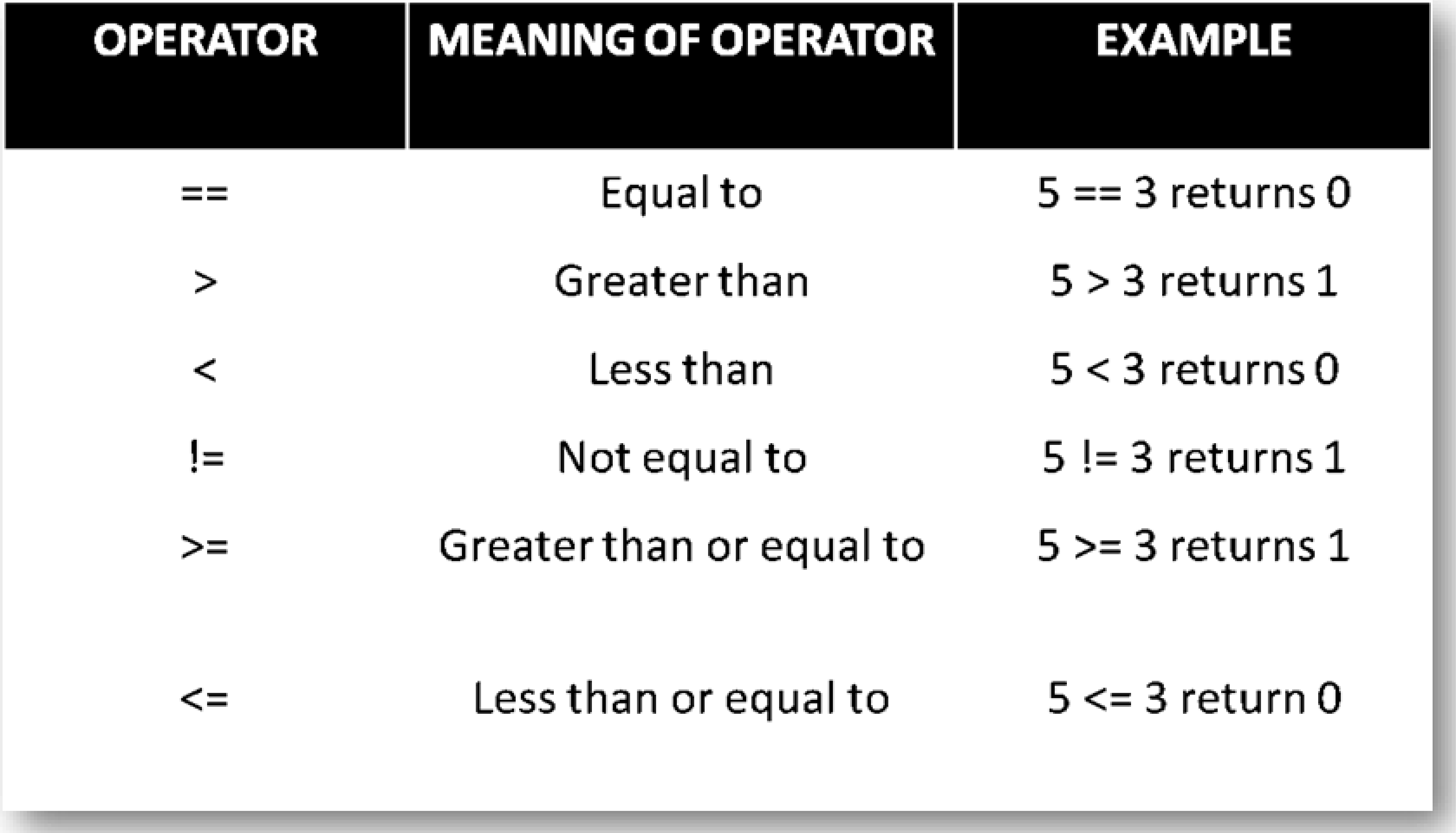
/ -> Quotient

% -> Remainder

Relational Operators

- A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.
- Relational operators are used in decision making and loops.
- It is used to form a condition.

Relational Operators



OPERATOR	MEANING OF OPERATOR	EXAMPLE
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

Logical Operators

- An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false.
- It is used to combine the conditions when you have more than one.

Logical Operators

OPERATOR	MEANING OF OPERATOR	EXMPLE
&&	Logical **AND.** True only if all operands are true.	If c = 5 and d = 2 then, expression ((c = 5) && (d>5)) equals to 0.
||	Logical **OR**. True only if either one operand is true.	If c = 5 and d = 2 then, expression ((c = 5) || (d>5)) equals to 1.
!	Logical **NOT**. True only if the operand is 0.	If c = 5 then expression !(c == 5) equals to 0.

Assignment Operators

Right side value will be assigned to the left side variable

Operator	Example	Meaning
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Bitwise Operators

To perform bit-level operations.

Uses:

- Communication stack
- Compressing(Audio, video, text, image..etc)

  E.g: Zip file

Bitwise Operators

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

```cpp
// Program
#include <iostream>
using namespace std;
int main()
{
      int x = 11;
      int y = 7;
      int a = 8;
      int b = 1;
      cout << x | y ;
      cout << x & y ;
      cout << x ^ y ;
      cout << (a << b);
      cout << (a >> b);
      return 0;
}
```

comment

```
    x = 1 0 1 1

    y = 0 1 1 1

x | y = 1 1 1 1
```

```cpp
// Program
#include <iostream>
using namespace std;
int main()
{
        int x = 11;
        int y = 7;
        int a = 8;
        int b = 1;
        cout << x | y ;
        cout << x & y ;
        cout << x ^ y ;
        cout << (a << b);
        cout << (a >> b);
        return 0;
}
```

comment

```
    x = 1 0 1 1

    y = 0 1 1 1

x & y = 0 0 1 1
```

```
// Program
#include <iostream>
using namespace std;
int main()
{
        int x = 11;
        int y = 7;
        int a = 8;
        int b = 1;
        cout << x | y ;
        cout << x & y ;
        cout << x ^ y ;
        cout << (a << b);
        cout << (a >> b);
        return 0;
}
```

comment

```
    x = 1 0 1 1
    y = 0 1 1 1
x ^ y = 1 1 0 0
```

```
// Program
#include <iostream>
using namespace std;
int main()
{
      int x = 11;
      int y = 7;
      int a = 8;
      int b = 1;
      cout << x | y ;
      cout << x & y ;
      cout << x ^ y ;
      cout << (a << b);
      cout << (a >> b);
      return 0;
}
```

comment

Output:

```
      16
```

How?

```
// Program
#include <iostream>
using namespace std;
int main()
{
        int x = 11;
        int y = 7;
        int a = 8;
        int b = 1;
        cout << x | y ;
        cout << x & y ;
        cout << x ^ y ;
        cout << (a << b);
        cout << (a >> b);
        return 0;
}
```

comment

Output:

```
      4
```

How?

Increment and Decrement Operators

++	Increment the value by 1.
--	Decrement the value by 1.

What is the difference between **++a** and **a++**?

++a

- Pre – increment
- First increment by 1 then, it returns the value

a++

- Post - increment
- First return the original value then, it is incremented by 1.

Similarly, --a and a--

Special Operators

1) Sizeof()
2) &
3) *
4) Ternary(? :)

Special Operators

1) sizeof() operator will returns the number of memory bytes allocated for the data (constant, variables, array, structure etc).

```cpp
// Program
#include <iostream>
using namespace std;

int main()
{
    int a;
    float b;
    double c;
    char d;
    cout << sizeof(a));
    cout << sizeof(b));
    cout << sizeof(c));
    cout << sizeof(d));
    return 0;
}
```

Special Operators

2) & is used to get the address of the variable

3) * is used to get the value of the variable pointed by the pointer

Special Operators

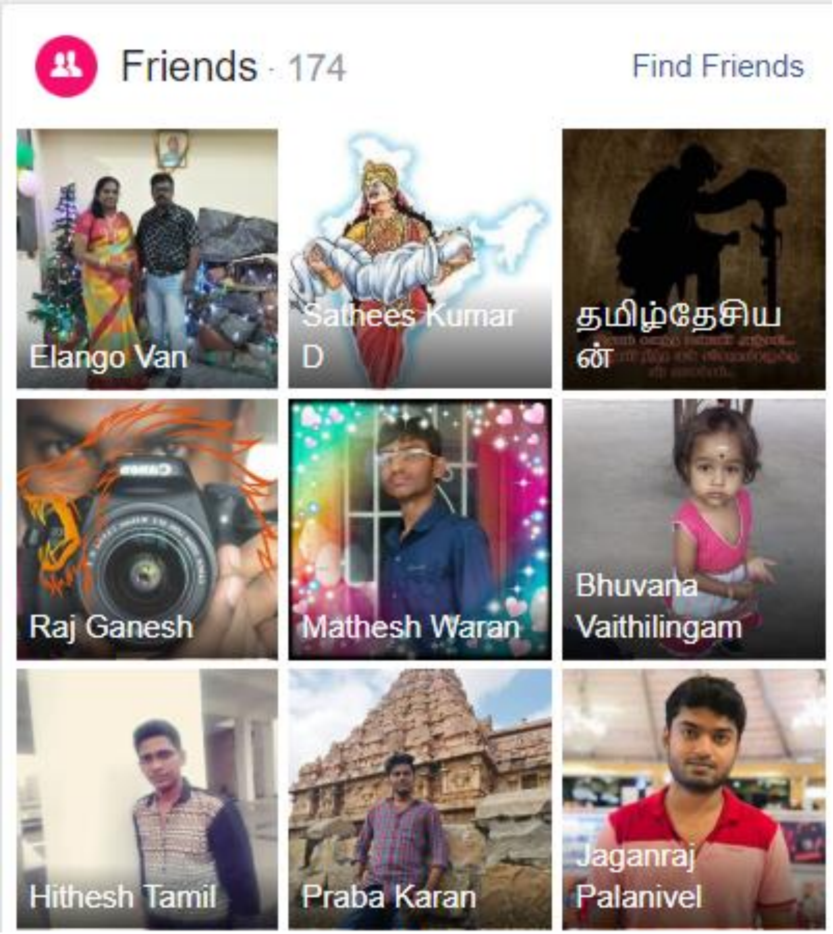
4) Ternary Operator is also known as conditional operator. It works on three operands.

Syntax:

```
    condition ? (statement1) :  (statement2);
```

E.g: 10 < 20 ? printf(" True ") : printf(" False ");

Operators Application



Friends · 174 Find Friends

Elango Van

Sathees Kumar D

தமிழ்தேசியன்

Raj Ganesh

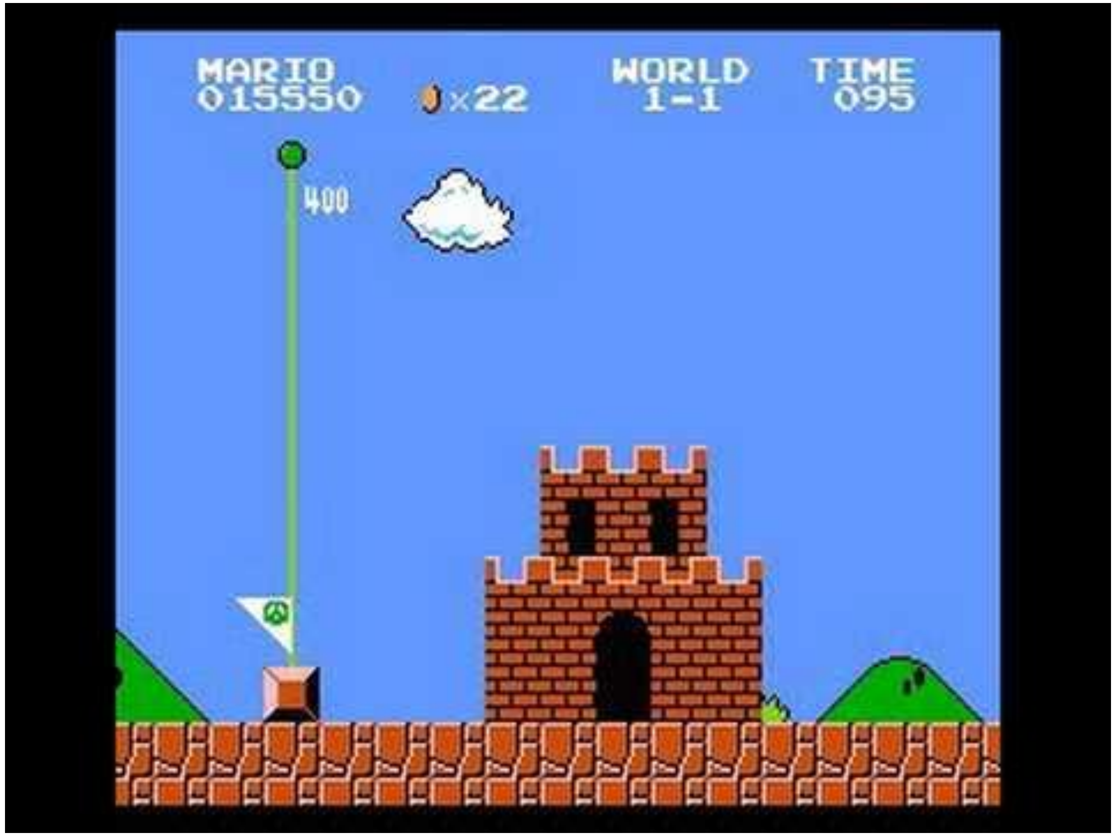
Mathesh Waran

Bhuvana Vaithilingam

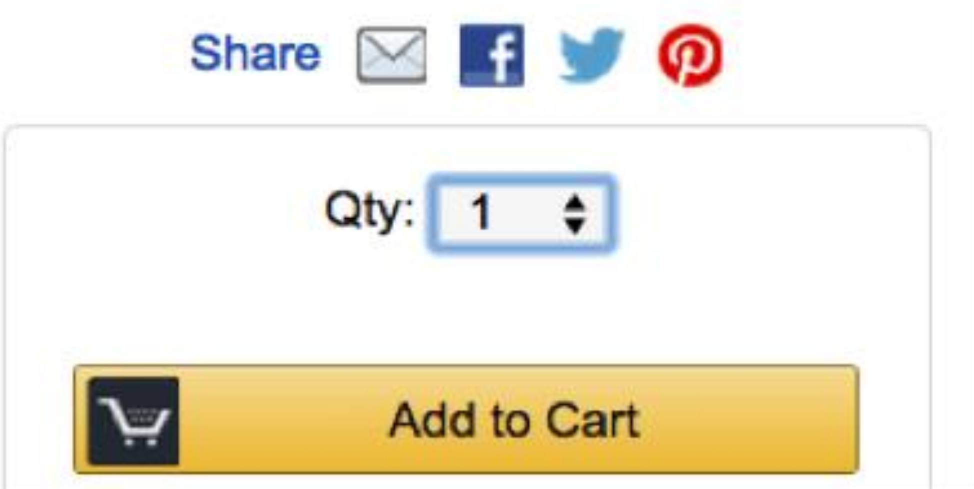
Hithesh Tamil

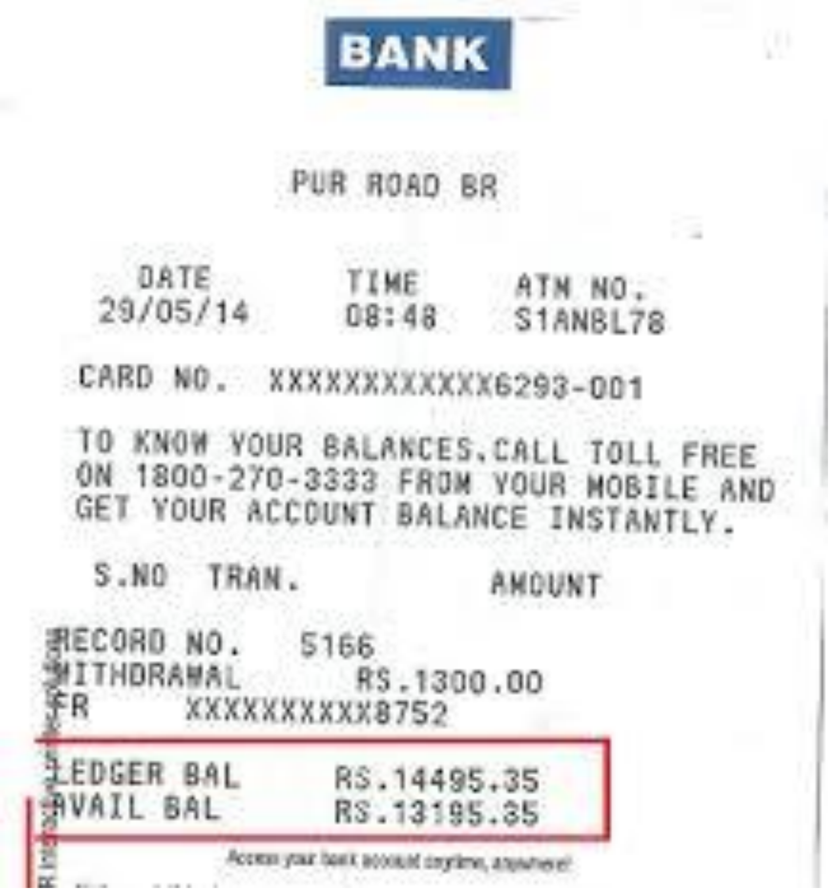
Praba Karan

Jaganraj Palanivel



Operators Application

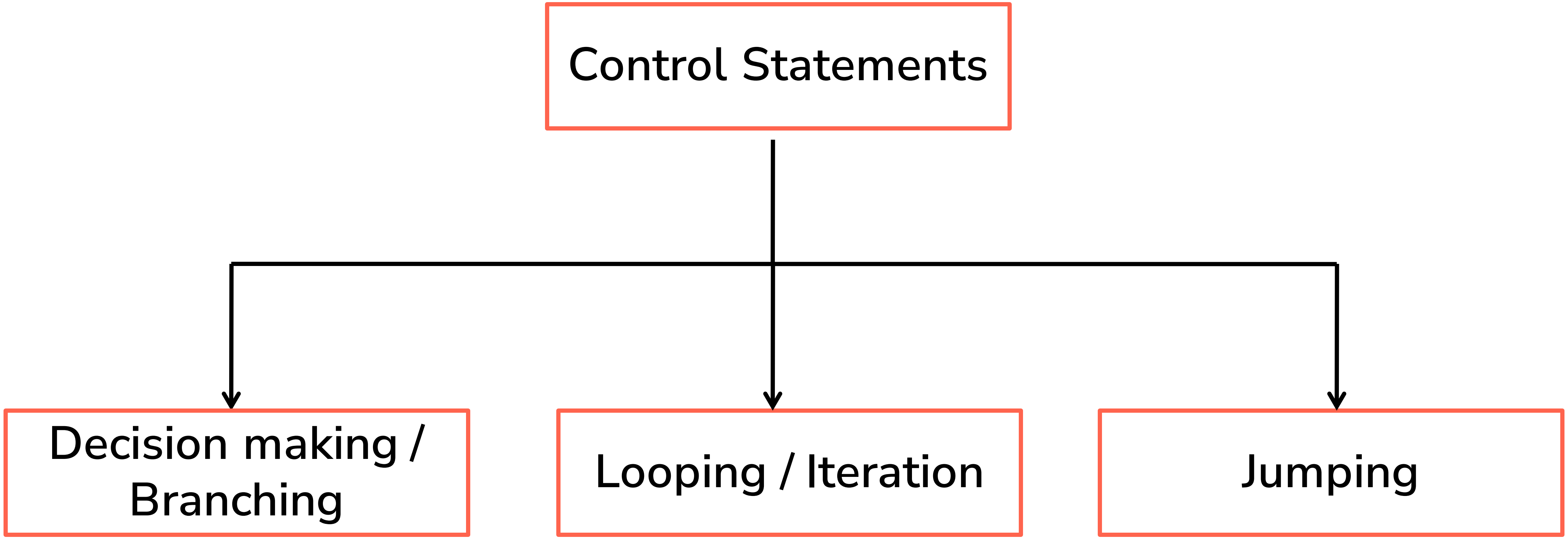




```
BANK

PUR ROAD BR

   DATE          TIME       ATM NO.
 29/05/14       08:48      S1ANBL78

CARD NO.  XXXXXXXXXXXX6293-001

TO KNOW YOUR BALANCES.CALL TOLL FREE
ON 1800-270-3333 FROM YOUR MOBILE AND
GET YOUR ACCOUNT BALANCE INSTANTLY.

 S.NO  TRAN.              AMOUNT

RECORD NO.   5166
WITHDRAWAL         RS.1300.00
FR     XXXXXXXXXX8752

LEDGER BAL        RS.14495.35
AVAIL BAL         RS.13195.35
```

Access your bank account anytime, anywhere!

Control Structures



Branching

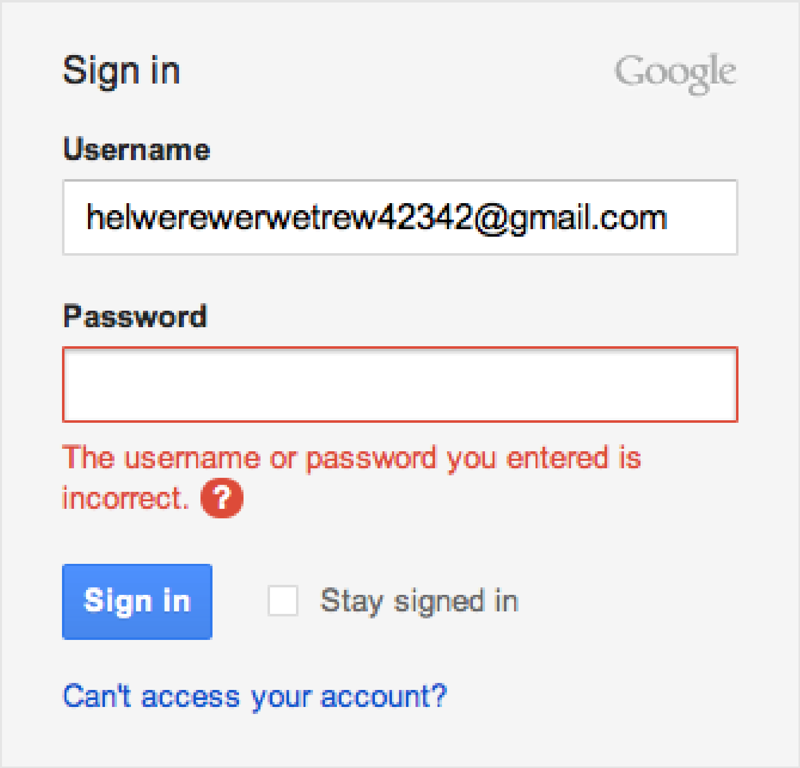
- **Decides what actions to take**

Looping

- **Decides how many times to take a certain action.**

Decision Making / Branching

Where we are using decision making?



Sign in

Username

helwerewerwetrew42342@gmail.com

Password

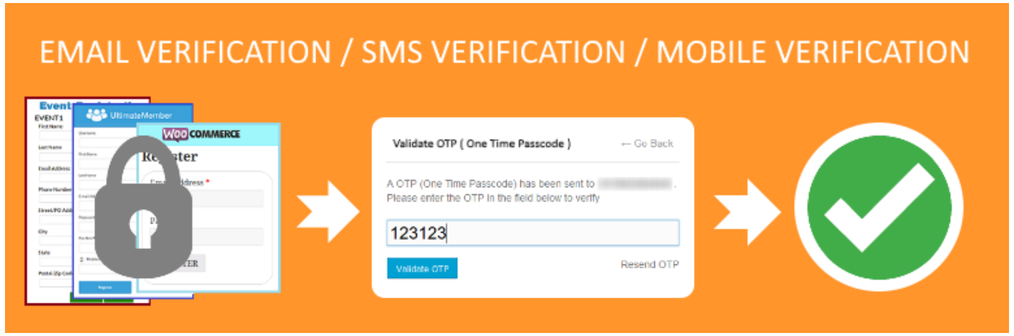
The username or password you entered is incorrect. ?

Sign in

□ Stay signed in

Can't access your account?

Where we are using decision making?



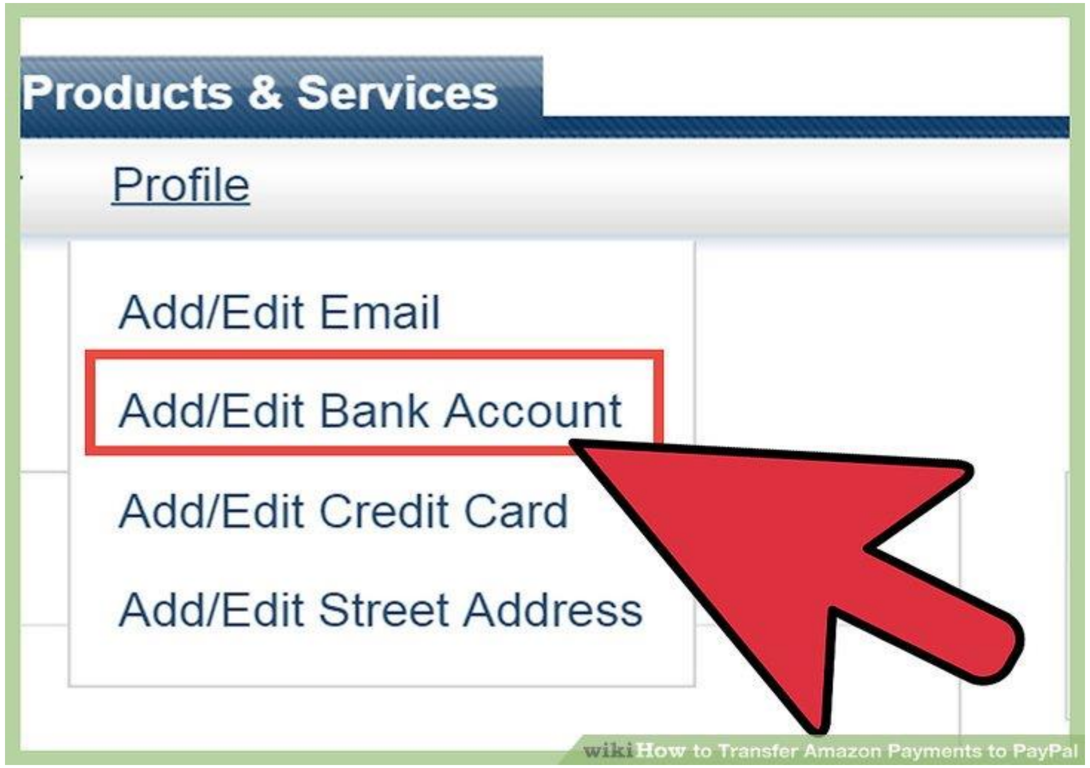
Validate OTP (One Time Passcode) ← Go Back

A OTP (One Time Passcode) has been sent to [redacted] .
Please enter the OTP in the field below to verify

123123

Validate OTP Resend OTP

Where we are using decision making?



Products & Services

Profile

Add/Edit Email

Add/Edit Bank Account

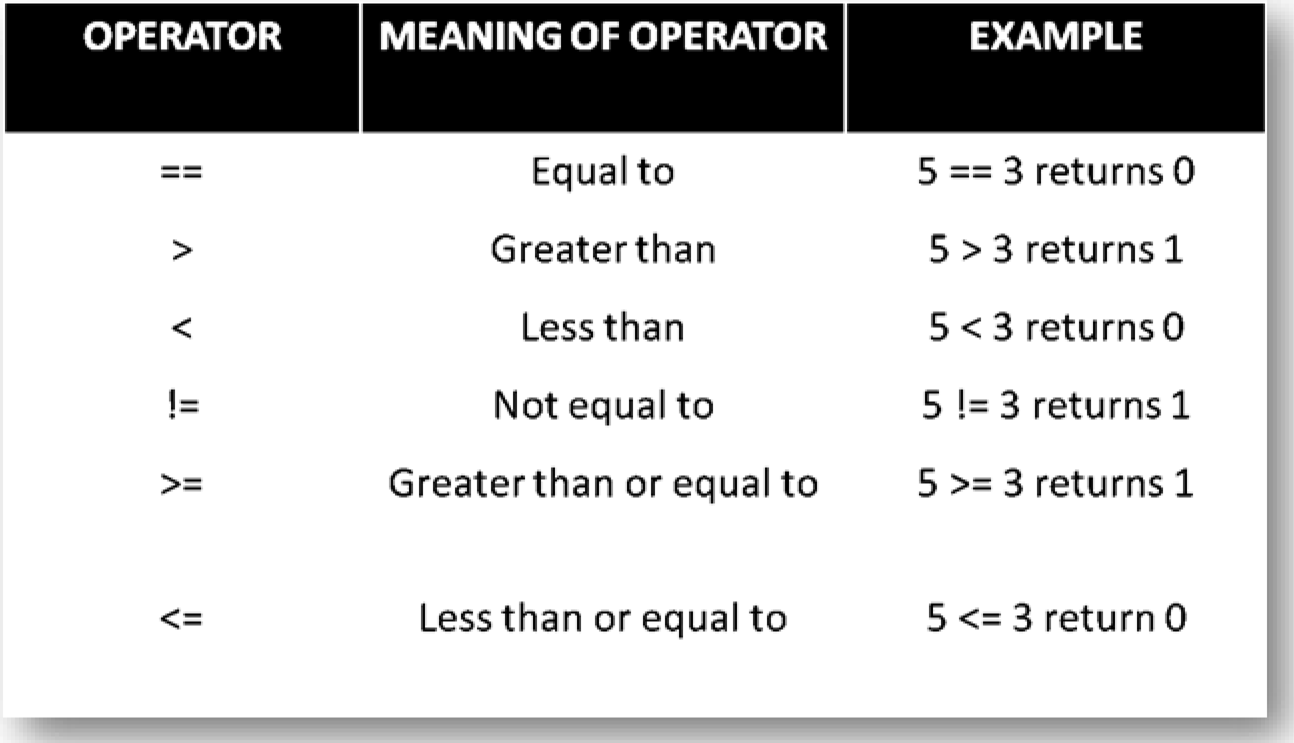
Add/Edit Credit Card

Add/Edit Street Address

wikiHow to Transfer Amazon Payments to PayPal

Conditions

Conditions are formed by relational operators.



OPERATOR	MEANING OF OPERATOR	EXAMPLE
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

Types

1. If – Else
2. Cascaded (If – Else – If)
3. Nested If
4. Switch Case

Who is Taller ?

- How will you find it ?
- By comparing
- How to do that in programming ?
- Using Relational Operators

```
// Pseudocode
If(height of person 1 > height of person 2)
{
      Print(“Person 1 is taller”)
}
Else
{
      Print(“Person 2 is taller”)
}
```

```
// Pseudocode
If(height of person 1 < height of person 2)
{
        Print(“Person 2 is taller”)
}
Else
{
        Print(“Person 1 is taller”)
}
```

If- Else Statement

Syntax:

```
if(condition) {
     //code to be executed if condition is true
}
else {
     //code to be executed if condition is false
}
```

```cpp
// Code
#include<iostream>
using namespace std;
int main()
{
    int h1,h2;
    cin >> h1 >> h2
    if(h1 > h2)
        cout << ”Person 1 is taller";
    else
        cout << "Person 2 is taller“;
    return 0;
}
```

Question 1

Which one of the following condition(s) has to be satisfied to check if a person is eligible to donate blood?

A) Age > 18

B) Weight > 50

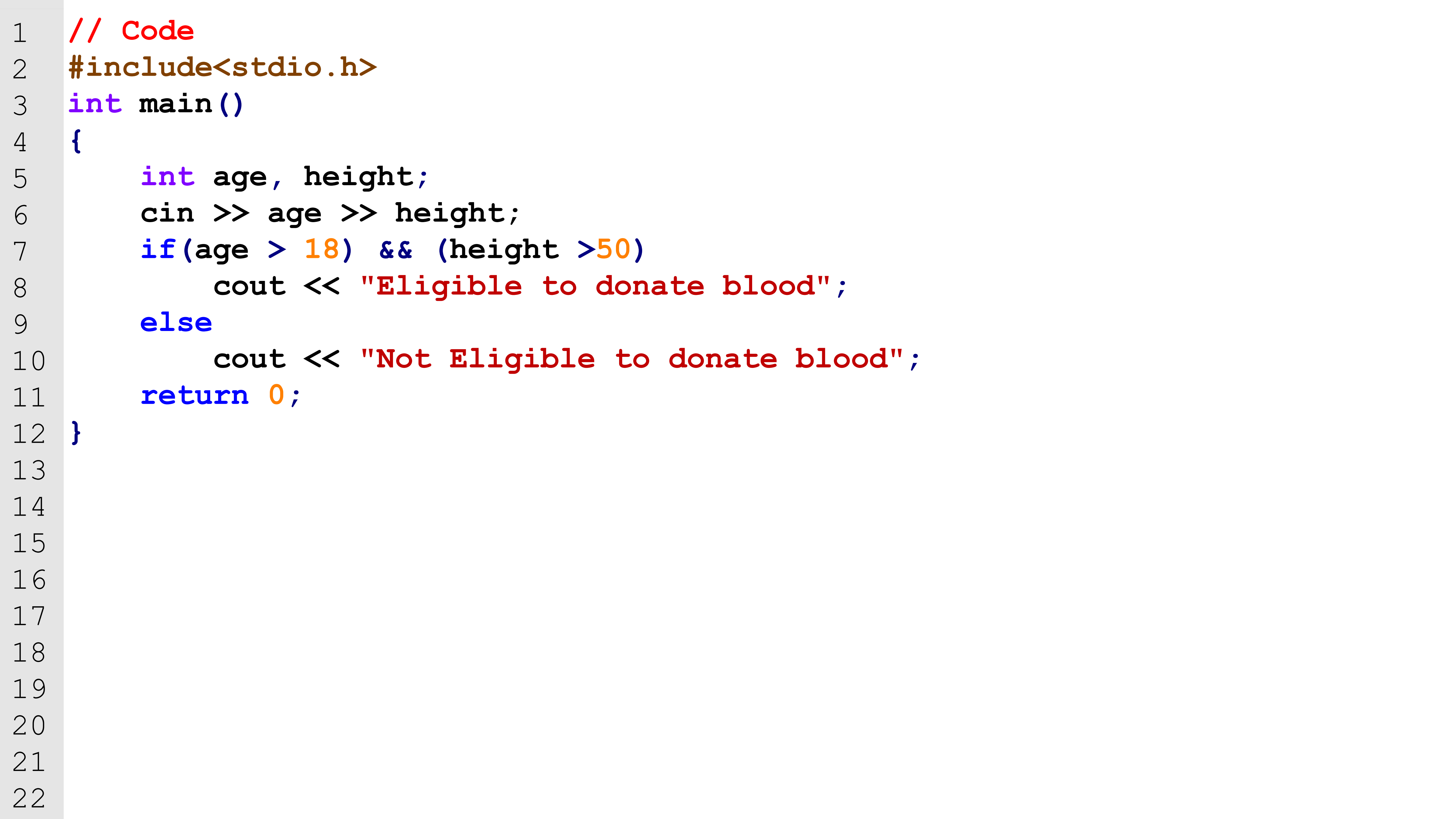
C) Both A and B

D) None of the Mentioned

Logical Operators

Operators	**Meaning**
&&	AND
||	OR
!	NOT

```
// How to use Logical operators in the program
// Pseudocode
If(age of person > 18) && (weight of person  >  50 kg)
{
      Print(“Eligible to donate blood”)
}
Else
{
      Print(“Not eligible”)
}
```



```
// Code
#include<stdio.h>
int main()
{
    int age, height;
    cin >> age >> height;
    if(age > 18) && (height >50)
        cout << "Eligible to donate blood";
    else
        cout << "Not Eligible to donate blood";
    return 0;
}
```

Who is taller ?

- How will you find it ?
- By Comparing
- How to do that in programming ?

Cascaded (if - else if)

Syntax:

```
if(condition1)  {
      //code to be executed if condition1 is true
}
else if(condition2) {
      //code to be executed if condition2 is true
}
else if(condition3) {
      //code to be executed if condition3 is true
}
else{
      //code to be executed if all the conditions are false
}
```

Decision Making in PUBG

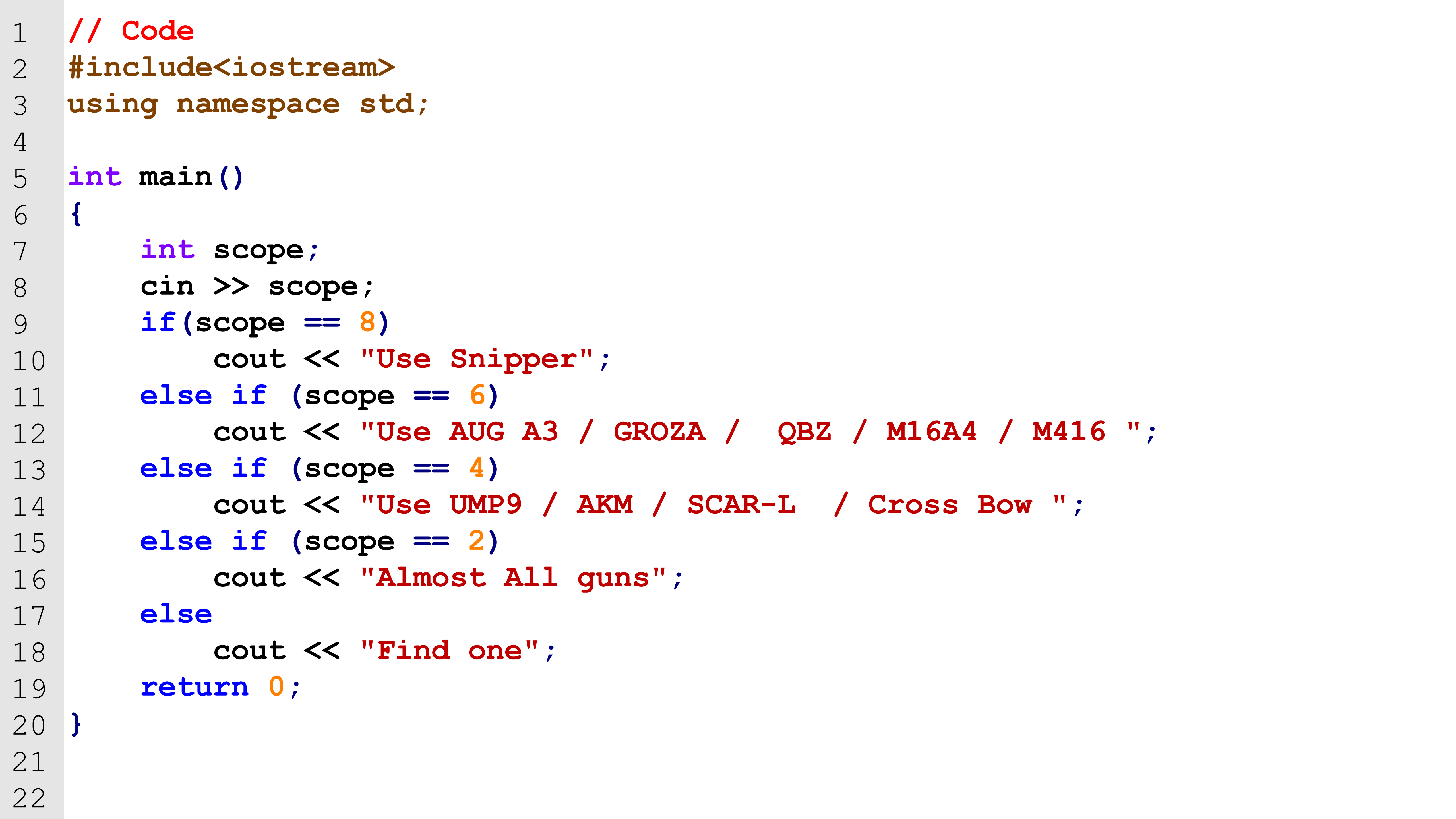
PUBG players are not going to get a Chicken Dinner anytime soon without the ability to aim at targets and take them down with relative ease. So to aim the target they must use scope. It can take hundreds of rounds before you become more comfortable with all the weapons on offer and start landing your shots, but we're here to help speed that process up.

Decision Making in PUBG

Conditions:

- If you have 8x scope, Use snipper gun.
- If you have 6X scope, Use AUG A3, GROZA, QBZ, M16A4, M416 .
- If you have 4x Scope, Use UMP9, AKM, SCAR-L, Cross Bow .
- If you have 2x Scope, almost all guns.
- If you don’t have scope, find one.

```
// Pseudocode
if(scope == 8)
{
        Print(“Use Snipper”);
}
else if(scope == 6)
{
        Print(“Use AUG A3 / GROZA / QBZ / M16A4 / M416 “);
}
else if(scope == 4)
{
        Print(“Use UMP9 / AKM / SCAR-L / Cross Bow );
}
else if(scope == 2)
{
        Print(“Almost all guns”);
}
else
{
        Print(“Find one”);
}
```



```cpp
// Code
#include<iostream>
using namespace std;

int main()
{
    int scope;
    cin >> scope;
    if(scope == 8)
        cout << "Use Snipper";
    else if (scope == 6)
        cout << "Use AUG A3 / GROZA /  QBZ / M16A4 / M416 ";
    else if (scope == 4)
        cout << "Use UMP9 / AKM / SCAR-L  / Cross Bow ";
    else if (scope == 2)
        cout << "Almost All guns";
    else
        cout << "Find one";
    return 0;
}


```

Nested If

Syntax:

```
if (condition1)
{
     // code to be executed if condition1 is true
     if (condition2)
     {
          // code to be executed if condition2 is true
     }
}
```

Bunjee Jumping

Have you tried or seen Bunjee Jumping ? It's a weird experience, Isn't it?. But if someone is very eager to try bunjee jumping, they must satisfy few conditions

Conditions:

1) Minimum Weight must be 40 kgs
2) Maximum Weight must be 110 kgs[If Weight is greater than maximum, extra ropes will be added]
3) Minimum age required is 12 years.

```
// PseudoCode
If(age >= 12)
{
    If(weight >= 40)
    {
         If(weight <= 110)
         {
            Print(“He can Jump”);
         }
         Else{
             Print(“Extra ropes will be added”);
         }
    }
    Else
    {
         Print(“He can’t Jump”);
    }
}
Else
{
    Print(“He can’t Jump”);
}
```

```
#include<iostream>
using namespace std;
int main()
{
    int age, weight;
    cin >> age >> weight;
    if(age >= 12)
    {
        if(weight >= 40)
        {
            if(weight <= 110)
            {
              cout << “He can Jump”;
            }
            else
            {
              cout << “Extra ropes
                     will be added”;
            }
        }


        else
        {
            cout << “He can’t Jump”;
        }
    }
    else
    {
        cout << “He can’t Jump”;
    }
}
```

Switch Case

Syntax:

```
switch(expression){
      case 1:
            //code to be executed;
            break;
      case 2:
            //code to be executed;
            break;
                        ......
      default:
            code to be executed if all cases are not
            matched;
}
```

PUBG

Do you know how many maps are there in PUBG ?

- Erangel [Forest]
- Miramar[Desert]
- Sanhok[Rain Forest]
- Vikendi[Snow Forest]

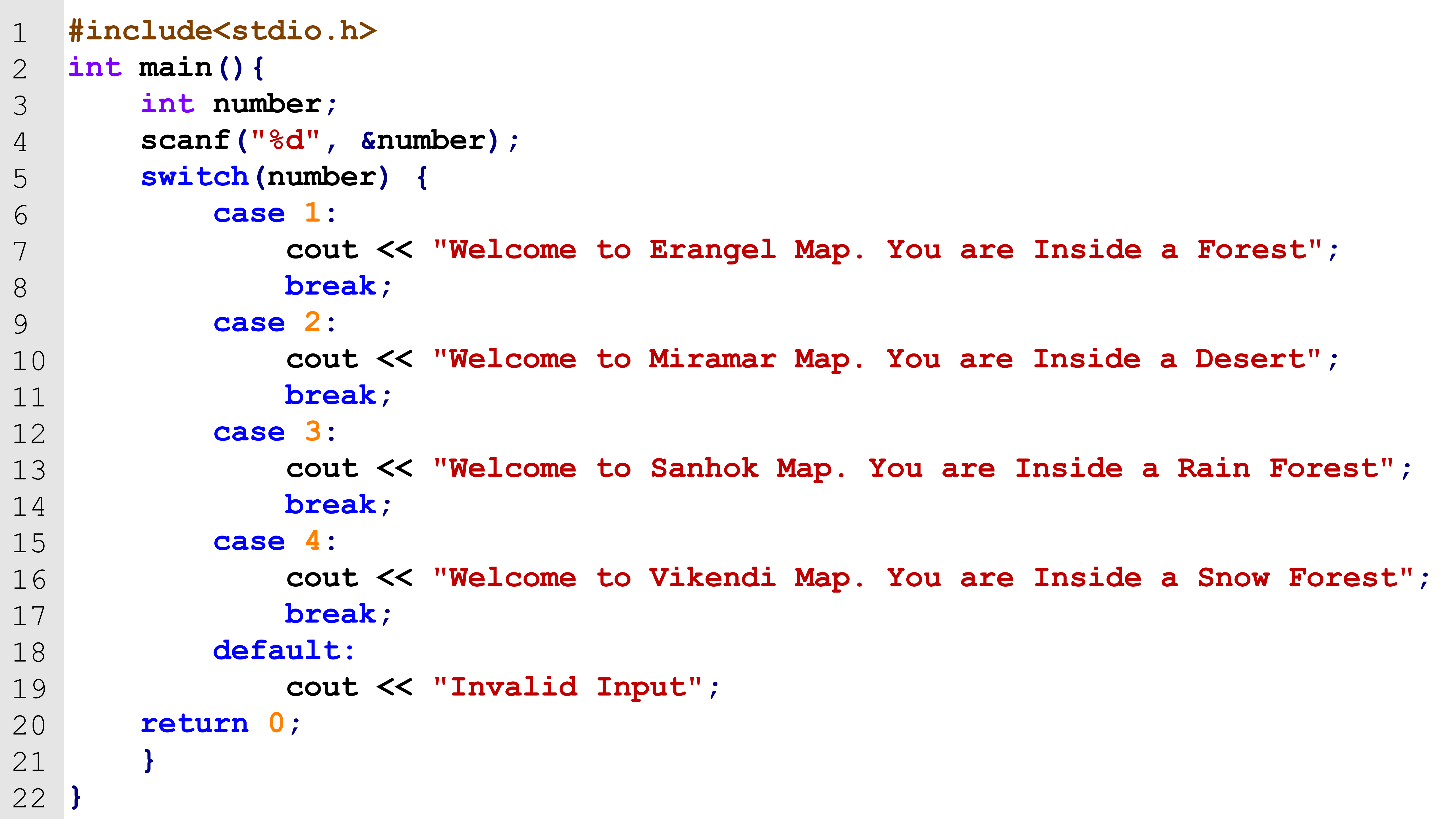
PUBG

When user enters the corresponding number of maps [1, 2, 3, 4].
It must get into that map, show a Welcome message and
displays the type of that map [Forest, Desert, Rain Forest, Snow Forest].

PUBG

Use Switch Case.
Switch case is a multiple-branching statement

```
//Pseudocode
Switch(number)
{
    case 1:
            Print(“Welcome to Erangel Map. You are Inside a Forest”);
            break;
    case 2:
            Print(“Welcome to Miramar Map. You are Inside a Desert”);
            break;
    case 3:
            Print(“Welcome to Sanhok Map. You are Inside a Rain Forest”);
            break;
    case 4:
            Print(“Welcome to Vikendi Map. You are Inside a Snow Forest”);
            break;
    Default:
            Print(“Invalid Input”);
}
```



```
#include<stdio.h>
int main(){
    int number;
    scanf("%d", &number);
    switch(number) {
        case 1:
            cout << "Welcome to Erangel Map. You are Inside a Forest";
            break;
        case 2:
            cout << "Welcome to Miramar Map. You are Inside a Desert";
            break;
        case 3:
            cout << "Welcome to Sanhok Map. You are Inside a Rain Forest";
            break;
        case 4:
            cout << "Welcome to Vikendi Map. You are Inside a Snow Forest";
            break;
        default:
            cout << "Invalid Input";
    return 0;
    }
}
```

Switch case (Fall Through)

Syntax:

```
switch(expression){
      case 1:
      case 2:
      case 3:
            // Code to be executed
            break
      case 4:
      case 5:
      case 6:
            // Code to be executed
            break
 }
```

PUBG

After selecting the map in PUBG, the next steps are common in all the maps. Let’s take a look at the steps.

1. Selecting an area to drop out.
2. Looting weapons and equipment's.
3. Stay out of blue circle.
4. Kill your enemies.

So, for all the four maps, we are going to display these instructions. So How will you do that ?

```
// Pseudocode
Switch(number)
{
      case 1:
      case 2:
      case 3:
      case 4:
             Print(“1.Selecting an area to drop out. \n2.Looting weapons
             and equipments.\n3.Stay out of blue circle.\n4.Kill your
             enemies.”);
             break;
}
```

```
// Code
#include<iostream>
using namespace std;

int main()
{
    int number;
    cin >> number;
    switch(number)
    {
        case 1:
        case 2:
        case 3:
        case 4:
                cout << “1.Selecting an area to drop out.
                       \n2.Looting weapons and equipments.
                       \n3.Stay out of blue circle.
                       \n4.Kill your enemies.”;
                break;
    }
    return 0;
}
```

Question 1

How many times FACE is printed?

```
#include<iostream>
using namespace std;
int main()
{
    int i=0;
    lbl:
    cout<<“FACE";
    i++;
    if(i<5)
    {
      goto lbl;
    }
    return 0;
}
```

Question 1

A) Compilation Error

B) 5 times

C) 4 times

D) 6 times

Question 2

Predict the output:

```
#include<iostream>
using namespace std;
int main()
{
 if(0)
 {
    cout<<"Hi";
 }
 else
 {
    cout<<"Bye";
 }
 return 0;
}
```

Question 2

A) Hi

B) Bye

C) HiBye

D) Compilation Error

Question 3

Predict the output:

```
#include<iostream>
using namespace std;

int main()
{
int x = 5;
if(x++ == 5)
cout<<"Five"<<endl;

else if(++x == 6)
cout<<"Six"<<endl;

return 0;
}
```

Question 3

A) FiveSix

B) Six

C) None

D) Five

Errors

- Illegal operation performed by the user
- Errors are undetected until it is compiled or executed
- Errors should be removed before executing

Types of Errors

- Compiler Errors
- Linker Error
- Runtime Errors
- Logic Errors

Compiler Errors

- Programming languages have rules

- Syntax errors – something wrong with the structure
  - ✓ std :: cout << “Errors << std:: endl;

- Semantic errors – something wrong with the meaning
  - ✓ a + b; when it doesn’t make sense to add a and b

Compiler Warnings

- Do not ignore them!

- The compiler has recognized an issue with your code that could lead to potential problem

- int var;
  std :: cout << var;

- warning: ‘var’ is used uninitialized…

Linker Error

- These error occurs when after compilation we link the different object files with main's object

- These are errors generated when the executable of the program cannot be generated.

- One of the most common linker error is writing Main() instead of main().

Runtime Errors

- Errors that occur when the program is executing

- Some typical runtime errors
  - ✓ Divide by zero
  - ✓ File not found
  - ✓ Out of memory

- Can cause your program to ‘crash’

- Exception handling can help deal with runtime errors

Logic Errors

- Errors or bugs in your code that cause your program to run incorrectly

- Logic errors are mistakes made by the programmer

- Suppose we have a program that determines if a person can vote in an election and you must be 18 years or older to vote.

```
✓ if(age > 18)
    {
             std :: cout << “Yes, you can vote!”;
    }
```

Looping / Iteration

Repeating a set of instruction for a period of time

Looping / Iteration

Repeating a set of instruction for a period of time

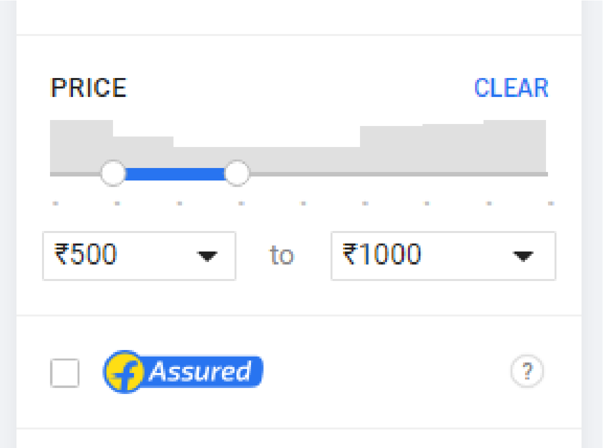
Looping

- Loops are iterative statements
- Block of statements are repeatedly executed as long as condition is true
- Infinite loop
- Finite loop
- Type of loops

Types of Loops

- Pre-tested loop
- Post-tested loop
- Counter controlled loop

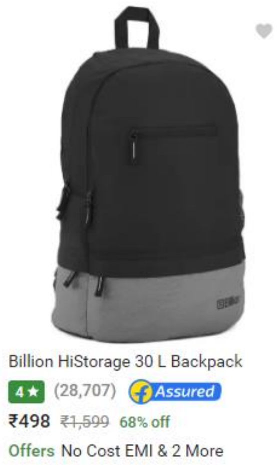
Looping / Iteration



PRICE CLEAR

₹500 to ₹1000

□ Assured ?

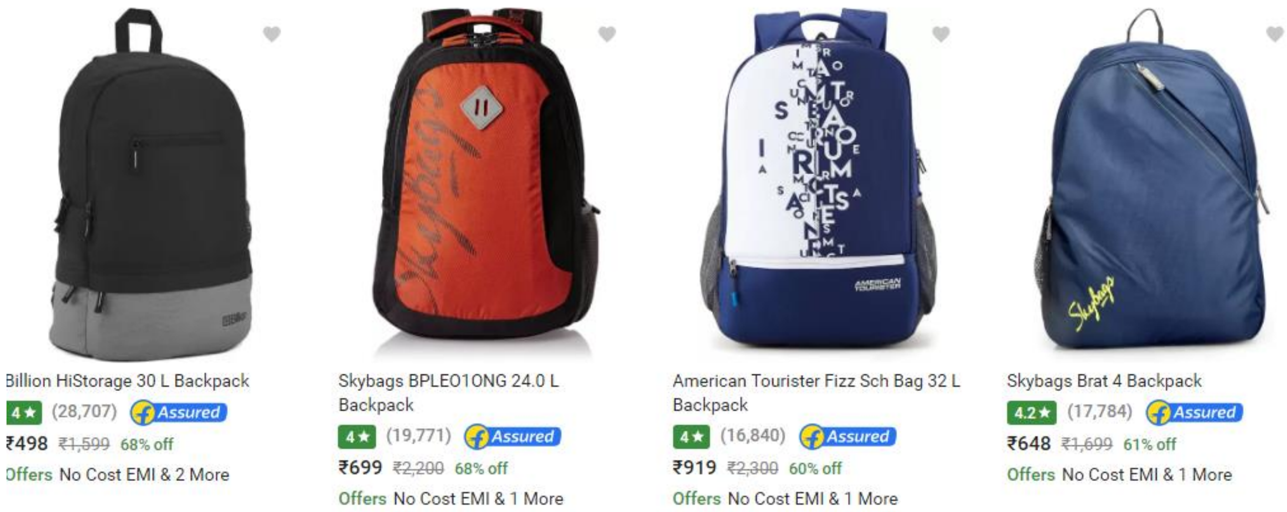


Billion HiStorage 30 L Backpack

4★ (28,707) Assured

₹498 ~~₹1,599~~ 68% off

Offers No Cost EMI & 2 More



Billion HiStorage 30 L Backpack

4★ (28,707) Assured

₹498 ~~₹1,599~~ 68% off

Offers No Cost EMI & 2 More

Skybags BPLEO1ONG 24.0 L Backpack

4★ (19,771) Assured

₹699 ~~₹2,200~~ 68% off

Offers No Cost EMI & 1 More

American Tourister Fizz Sch Bag 32 L Backpack

4★ (16,840) Assured

₹919 ~~₹2,300~~ 60% off

Offers No Cost EMI & 1 More

Skybags Brat 4 Backpack

4.2★ (17,784) Assured

₹648 ~~₹1,699~~ 61% off

Offers No Cost EMI & 1 More

```
// Pseudocode
Repeat If(Price >= 500 && Price <= 1000)
{
        Display the bag;
}

while(Price >= 500 && Price <= 1000)
{
        Display the bag;
}
```

Question 1

Print 1 to N

Sample Input:

```
10
```

Sample Output:

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

```
// Pseudocode
number = 1
Repeat If(Number <= N)
{
      Print Number;
      Number++;
}
```

While Loop

- Condition is checked first, then loop body will be executed.
- If condition is false, loop body will not be executed.
- Entry controlled loop

While Loop

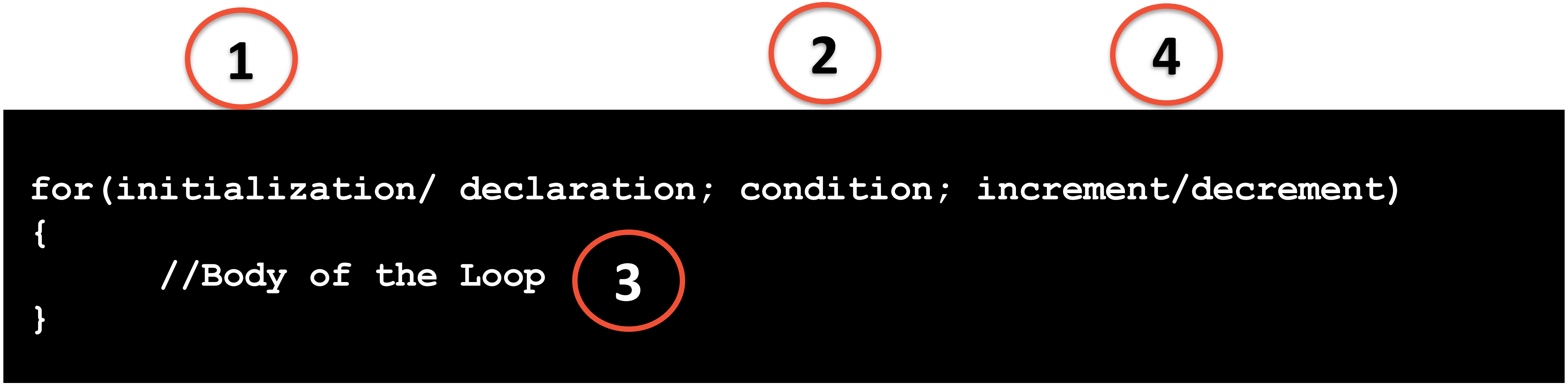
Syntax:

```
while (condition)
{
      //body of the loop
      Increment/ Decrement
}
```

```
// Code
#include<iostream>
using namespace std;

int main()
{
    int N;
    cin >> N;
    int number = 1;
    while(number <= N)
    {
        cout << number;
        number++;
    }
    return 0;
}
```

```
// Pseudocode
number = 1
Repeat If(Number <= N)
{
      Print Number;
      Number++;
}

for(number = 1; number <= N; number++)
{
      Print Number;
}
```

For Loop



```
for(initialization/ declaration; condition; increment/decrement)
{
      //Body of the Loop
}
```

```
// Code to print numbers from 1 to N
include<stdio.h>
int main()
{
    int N;
    cin >> N;
    for(int number = 1; number <= N; number++)
    {
        cout << number;
    }
    return 0;
}
```

Question 2

Count the Number of digits

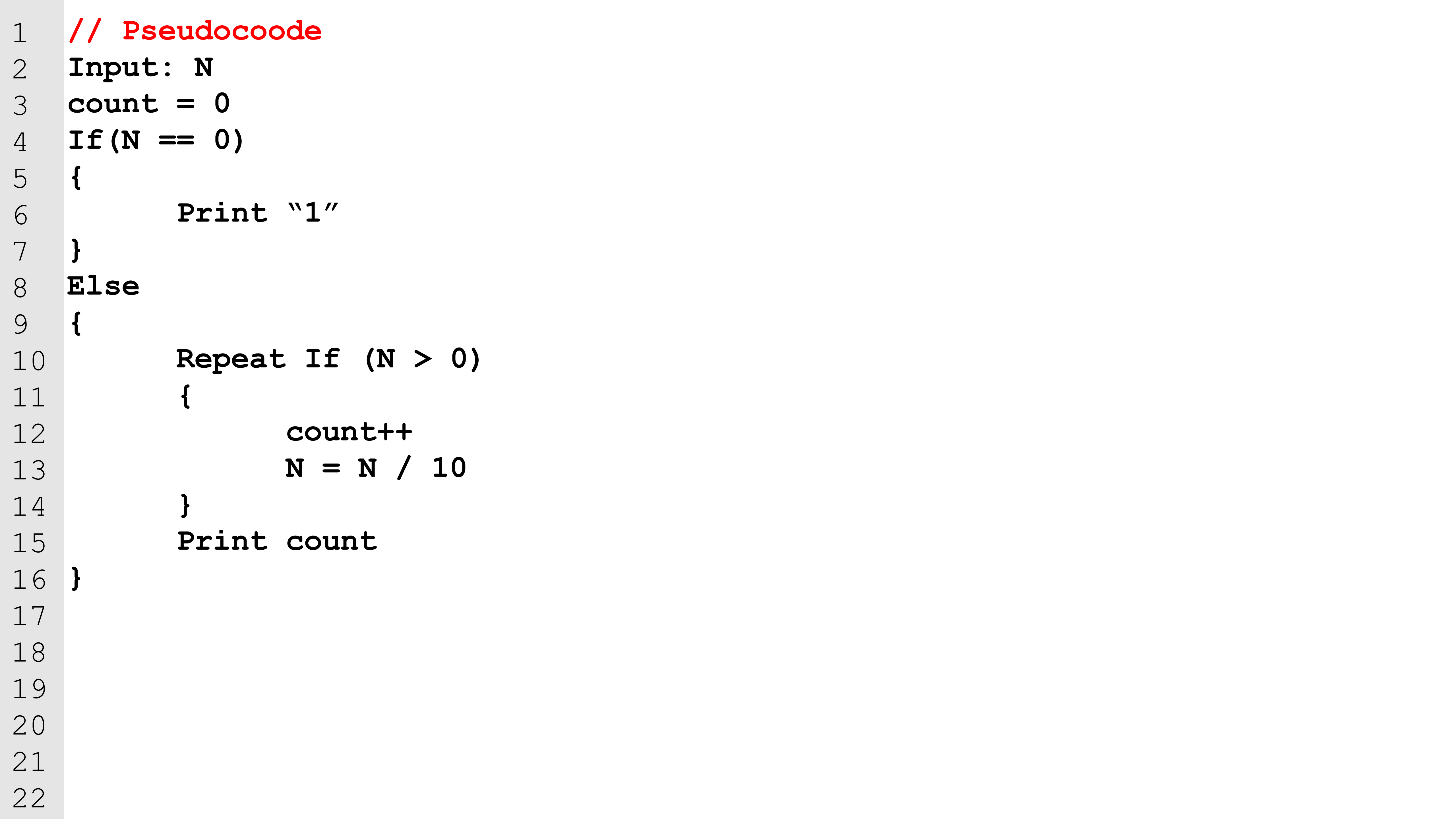
Sample Input:

44

Sample Output:

2

```
// Pseudocoode
Input: N
count = 0
Repeat If (N > 0)
{
      count++
      N = N / 10
}
Print count
```



```
// Pseudocoode
Input: N
count = 0
If(N == 0)
{
      Print "1"
}
Else
{
      Repeat If (N > 0)
      {
            count++
            N = N / 10
      }
      Print count
}
```

```
// Pseudocoode
Input: N
count = 0
Do
{
      count++
      N = N / 10
}while(N > 0);
Print count
```

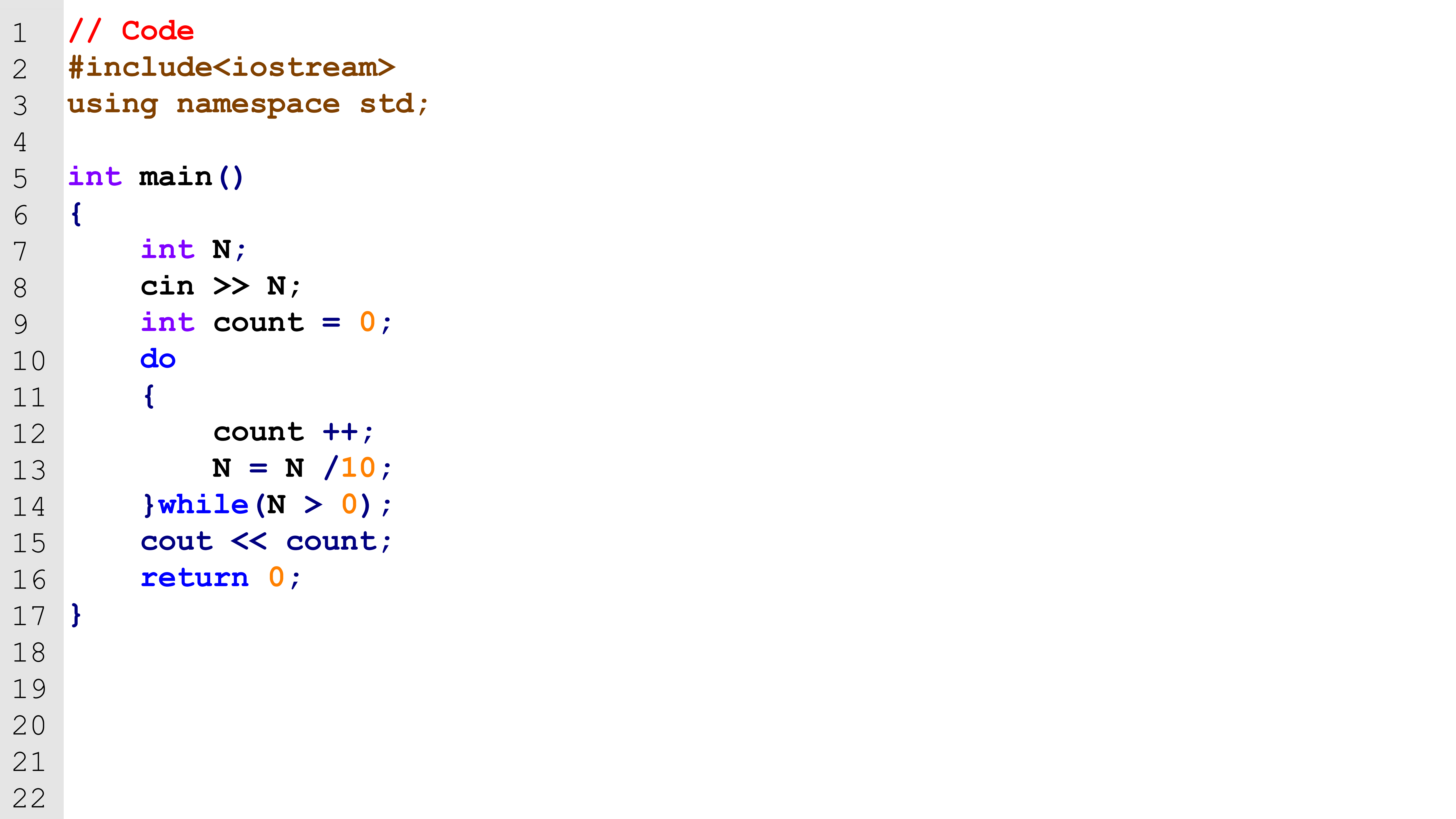
Do While Loop

- Loop body will be executed first, then condition is checked
- If condition is false, loop body will be executed at least once
- Exit controlled loop

Do While Loop

Syntax:

```
Do
{
      //statement
}while(condition);
```



```
// Code
#include<iostream>
using namespace std;

int main()
{
    int N;
    cin >> N;
    int count = 0;
    do
    {
        count ++;
        N = N /10;
    }while(N > 0);
    cout << count;
    return 0;
}
```

break

- No further statements in the body of the loop are executed
- Loop is immediately terminated
- Control immediately goes to the statement following the loop construct

```cpp
#include<iostream>
using namespace std;

int main()
{
      for(int i = 1; i <= 10; i++)
      {
            if(i == 5)
            {
                  break;
            }
            cout << i;
      }
      return 0;
}
```

Output

```
1234
```

continue

- No further statements in the body of loop are executed
- Control immediately goes directly to the beginning of the loop for the next iteration

```cpp
#include<iostream>
using namespace std;

int main()
{
      for(int i = 1; i <= 10; i++)
      {
            if(i == 5)
            {
                  continue;
            }
            cout << i;
      }
      return 0;
}
```

Output

```
1234678910
```

Range based for loop

```
for(var_type var_name : sequence)
        statement; // can use var_name

for(var_type var_name : sequence)
{
        statements; // can use var_name
}
```

```cpp
#include<iostream>
using namespace std;

int main()
{
      int scores[] {100,90,87};
      for(int score : scores)
      {
            cout << score << endl;
      }
      return 0;
}
```

Output

```
100
90
87
```

```cpp
#include<iostream>
using namespace std;

int main()
{
        int scores[] {100,90,87};
        for(auto score : scores)
        {
                cout << score << endl;
        }
        return 0;
}
```

Output

```
100
90
87
```

```cpp
#include<iostream>
using namespace std;

int main()
{
        string s = "Hello";
        for(char s1 : s)
        {
                cout << s1 << endl;
        }
        return 0;
}
```

Output

```
H
e
l
l
o
```

```cpp
#include<iostream>
using namespace std;

int main()
{
      string s = “ABC";
      for(int s1 : s)
      {
            cout << s1 << endl;
      }
      return 0;
}
```

Output

```
65
66
67
```

Question 1

Printing * horizontally

Sample Input:

N = 4

Sample Output:

```
****
```

```
#include<iostream>
using namespace std;

int main()
{
    int N;
    cin >> N;
    for(int starcount = 1; starcount <= N;starcount++)
    {
        cout << “*”;
    }
    return 0;
}
```

Question 2

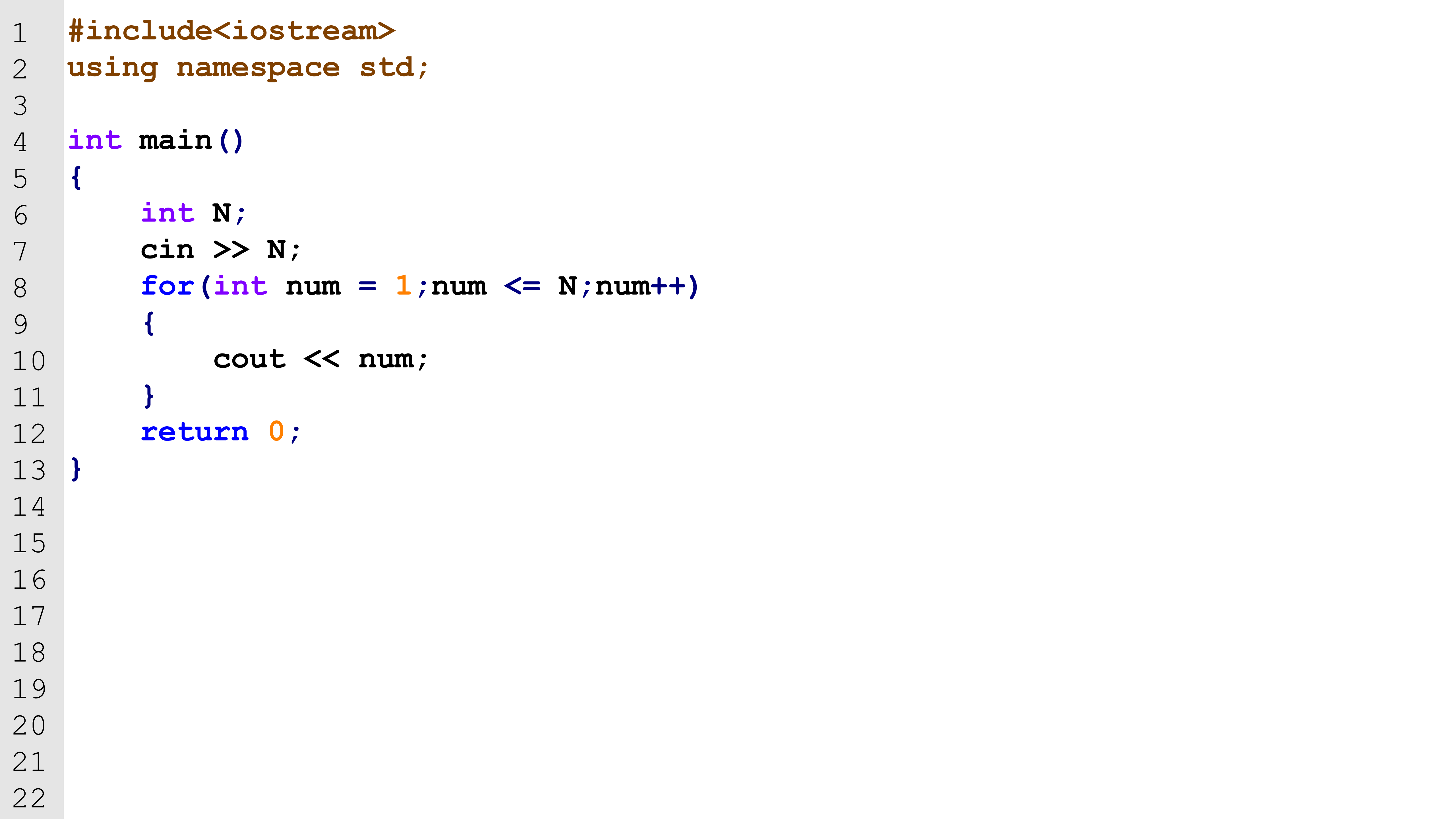
Printing numbers from 1 to N

Sample Input:

N = 4

Sample Output:

1234



```cpp
#include<iostream>
using namespace std;

int main()
{
    int N;
    cin >> N;
    for(int num = 1;num <= N;num++)
    {
        cout << num;
    }
    return 0;
}
```

Question 3

Printing numbers from 1 to N with comma

Sample Input:

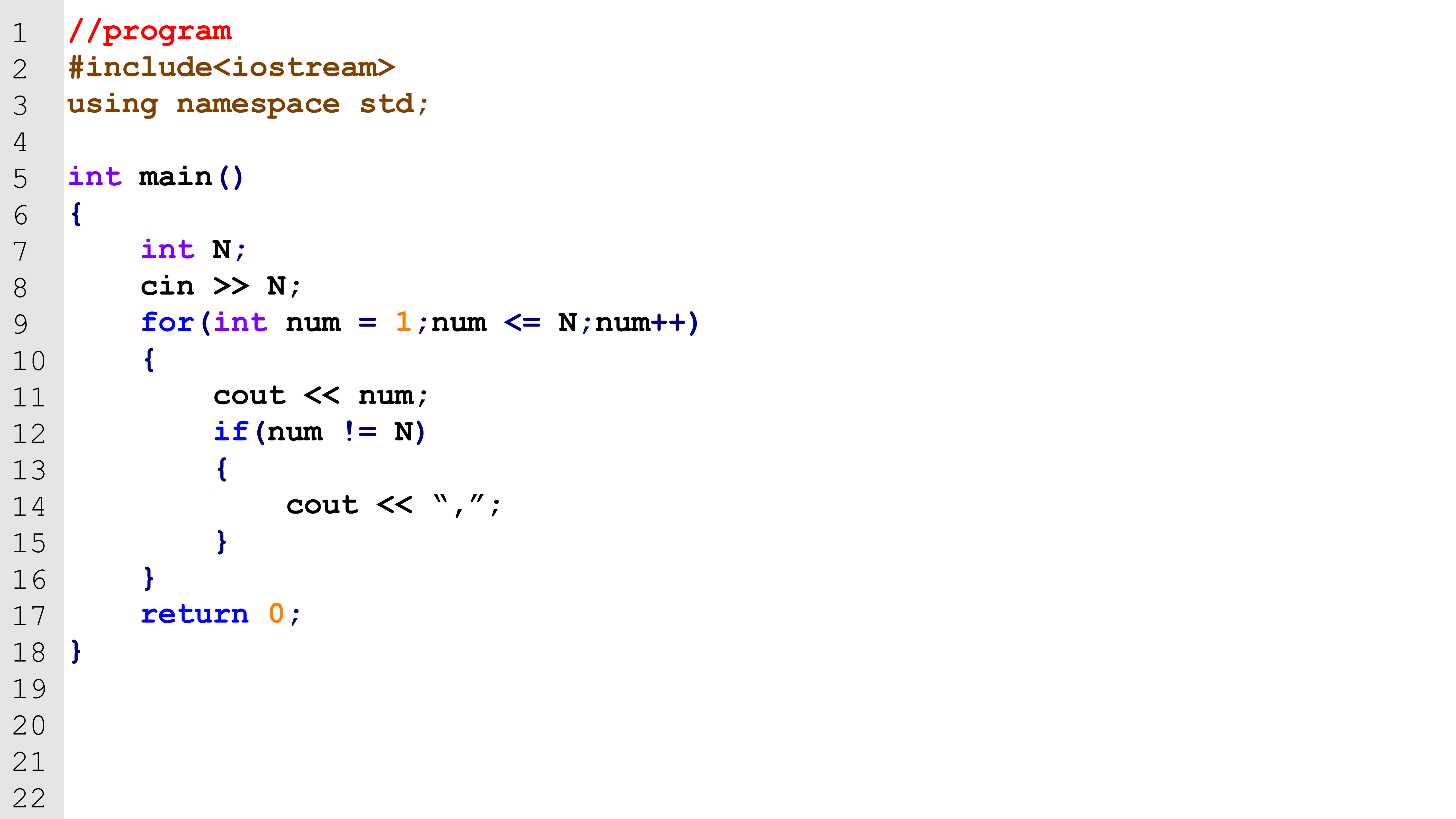
N = 4

Sample Output:

1,2,3,4

```
//Pseudocode
Input: N
for num = __1__ to _N_
{
        Print  num
        Print “,”
}
```

```
Sample Input:
4
Actual Output:
1,2,3,4,
//Pseudocode
Input: N
for num =  __1__  to _N_
{
      Print  num
      If(num != N)
      {
            Print “,”
      }
}
```



```
//program
#include<iostream>
using namespace std;

int main()
{
     int N;
     cin >> N;
     for(int num = 1;num <= N;num++)
     {
          cout << num;
          if(num != N)
          {
               cout << “,”;
          }
     }
     return 0;
}
```

Question 4

Printing * with #

Sample Input:

N = 8

M = 3

Sample Output:

```
***#***#**
```

```
//Pseudocode
Input: N, M
for starcount  =  1  to  N
{
      Print  "*"
      If(starcount  % M == 0){
            Print "#"
      }
}
```

Sample Input:

N = 9

M = 3

Actual Output:

```
***#***#***#
```

Expected Output:

```
***#***#***
```

```
//Pseudocode
Input: N, M
for starcount  =  1  to  N
{
      Print  “*”
      If(starcount  % M == 0 && starcount != N){
            Print “#”
      }
}
```

```cpp
//program
#include<iostream>
using namespace std;

int main()
{
    int N,M;
    cin >> N >> M;
    for(int starcount = 1; starcount <= N; starcount++)
    {
        cout << "*" ;
        if(starcount % M == 0 && starcount != N)
        {
            cout << "#" ;
        }
    }
    return 0;
}
```

Question 5

Printing * in 2-D

Sample Input:

N = 4

M = 2

Sample Output:

```
**
**
**
**
```

```
//pseudocode
Input: N, M
total_count = N * M
for starcount = __1__ to _total_count_
{
    Print “* ”
    If(starcount % M == 0){
        Print “\n”
    }
}
```

Better Coding convention…?

Key Idea

Sample Input:

N = 4

M = 2

Sample Output:

```
**
**
**
**
```

Row No	No. of Columns
1	2
2	2
3	2
4	2

For any value

Row No	No. of Columns
1	2
2	2
….	….
x	2

```
//Pseudocode
Input: N, M

for row_no = _1_ to _N_
{
      for col_no = _1_ to _M_
      {
            Print "*"
      }
      Print "\n"
}
```

```cpp
//program
#include<iostream>
using namespace std;

int main()
{
    int N,M;
    cin >> N >> M;
    for(int row_no = 1; row_no <= N; row_no++)
    {
        for(int col_no = 1; col_no <= M; col_no++)
        {
            cout << "*" ;
        }
        cout << endl;
    }
    return 0;
}
```

Question 6

Printing N lines of stars in Square fashion

Sample Input:

N = 4

Sample Output:

```
****
****
****
****
```

Key Idea

Sample Input:

N = 4

Sample Output:

```
****
****
****
****
```

Row No	No. of Columns
1	4
2	4
3	4
4	4

For any value

Row No	No. of Columns
1	4
2	4
....	....
x	4

```
//Pseudocode
Input: N
for row_no = _1_ to _N_
{
        for col_no = _1_ to _N_
        {
                Print “*”
        }
        Print “\n”
}
```

```cpp
//program
#include<iostream>
using namespace std;

int main()
{
    int N;
    cin >> N;
    for(int row_no = 1; row_no <= N; row_no++)
    {
        for(int col_no = 1; col_no <= N; col_no++)
        {
            cout << "*" ;
        }
        cout << endl;
    }
    return 0;
}
```

Question 7

Printing stars in right angle triangle

Sample Input:

N = 4

Sample Output:

```
*
**
***
****
```

Key Idea

Sample Input:
N = 4

Sample Output:

```
*
**
***
****
```

Row No	No. of Columns
1	1
2	2
3	3
4	4

For any value

Row No	No. of Columns
1	1
2	2
....	....
x	x

```
//Pseudocode
Input: N
for row_no = 1 to N
{
        for col_no = 1 to row_no
        {
                Print "*"
        }
        Print "\n"
}
```

```cpp
//program
#include<iostream>
using namespace std;

int main()
{
    int N;
    cin >> N;

    for(int row_no = 1; row_no <= N; row_no++)
    {
        for(int col_no = 1; col_no <= row_no; col_no++)
        {
            cout << "*" ;
        }
        cout << endl;
    }
    return 0;
}
```

Question 8

Printing numbers in right angle triangle

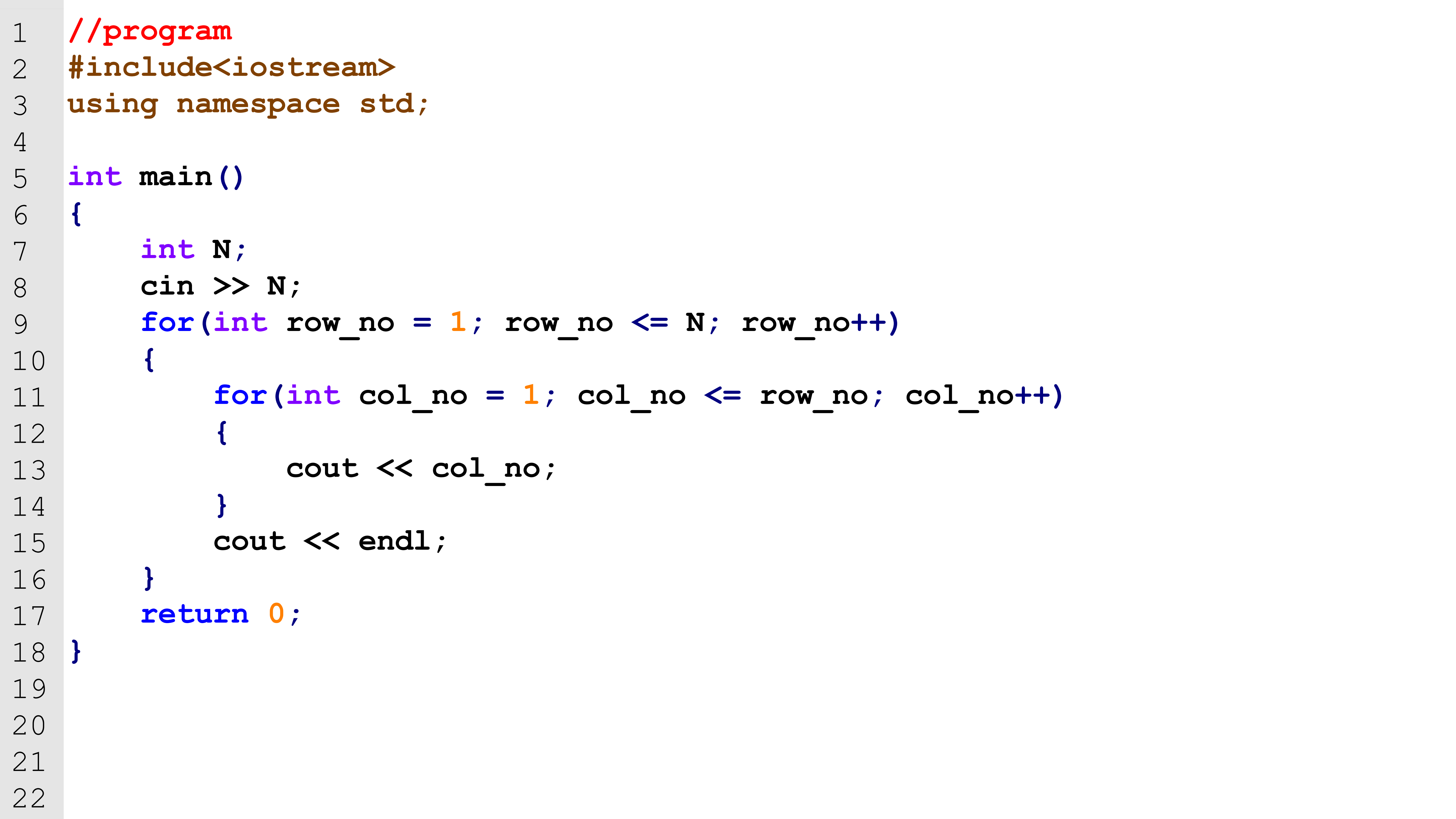
Sample Input:

N = 4

Sample Output:

```
1
12
123
1234
```

```
//Pseudocode
Input: N
for row_no = _1_ to _N_
{
    for col_no = _1_ to _row_no_
    {
        Print col_no
    }
    Print "\n"
}
```



```
//program
#include<iostream>
using namespace std;

int main()
{
    int N;
    cin >> N;
    for(int row_no = 1; row_no <= N; row_no++)
    {
        for(int col_no = 1; col_no <= row_no; col_no++)
        {
            cout << col_no;
        }
        cout << endl;
    }
    return 0;
}
```

Question 9

Printing increment numbers in right angle triangle

Sample Input:

N = 4

Sample Output:

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

```
//Pseudocode
Input: N
num = 1
for row_no = _1_ to _N_
{
        for col_no = _1_ to _row_no_
        {
                Print num
                num++
        }
        Print "\n"
}
```

```cpp
//program
#include<iostream>
using namespace std;

int main()
{
    int N,num = 1;
    cin >> N;

    for(int row_no = 1; row_no <= N; row_no++)
    {
        for(int col_no = 1; col_no <= row_no; col_no++)
        {
            cout << num;
            num++;
        }
        cout << endl;
    }
    return 0;
}
```

Question 10

Printing * and # in right angle triangle

Sample Input:

N = 4

Sample Output:

```
*
# #
* * *
# # # #
```

```
//Pseudocode
Input: N
for row_no = ___ to ___
{             1      N
      for col_no = ___ to ________{
                    1      row_no
            If(row_no % 2 != 0){
                  Print "*"
            }
            Else{
                  Print "#"
            }
      }
      Print "\n"
}
```

```cpp
//program
#include<iostream>
using namespace std;
int main()
{
    int N;
    cin >> N;
    for(int row_no = 1; row_no <= N; row_no++)
    {
        for(int col_no = 1; col_no <= row_no; col_no++)
        {
            if(row_no % 2 != 0){
                cout << "* " ;
            }
            else{
                cout << "# " ;
            }
        }
        cout << endl;
    }
    return 0;
}
```

Question 11

Printing numbers in inverted right angle triangle

Sample Input:

N = 4

Sample Output:

```
1 2 3 4
1 2 3
1 2
1
```

```
//Pseudocode
Input: N
num = _N_
for row_no = __1__ to ___N___
{
    // Handle stars for each row
    for col_no = __1__ to _num_____
    {
        print ______col_no______
    }
    print "\n"
    num = num - 1;
}
```

```
Sample Input:
4

Sample Output:
1 2 3 4
1 2 3
1 2
1
```

Question 12

Printing numbers in inverted right angle triangle

Sample Input:

N = 4

Sample Output:

```
1 2 3 4
2 3 4
3 4
4
```

```
//Pseudocode
Input: N
 for row_no = ____ to _______
 {
     // Handle stars for each row
     for col_no = ____ to ________
     {
         print  ______________
     }
     print “\n”
 }
```

Sample Input:

```
4
```

Sample Output:

```
1 2 3 4
2 3 4
3 4
4
```

Question 13

Printing stars in pyramid

Sample Input:

N = 4

Sample Output:

```
   *
  * *
 * * *
* * * *
```

Key Idea

Row No	No. of Columns	
	No. of Spaces	No. of stars
1	3	1
2	2	2
3	1	3
4	0	4

For any value,

Row No	No. of Columns	
	No. of Spaces	No. of stars
1	N-1	1
2	N-2	2
....	....	....
x	N-x	x

```
//pseudocode
Input: N
for row_no = _1_ to ___N___
{
    // Handle spaces for each row
    for space = _1_ to ___N-row_no___
    {
        print " "
    }
    // Handle stars for each row
    for col_no = _1_ to ___row_no___
    {
        print "* "
    }
    print "\n"
}
```

Row No	No. of Columns	
	No. of Spaces	No. of stars
1	N-1	1
2	N-2	2
....	....	....
x	N-x	x

Question 14

Printing stars in inverse pyramid

Sample Input:

N = 4

Sample Output:

```
* * * *
 * * *
  * *
   *
```

Key Idea

Row No	No. of Columns	
	No. of Spaces	No. of stars
1	0	4
2	1	3
3	2	2
4	3	1

For any value, star_count = N

Row No	No. of Columns	
	No. of Spaces	No. of stars
1	1-1	4
2	2-1	3
….	….	….
x	x-1	star_count

```
//pseudocode
Input: N
for row_no = ___ to ____
{
        // Handle space for each row
        for space = ____ to ________
        {
            print " "
        }
        // Handle stars for each row
        for curr_col_no = __  to _____
        {
            print "* "
        }
        print "\n"

}
```

Row No	No. of Columns	
	No. of Spaces	No. of stars
1	1-1	4
2	2-1	3
….	….	….
x	x-1	star_count

```cpp
//program
#include<iostream>
using namespace std;

int main()
{
    int N;
    scanf("%d", &N);
    int star_count = N;
    for(int cur_row_no = 1; cur_row_no <= N; ++cur_row_no){
        for(int space = 1; space <= cur_row_no - 1; ++space){
            printf(" ");
        }
        for(int cur_col_no = 1; cur_col_no <= star_count; ++cur_col_no){
            printf("* ");
        }
        printf("\n");
        star_count = star_count-1;
    }
    return 0;
}

```

Question 15

Printing stars in diamond fashion

Sample Input:

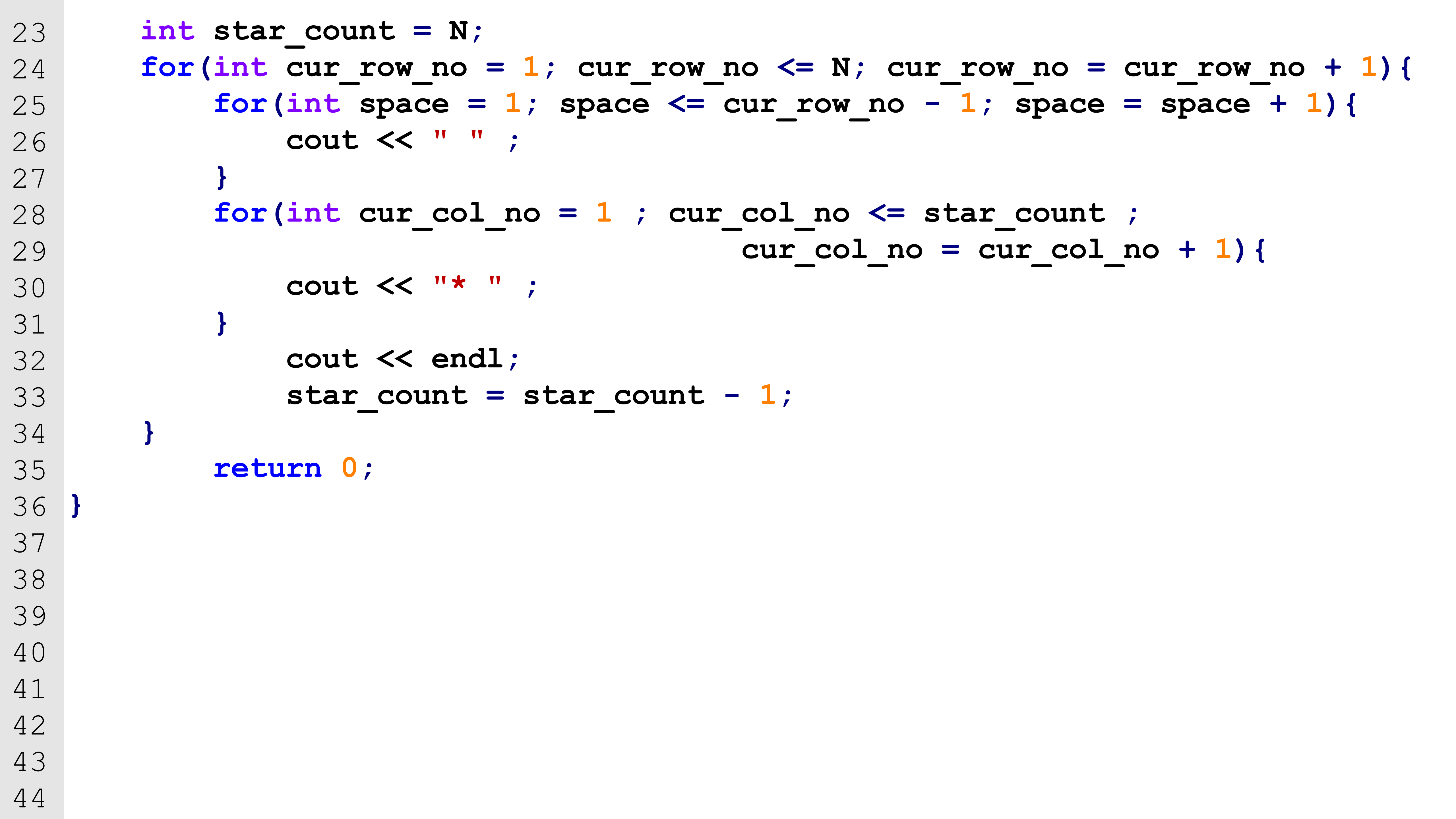
N = 4

Sample Output:

```
   *
  * *
 * * *
* * * *
* * * *
 * * *
  * *
   *
```

```
//pseudocode
Input: N
for cur_row_no = 1 to N
{
  //Handle spaces for each row
  for space = 1 to N - cur_row_no
  {
    print " "
  }
  //Handle stars for each row
  for cur_col_no = 1 to cur_row_no
  {
    print "* "
  }
  print "\n"
}
star_count = N
for cur_row_no = 1 to N
{
  //Handle space for each row
  for space = 1 to cur_row_no-1
  {
    print " "
  }
  //Handle stars for each row
  for cur_col_no = 1 to star_count
  {
    print "* "
  }
  print "\n"
  star_count = star_count - 1
}
```

```cpp
//program
#include<iostream>
using namespace std;

int main()
{
    int N;
    cin >> N;
    for(int cur_row_no = 1; cur_row_no <= N; ++cur_row_no){
        for(int space = 1; space <= N - cur_row_no; ++space){
            cout << " " ;
        }
        for(int cur_col_no = 1; cur_col_no <= cur_row_no; ++cur_col_no){
            cout << "* " ;
        }
        cout << endl;
    }
```



```
    int star_count = N;
    for(int cur_row_no = 1; cur_row_no <= N; cur_row_no = cur_row_no + 1){
        for(int space = 1; space <= cur_row_no - 1; space = space + 1){
            cout << " " ;
        }
        for(int cur_col_no = 1 ; cur_col_no <= star_count ;
                                     cur_col_no = cur_col_no + 1){
            cout << "* " ;
        }
            cout << endl;
            star_count = star_count - 1;
    }
        return 0;
}
```

Question 16

Printing stars in mirrored right triangle

Sample Input:

N = 4

Sample Output:

```
      *
    * *
  * * *
* * * *
```

Key Idea

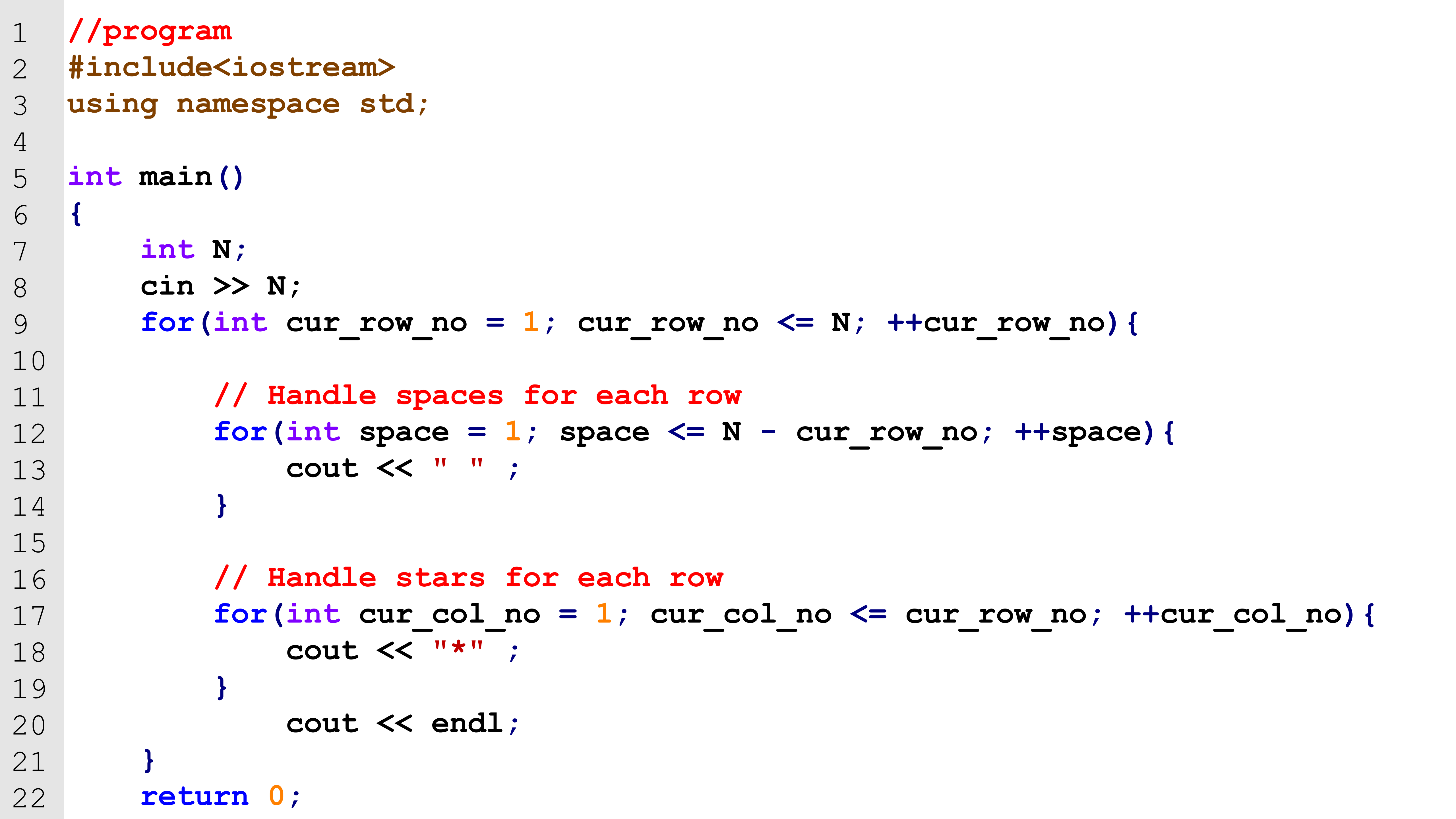
Row No	No. of Columns	
	No. of Spaces	No. of stars
1	3	1
2	2	2
3	1	3
4	0	4

For any value,

Row No	No. of Columns	
	No. of Spaces	No. of stars
1	N-1	1
2	N-2	2
….	….	….
x	N-x	x

```
//pseudocode
Input: N
for row_no = _1_ to _N_
{
    // Handle spaces for each row
    for space = __1__ to __N - row_no__
    {
        print " "
    }
    // Handle stars for each row
    for col_no = _1_ to ____row_no____
    {
        print "*"
    }
    print "\n"
}
```

Row No	No. of Columns	
	No. of Spaces	No. of stars
1	N-1	1
2	N-2	2
….	….	….
x	N-x	x



```
//program
#include<iostream>
using namespace std;

int main()
{
    int N;
    cin >> N;
    for(int cur_row_no = 1; cur_row_no <= N; ++cur_row_no){

        // Handle spaces for each row
        for(int space = 1; space <= N - cur_row_no; ++space){
            cout << " " ;
        }

        // Handle stars for each row
        for(int cur_col_no = 1; cur_col_no <= cur_row_no; ++cur_col_no){
            cout << "*" ;
        }
            cout << endl;
    }
    return 0;
```

Question 17

Printing stars in Inverse Half Diamond Pattern

Sample Input:

N = 4

Sample Output:

```
*******
 *****
  ***
   *
```

Key Idea

Row No	No. of Columns	
	No. of Spaces	No. of stars
1	0	7
2	1	5
3	2	3
4	3	1

For any value, *star_count = (2*N)-1*

Row No	No. of Columns	
	No. of Spaces	No. of stars
1	0	star_count **(2*N)-1**
2	1	star_count-2
….	….	….
x	x-1	star_count-2

```
//pseudocode
Input: N
star_count = ___2*N-1___
for row_no = _1_ to __N__
{

   // Handle spaces for each row
   for space = _1_ to _row_no - 1_
   {
     print " "
   }

   // Handle stars for each row
   for col_no =_1_ to _star_count- 2*(row_no-1)_
   {
     print "*"
   }
  print "\n"
   star_count = star_count - 2
}
```

Row No	No. of Columns	
	No. of Spaces	No. of stars
1	0	star_count
2	1	star_count-2
....	....	....
x	x-1	star_count-2

Anything Missing?

Alternate way ?

```
//program
#include<iostream>
using namespace std;

int main()
{
   int N;
   cin >> N;
   int star_count = 2 * N - 1;
   for(int row_no = 1; row_no <= N; ++row_no)
   {
          for(int space = 1;space <= row_no - 1;++space)
          {
              cout << " " ;
          }
          for(int col_no=1;col_no<=star_count-2*(row_no-1);++col_no)
          {
              cout << "*" ;
          }
          cout << endl;
   }
   return 0;
```

Question 18

Printing stars in diamond fashion

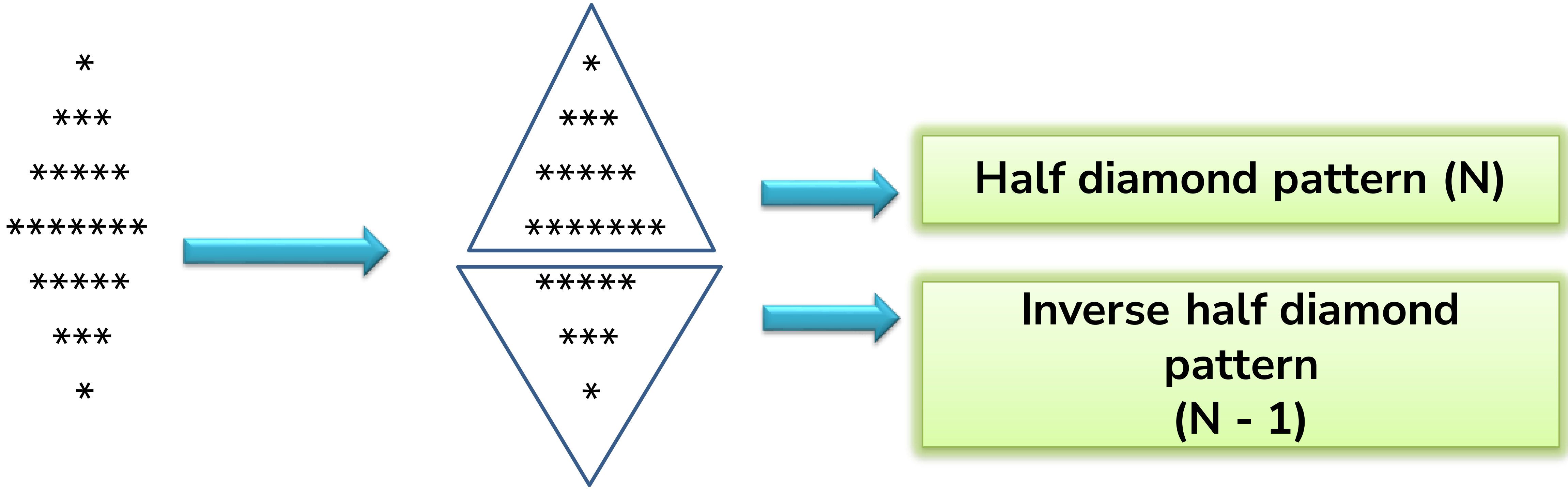
Sample Input:

N = 4

Sample Output:

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

Key Idea



```
   *
  ***
 *****
*******
 *****
  ***
   *
```

Half diamond pattern (N)

Inverse half diamond pattern
(N - 1)

```
//psudocode
Input: N
for row_no = _1_ to __N___
{
  //Handle spaces for each row
  for space = _1_ to __N-row_no___
  {
    print " "
  }
  //Handle stars for each row
  for col_no = _1_ to _2*row_no - 1_
  {
    print "*"
  }
  print "\n"
}
star_count = __(2*N-1)- 1__
for row_no = _1_ to __N___
{
  //Handle spaces for each row
  for space = _1_ to __row_no___
  {
    print " "
  }
  //Handle stars for each row
  for col_no = _1_to __star_count___
  {
    print "*"
  }
  print "\n"
  star_count = star_count - 2
}
```

Question 19

Triangle pattern - Nos and stars

Sample Input:

N = 4

Sample Output:

```
1
2*2
3*3*3
4*4*4*4
4*4*4*4
3*3*3
2*2
1
```

Question 20

Diamond pattern - Nos and stars

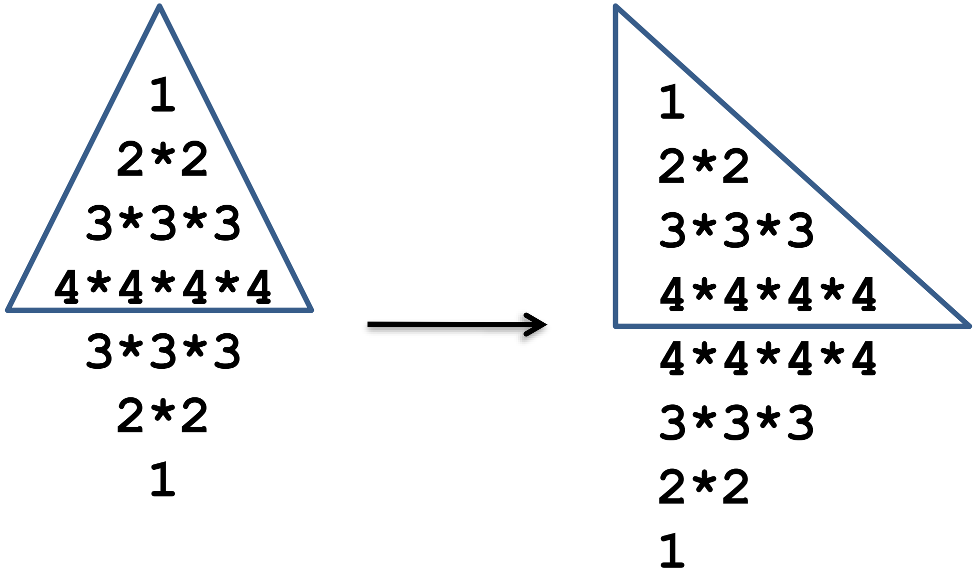
Sample Input:

N = 4

Sample Output:

```
   1
  2*2
 3*3*3
4*4*4*4
 3*3*3
  2*2
   1
```

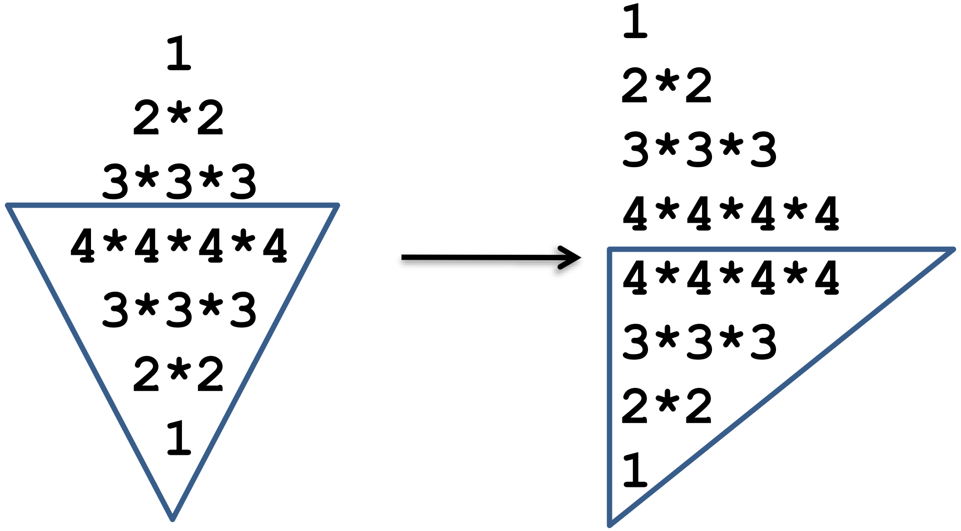
```
//pseudocode
Input: N
for row_no = 1 to N
{
  //Handle spaces for each row
  for space = 1 to N - row_no
  {
    print " "
  }
  //Handle stars for each row
  for col_no = 1 to 2 * row_no - 1
  {
    if( col_no % 2 == 1 ){
        print  row_no
    }
    else{
        print "*"
    }
  }
  print "\n"
}
```



```
   1
  2*2
 3*3*3
4*4*4*4
 3*3*3
  2*2
   1
```

```
1
2*2
3*3*3
4*4*4*4
4*4*4*4
3*3*3
2*2
1
```

```
num = N -1
N--
no_star_count = (2*N)  - 1
for row_no = 1 to N{
  // Handle spaces for each row
  for space = 1 to row_no -1{
    print " "
  }
  //Handle stars for each row
  for col_no = 1 to no_star_count{
    if(____col_no % 2 == 1____){
        print ____num
    }
    else{
        print "*"
    }
  }
  print "\n"
  star_count = no_star_count - 2
  num = num - 1
}

```



```
   1
  2*2
 3*3*3
4*4*4*4
 3*3*3
  2*2
   1
```

```
1
2*2
3*3*3
4*4*4*4
4*4*4*4
3*3*3
2*2
1
```

Question 21

Printing custom number pattern

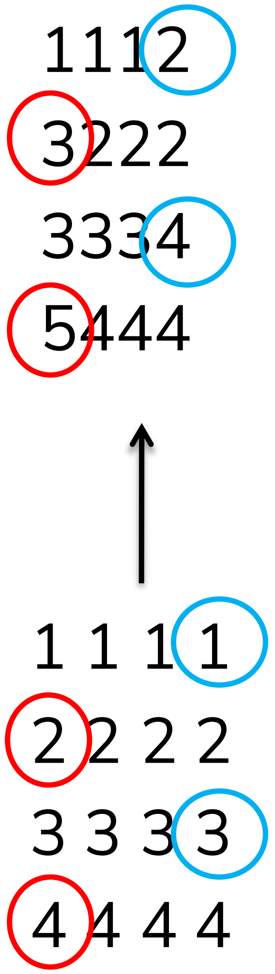
Sample Input:

N = 4

Sample Output:

```
1112
3222
3334
5444
```

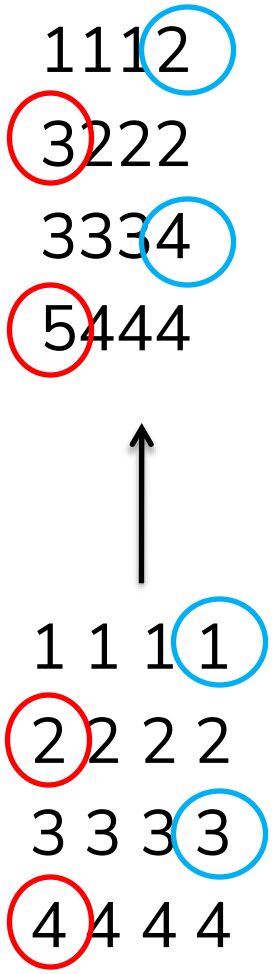
```
//pseudocode
Input: N
num = 1
for row_no = 1 to N
{
    for col_no = 1 to  N{
        Print row_no
    }
    print "\n"
}
```



```
1112
3222
3334
5444
```

```
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
```

```
//pseudocode
Input: N
for row_no = 1 to N
{
    for col_no = 1 to  N{
        if (((row_no % 2 == 0) && (col_no == 1))  ||
            ((row_no % 2 == 1) && (col_no == N))  )
        {
            print row_no + 1
        }
        else
        {
            print row_no
        }
    }
    print "\n"
}
```



```
1112
3222
3334
5444

1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
```

Question 1

A teacher wants to compute average marks in her history classes
In her class , students count is 10

```
// Pseudocode
Input: stu_1_mark, stu_2_mark, …,stu_10_mark
avg = (stu_1_mark + stu_2_mark + … + stu_10_mark) / 10
print avg
```

Question 2

A teacher wants to compute average marks in her history classes
In her class , students count is 120

```
// Pseudocode
Input: stu_1_mark, stu_2_mark, …,    …….stu_120_mark
avg = (stu_1_mark + stu_2_mark + … + stu_120_mark) / 120
print avg
```

Design for large inputs

Too many variables.

Lets find a solution

Goals:

- Variable count != number of inputs
- One variable for entire set
- Still identify inputs individually
- Like first number + second number + …

Design for large inputs

Solution:

marks[] = {21,24,25,28,32}, size = 5;

- **marks**
  - holds the list of numbers
  - Type – Array: Indicated by []
- **Size**
  - Number of elements in list marks

Design for large inputs

How to access elements in the list???

- First element - marks[0]
- Second element - marks[1]
- Last element ?

marks[4]

Memory Allocation

int marks[4];

Syntax:

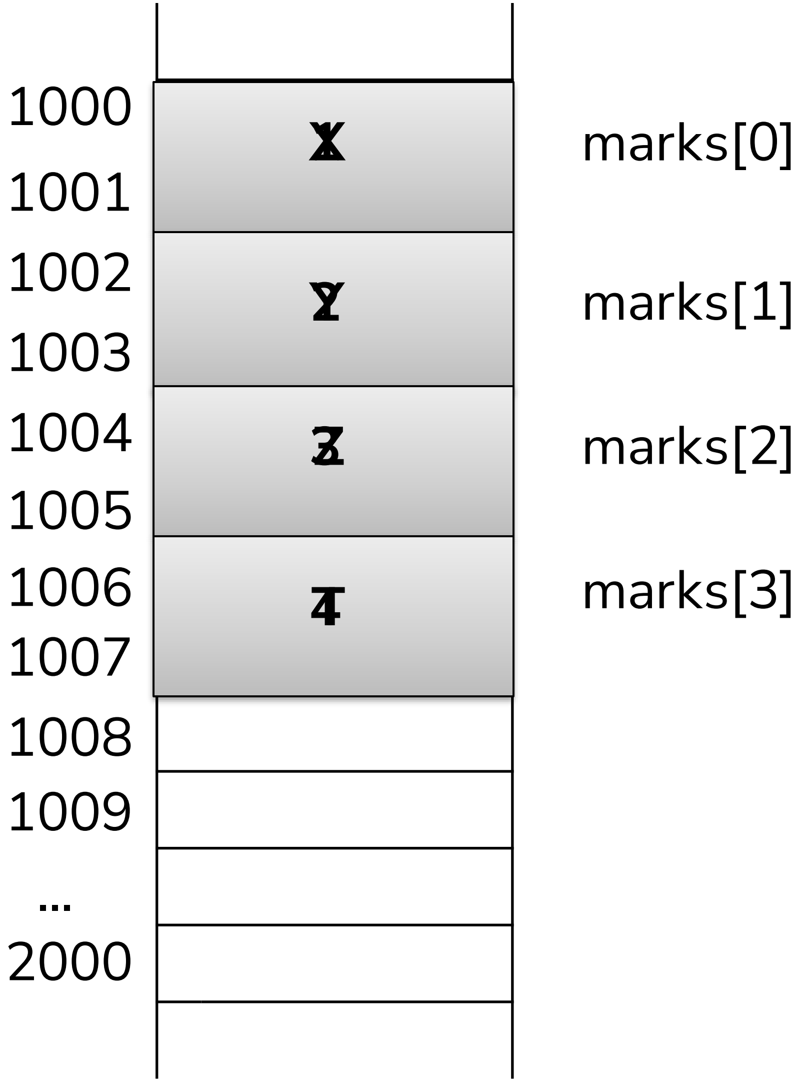
datatype arrayname[size];

marks[0] = 1

marks[1] = 2

marks[2] = 3

marks[3] = 4



Memory Allocation

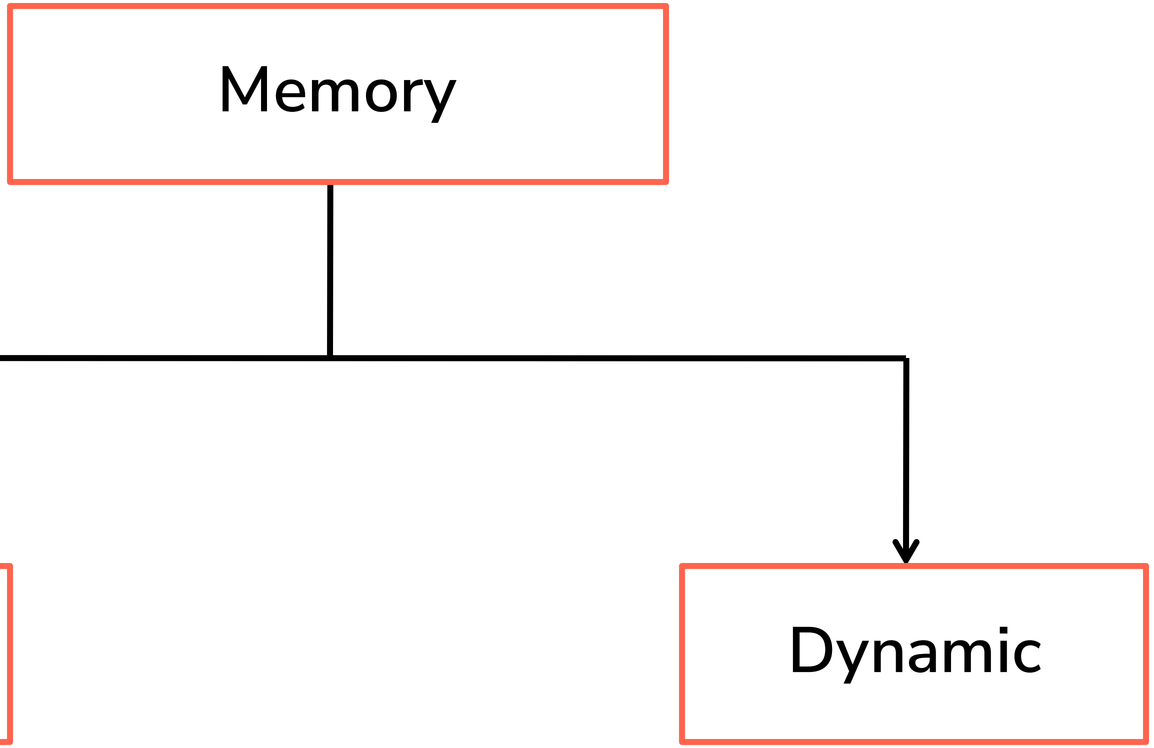
How to access ith element?

- Index = i - 1
- Access it as marks[i - 1]

Invalid indexing like marks[10] or marks[5]

- returns undefined value.

Array

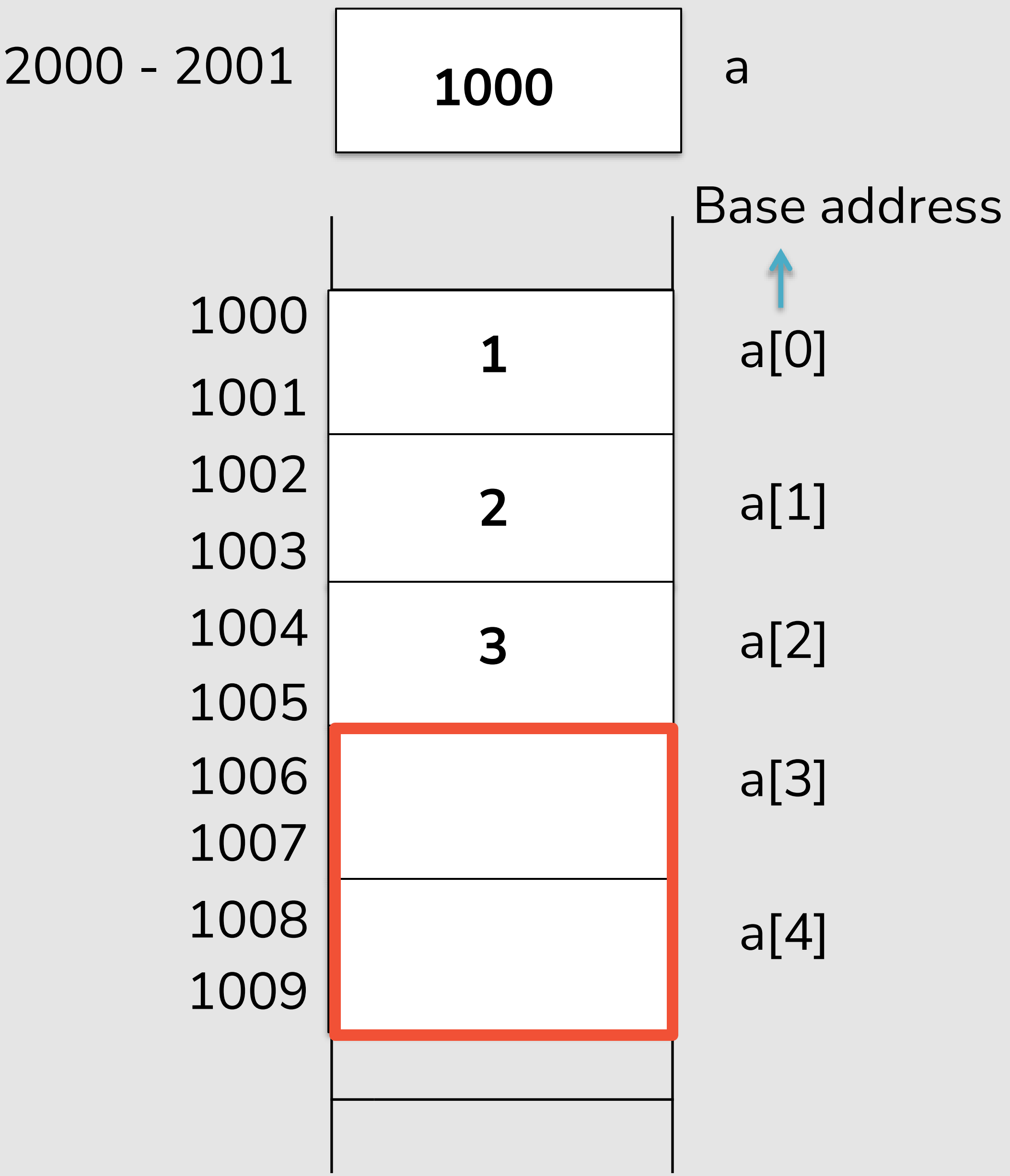


Memory

Static

Dynamic

```
// Code  [Static Array]
#include<iostream>
using namespace std;
int main()
{
      int a[5];
      int n;
      cin >> n; // 3
      for(int i = 0; i < n; i++)
      {
            cin >> a[i];
      }
      for(int i = 0; i < n; i++)
      {
            cout << a[i];
      }
      return 0;
}
```



Pointers

To accessing the address directly

Advantage:

- No copy of data
- Pointer is a variable which holds the address of another variable
- int *a; → Pointer to an integer
- **Syntax:**

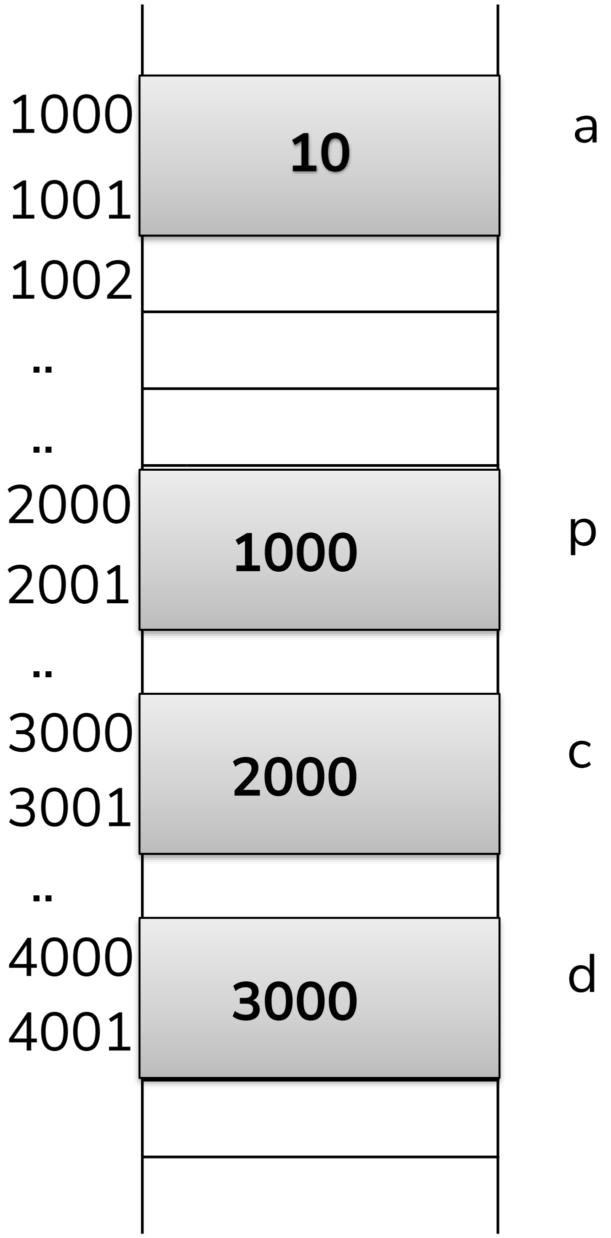
Size..??

```
datatype * var_name
```

Pointers

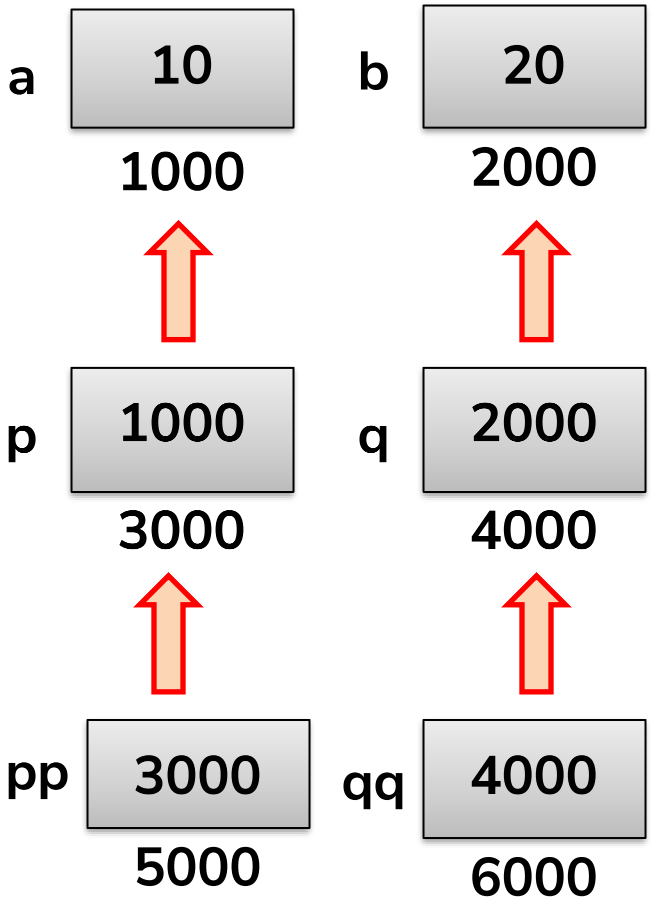
```
int a = 10;

int *p;

p = &a;

int **c;

c = &p;

int ***d;

d = &c;
```

a	=	10
&a	=	1000
p	=	1000
&p	=	2000
*p	=	10
c	=	2000
&c	=	3000
*c	=	1000
**c	=	10
d	=	3000
&d	=	4000
*d	=	2000
**d	=	1000
***d	=	10



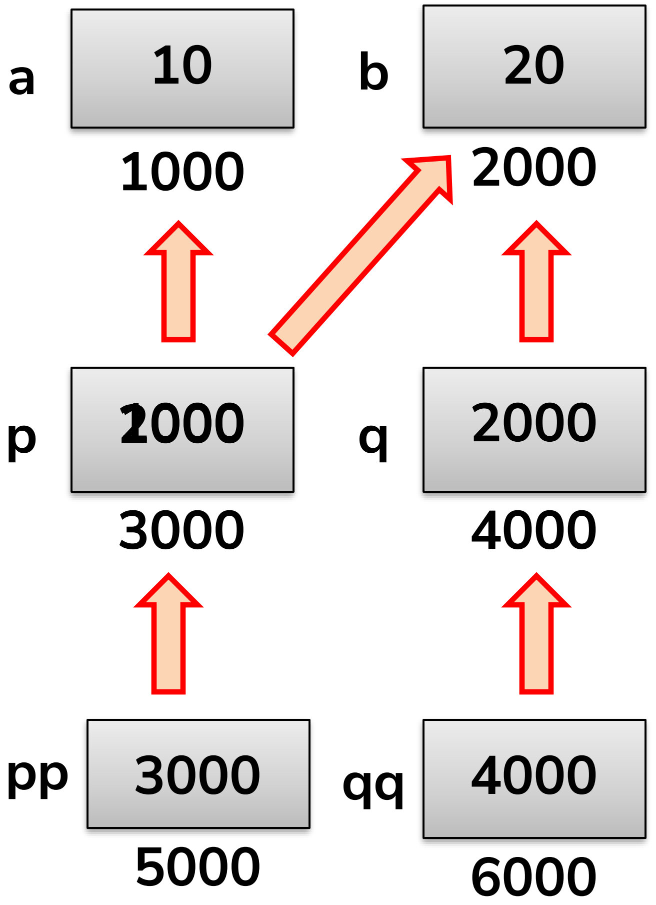
Pointers

a = 10
p = 1000
&p = 3000
*p = 10
pp = 3000
&pp = 5000
*pp = 1000
**pp = 10
b = 20
q = 2000
*q = 20
**qq =20



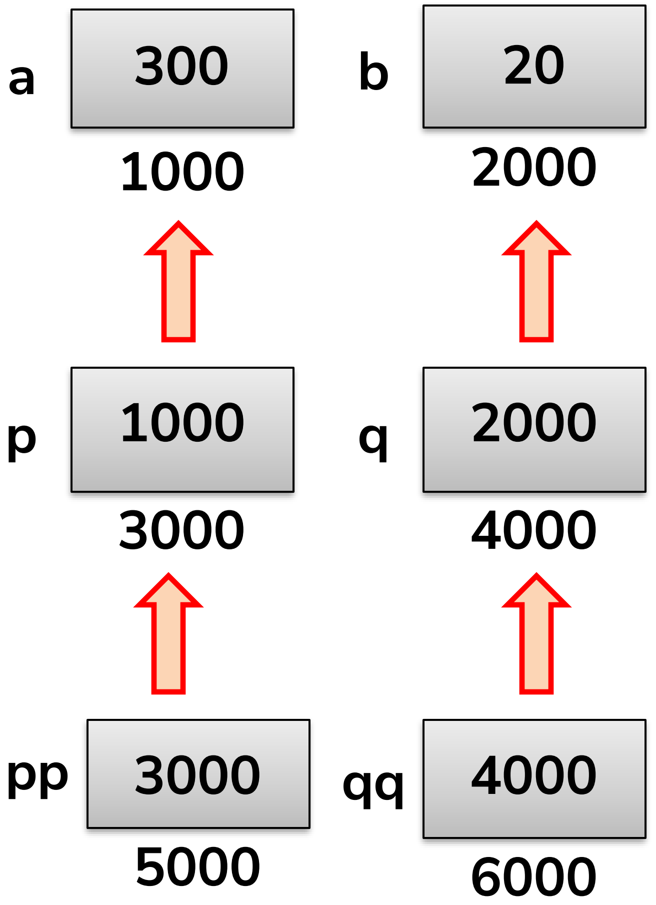
Pointers

p = q; p = 2000
a = 10
p = 2000
&p = 3000
*p = 20
pp = 3000
*pp = 2000
**pp = 20
b = 20
q = 2000
*q = 20
**qq =20



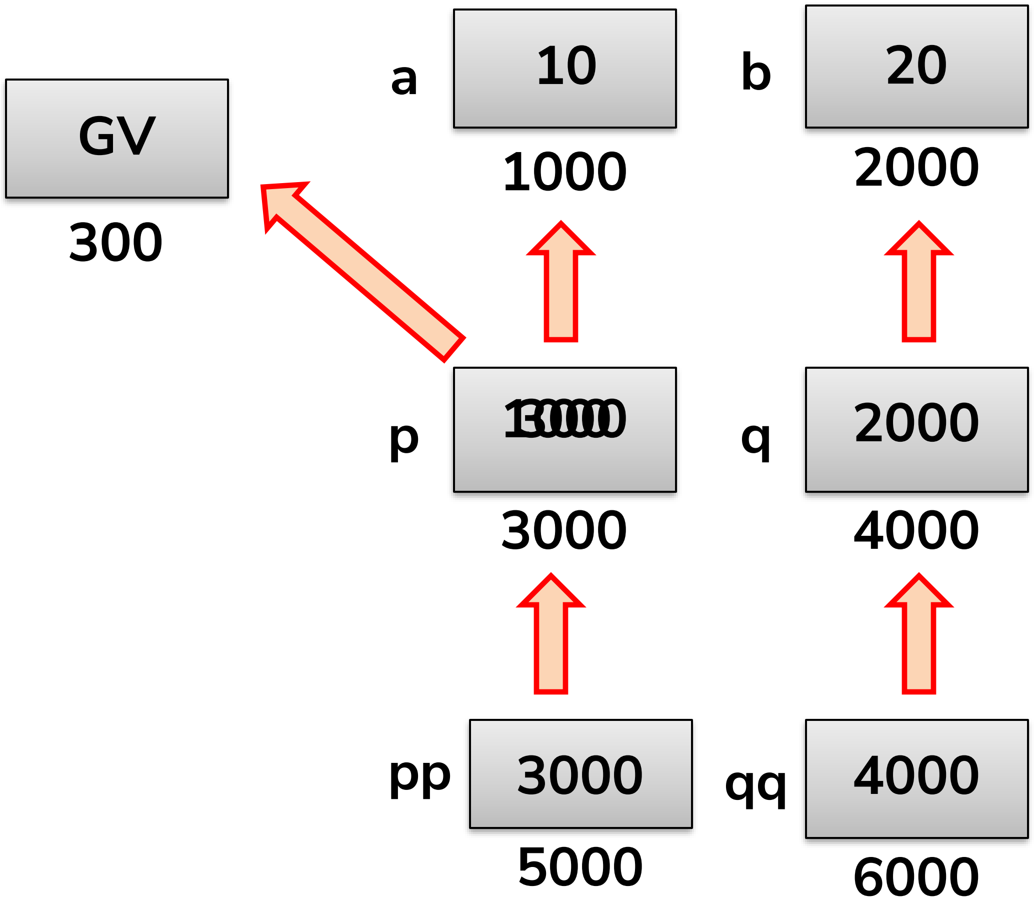
Pointers

*p = 300;
a = 300
p = 1000
&p = 3000
*p = 300
pp = 3000
&pp = 5000
*pp = 1000
**pp = 300
b = 20
q = 2000
*q = 20

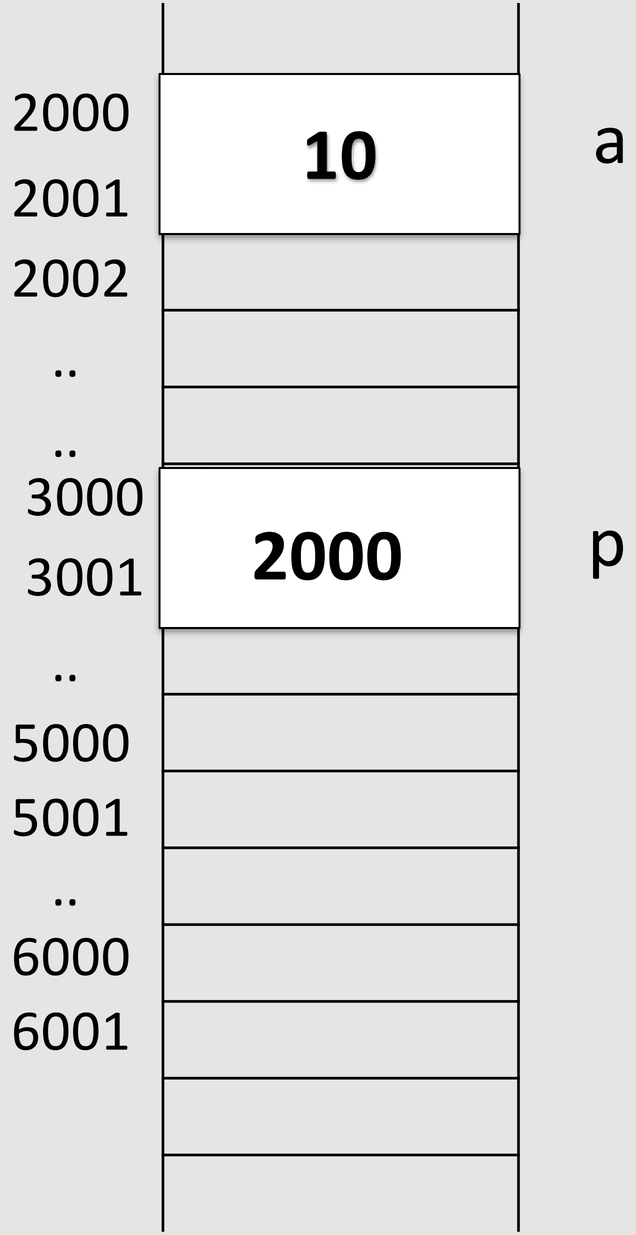


Pointers

p = 300;	
a	= 10
p	= 300
&p	= 3000
*p	= GV
pp	= 3000
&pp	= 5000
*pp	= 300
**pp	= GV
b	= 20
q	= 2000
*q	= 20



```
int a = 10;
int *p;
p = &a;

p + 2
  ↓

3000  + 2 * 2
  ↓

3004


p + 4
  ↓

3000  + 4 * 2
  ↓

3008
```



Dynamic Memory Allocation

- malloc()
- realloc()
- calloc()

} Allocates memory

- free() → Deallocates the memory

malloc()

- Allocates single block of requested memory.
- It returns NULL if memory is not sufficient.
- **Syntax :**

```
ptr=(cast-type*)malloc(byte-size)
```

calloc()

- Allocates multiple block of requested memory.
- It initially initialize all bytes to zero.
- It returns NULL if memory is not sufficient.
- **Syntax:**

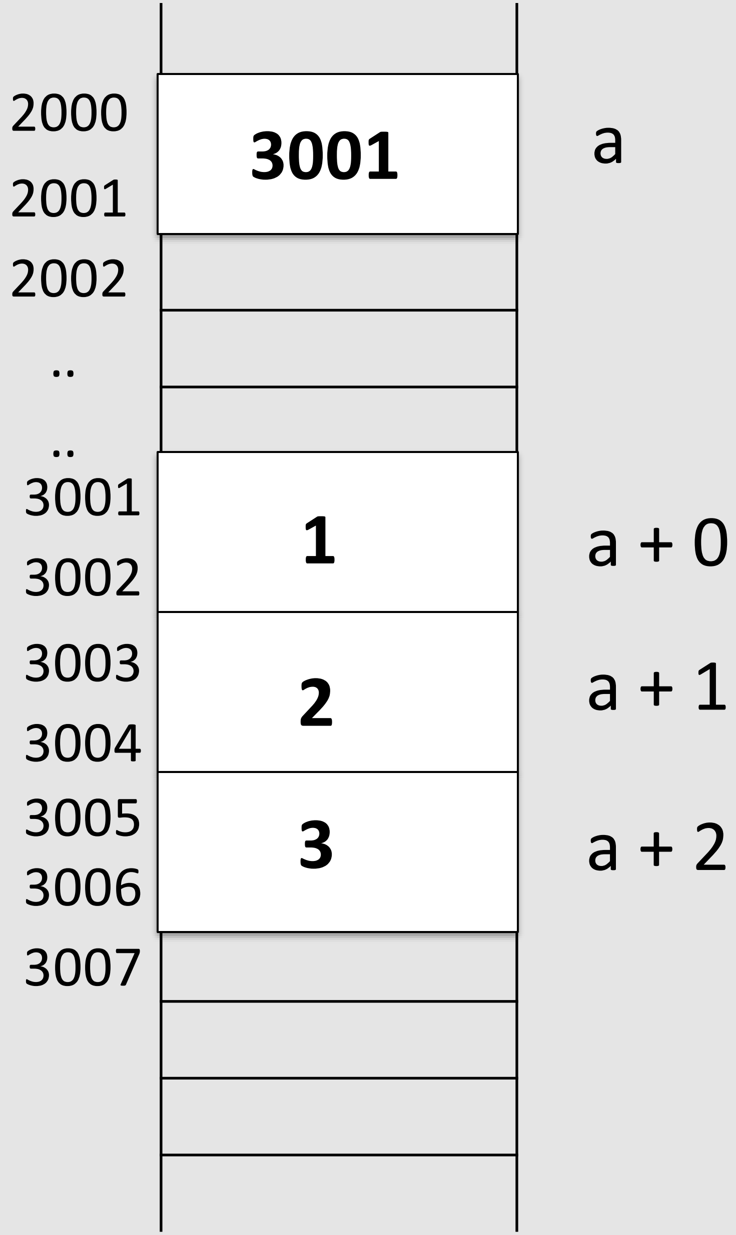
```
ptr=(cast-type*)calloc(number, byte-size)
```

realloc()

- If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function.
- In short, it changes the memory size.
- **Syntax:**

```
ptr=realloc(ptr, new-size)
```

```
// Code
#include<stdio.h>
int main()
{
    int *a;
    int n;
    cin >> n;           // 3
    a = (int *)malloc(n * sizeof(int));
    for(int i = 0; i < n; i++)
    {
        cin >> a + i;
    }
    for(int i = 0; i < n; i++)
    {
        cout << *(a + i);
    }
    return 0;
}
```



Question 1

A teacher wants to store a subject marks for every student.
In her class ,students count is 120. How will she do it?

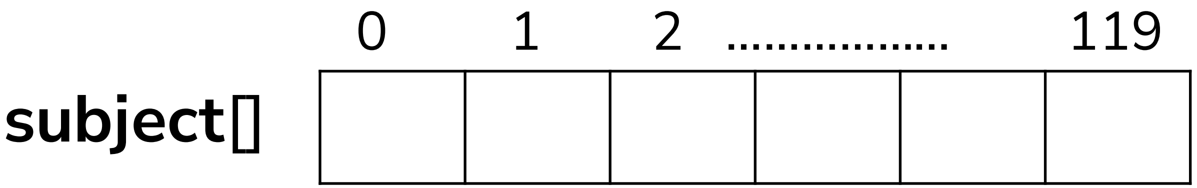
A) Structure

B) Union

[✓ **C)** Arrays]

D) Variables

How can we create Arrays?



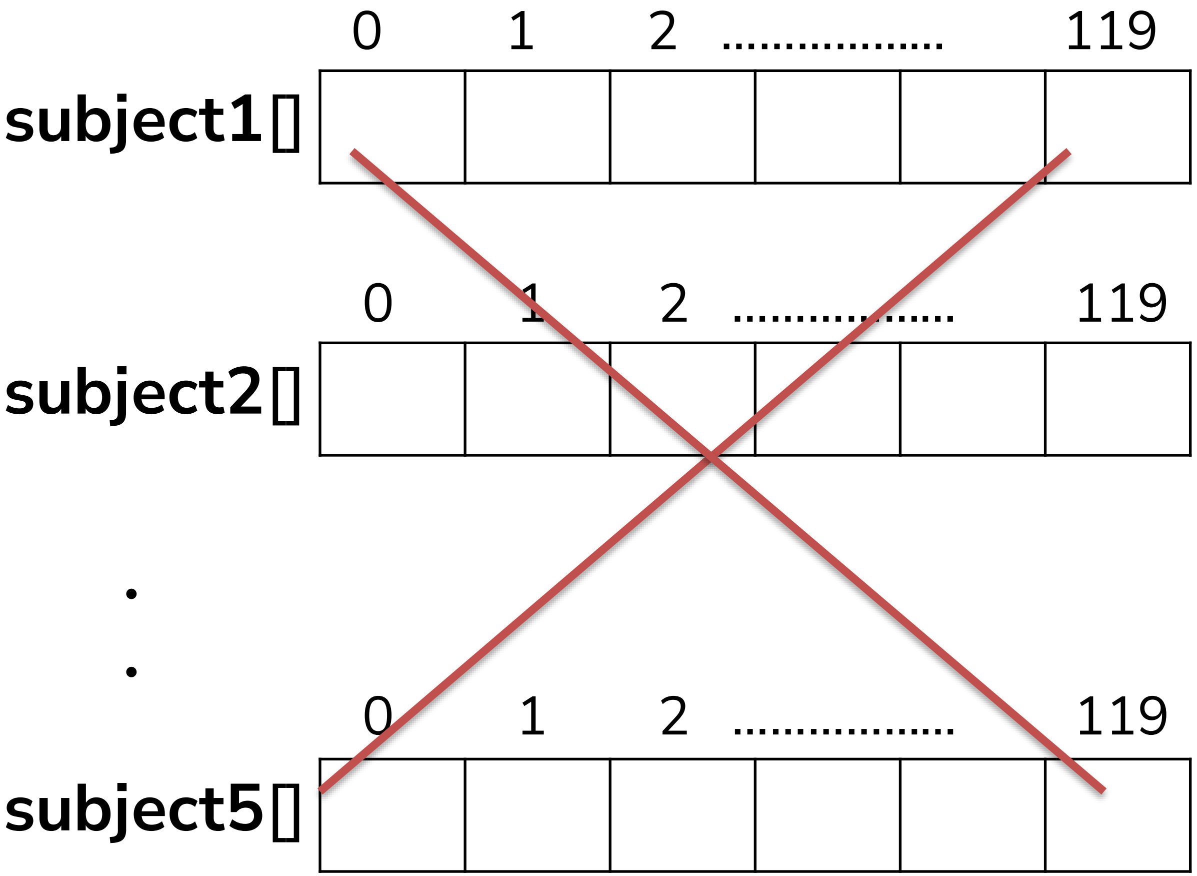
0 1 2 119

subject[]

Question 2

A teacher wants to store 5 subject marks for every student.
In her class ,students count is 120. How will she create it?

How to store values?



Design for Large Inputs

Too many arrays.

Lets find a solution

Goals

- Arrays count != number of subjects
- One array for entire set
- Still identify inputs individually
- Like 1st student’s 1stsubject mark +2ndstudent’s 1stsubject mark + …

Design for Large Inputs

Solution

- **marks[][] = { {21, 24, 25, 28, 32} , {69, 42, 63, 45, 95} };**
- **marks**
  - Holds the list of numbers
  - Type – 2D Array: Indicated by [][]
- **Row size**
  - Number of subjects in the array
- **Column size**
  - Number of students in the array

row size = 2;

column size = 5;

Design for Large Inputs

How to access elements in the array???

- First subject first student’s mark - marks[0][0]
- First subject second student’s mark – marks[0][1]
- Second subject last student’s mark ???

marks[1][4]

Memory Allocation

int marks[2][2];

Syntax:

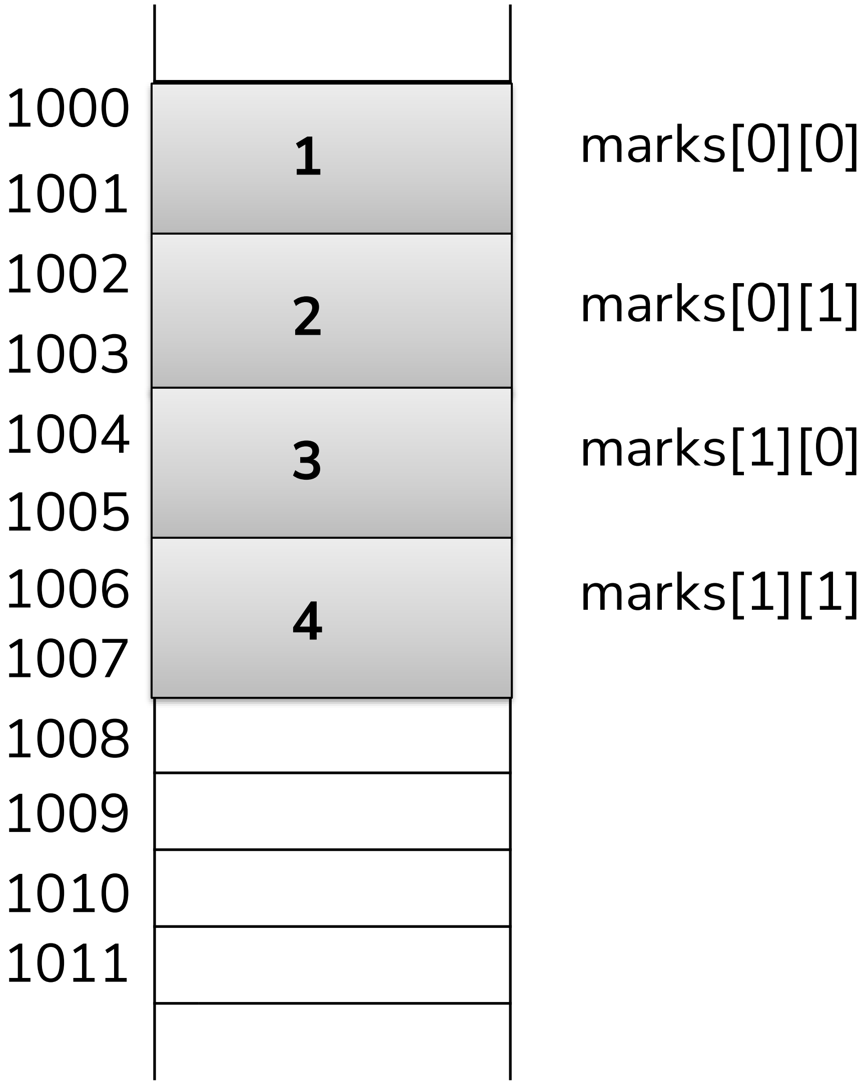
datatype arrayname[rowsize][columnsize];

marks[0][0] = 1

marks[0][1] = 2

marks[1][0] = 3

marks[1][1] = 4



Memory Allocation

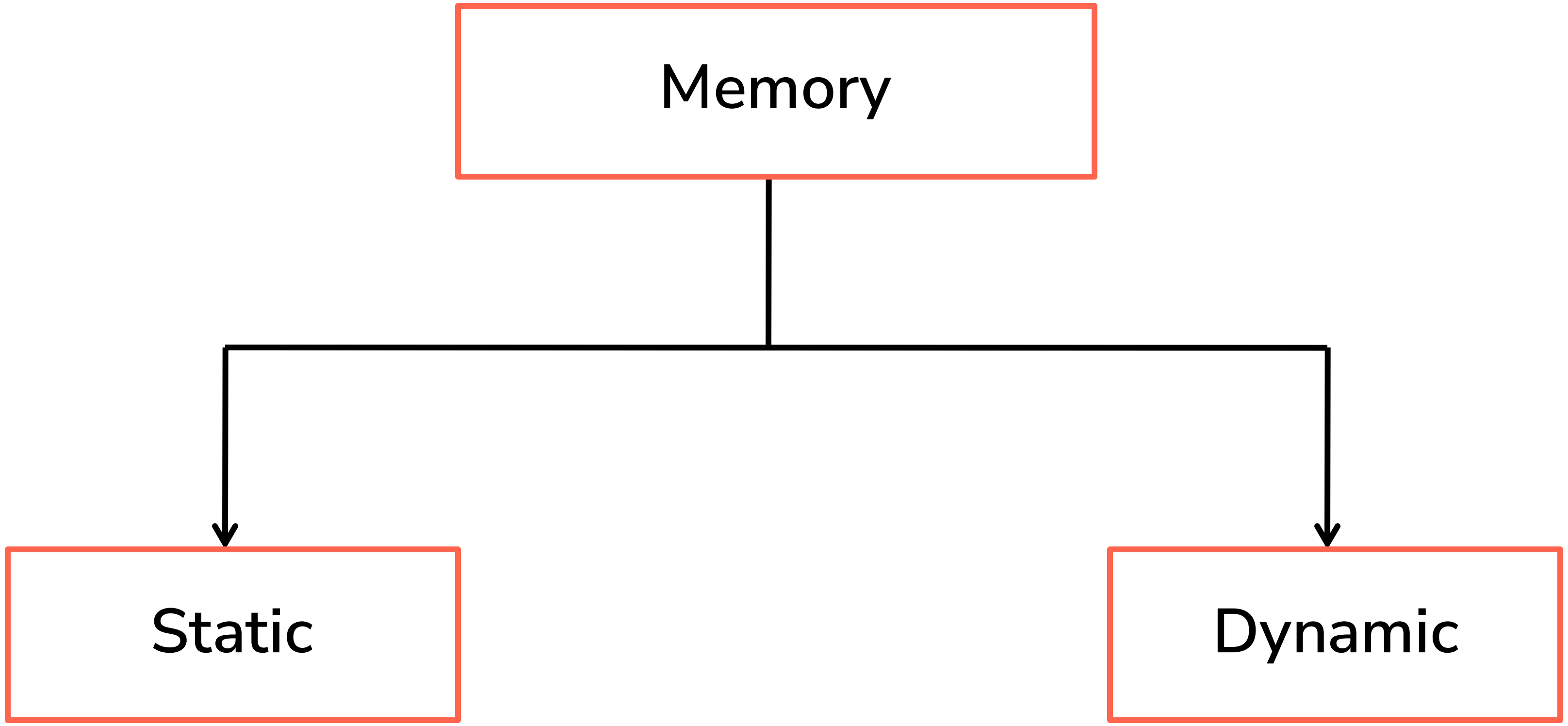
How to access i^th element and j^th element?

- Row Index = i – 1, Column Index = j – 1
- Access it as marks[i - 1][j – 1]

Invalid indexing like marks[3][4] or marks[5][0]

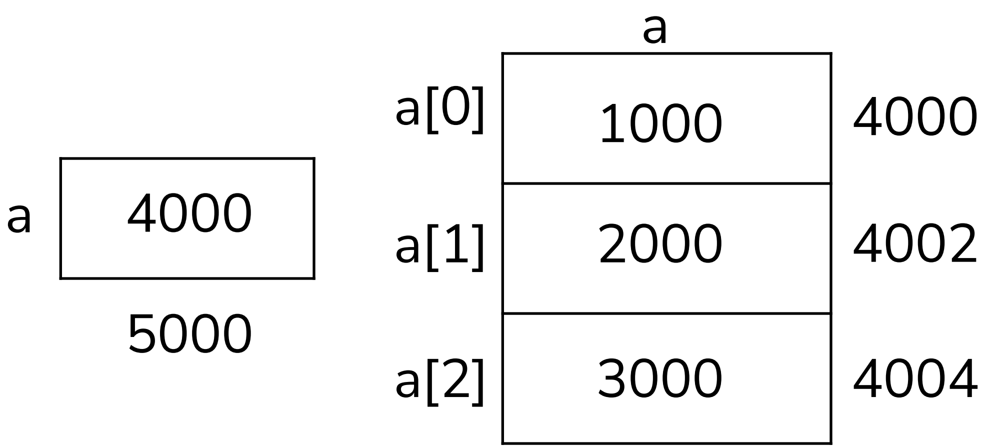
- Returns undefined value

2D Array



```
#include<stdio.h>
int main()
{
   int a[2][3]; // Array Declaration
   int r, c;
   scanf("%d%d", &r, &c); // 2 2
   for(int i = 0; i < r; i++)
   {
      for(int j = 0; j < c; j++)
      {
         cin >> &a[i][j];
      }
   }
   for(int i = 0; i < r; i++)
   {
       for(int j = 0; j < c; j++)
       {
           cout << a[i][j];
       }
       cout << endl;
    }
}
```

Memory Allocation



a 4000

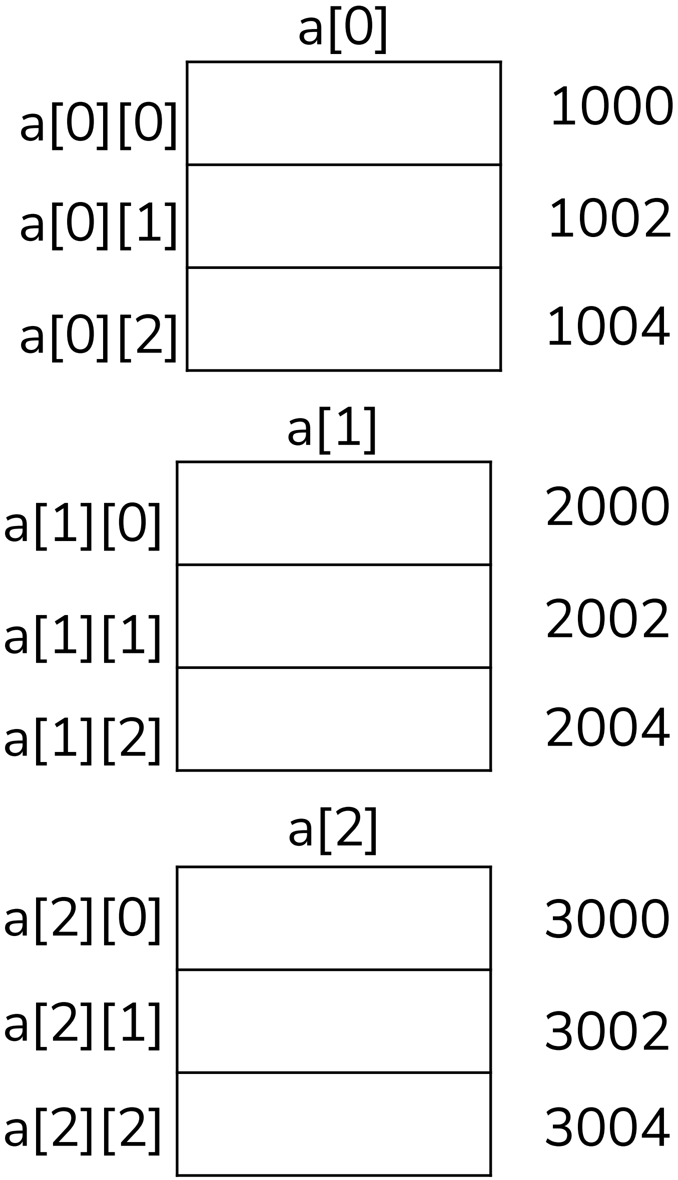
5000

a

a[0] 1000 4000

a[1] 2000 4002

a[2] 3000 4004



a[0]

a[0][0] 1000

a[0][1] 1002

a[0][2] 1004

a[1]

a[1][0] 2000

a[1][1] 2002

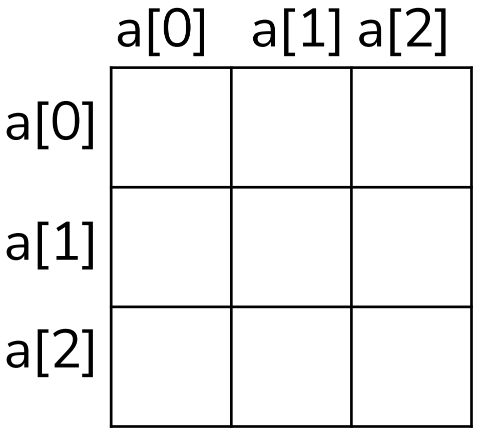
a[1][2] 2004

a[2]

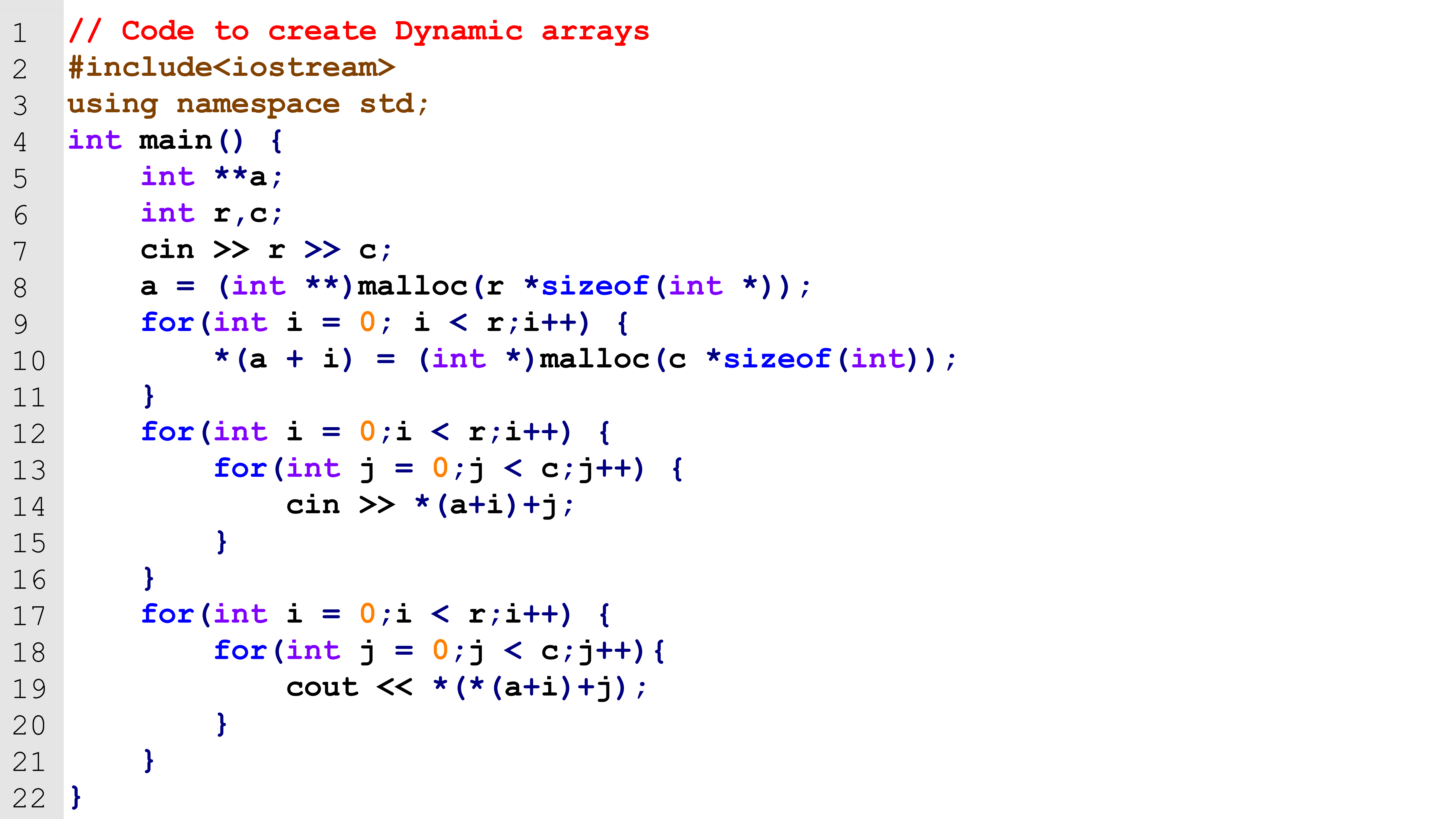
a[2][0] 3000

a[2][1] 3002

a[2][2] 3004



	a[0]	a[1]	a[2]
a[0]			
a[1]			
a[2]			



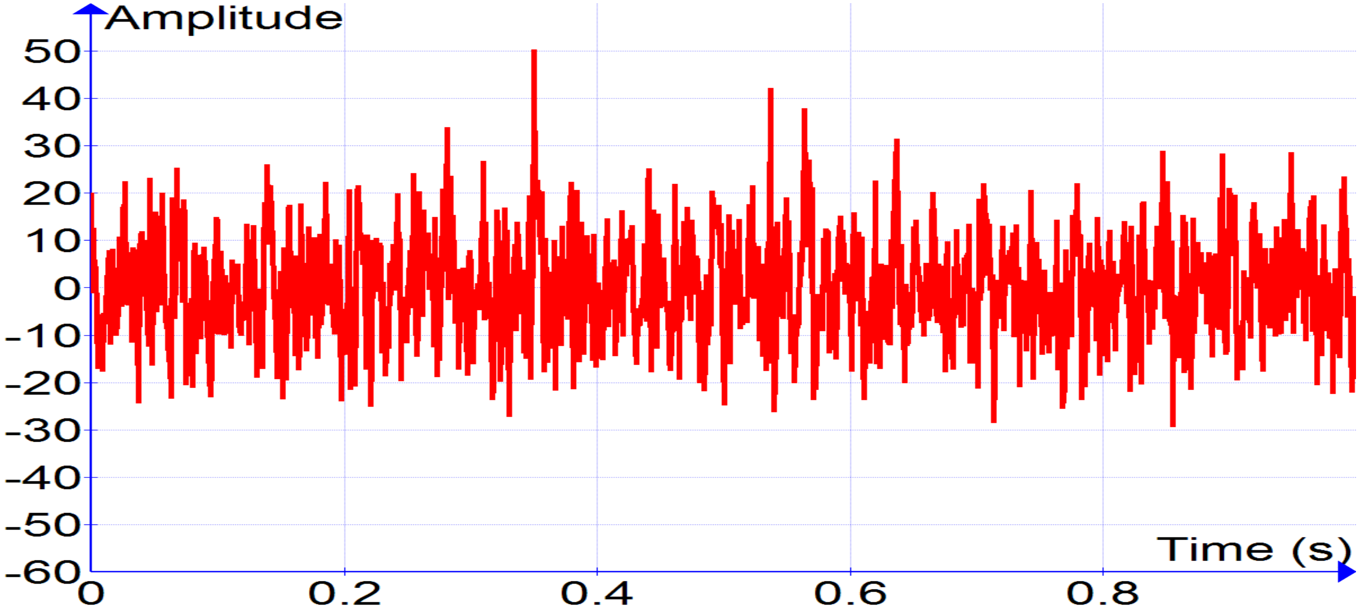
```
// Code to create Dynamic arrays
#include<iostream>
using namespace std;
int main() {
    int **a;
    int r,c;
    cin >> r >> c;
    a = (int **)malloc(r *sizeof(int *));
    for(int i = 0; i < r;i++) {
        *(a + i) = (int *)malloc(c *sizeof(int));
    }
    for(int i = 0;i < r;i++) {
        for(int j = 0;j < c;j++) {
            cin >> *(a+i)+j;
        }
    }
    for(int i = 0;i < r;i++) {
        for(int j = 0;j < c;j++){
            cout << *(*(a+i)+j);
        }
    }
}
```

Applications

Image Processing

Speech Processing

Applications

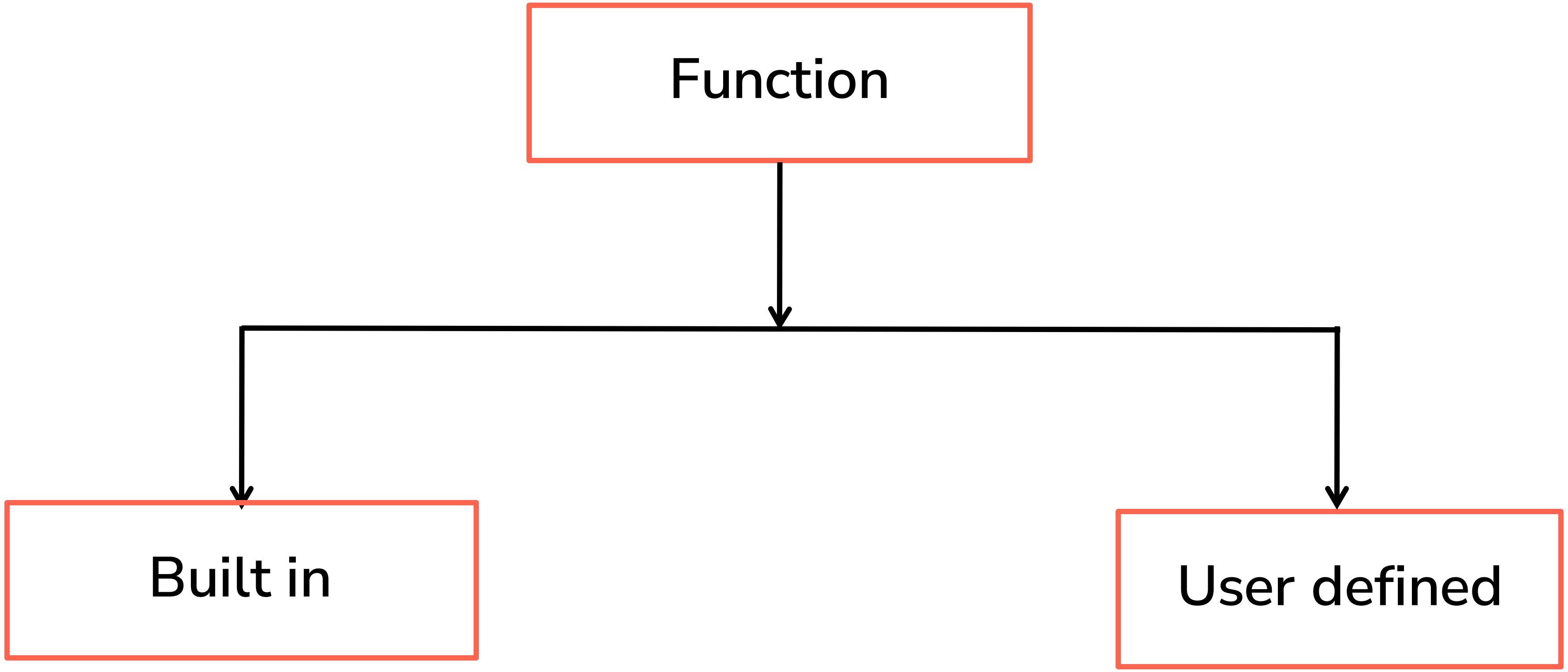


Signal Processing

Functions

- Divide the program into subtasks
- Reduce the number of lines
- Easy to read
- Reduce development cost and time

Functions



Function

Built in

User defined

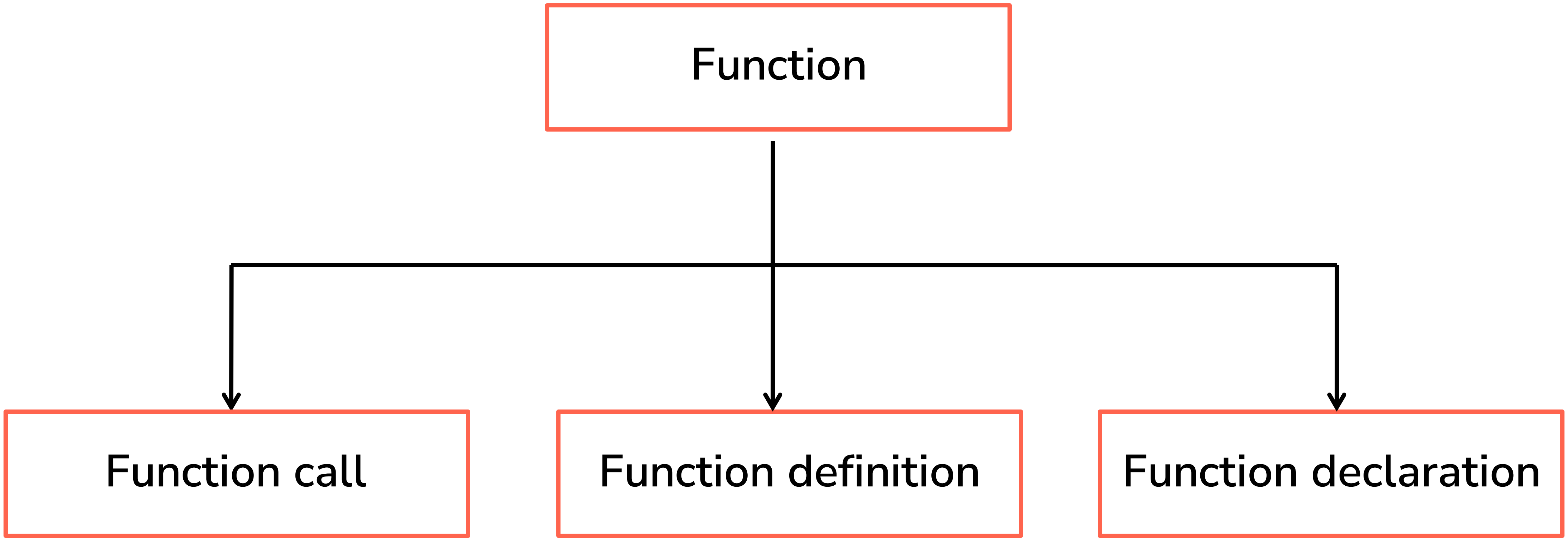
Built-in functions

- Also known as library functions
- Need not to declare as they are defined in libraries
- We can directly call them
  - ✓ pow(x,y), built-in function which is x to the power y.
  - ✓ This function is declared in cmath header file.

User-defined functions

- The functions that we declare and write in our programs.
- It groups code to perform a specific task and that group of code is given a name(identifier).
- When the function is invoked from any part of program, it all executes the codes defined in body of function.

Functions



Function

Function call

Function definition

Function declaration

```
#include<iostream>
using namespace std;

int addition (int, int);

int main()
{
        int x = 10, y = 20;
        int res = addition(x,y);
        cout << “result = ” << res;
        return 0;
}

int addition (int a, int b)
{
        int sum = a + b;
        return sum;
}
```

comment

Function declaration

Function call

Function definition

```
#include<iostream>
using namespace std;

int addition (int, int);

int main()
{
        int x = 10, y = 20;
        int res = addition(x,y);
        cout << ‟result = ‟ << res;
        return 0;
}

int addition (int a, int b)
{
        int sum = a + b;
        return sum;
}
```

comment

Function declaration

- It tells the compiler about a function name and how to call the function.
- The actual body of the function can be defined separately.

```
#include<iostream>
using namespace std;

int addition (int, int);

int main()
{
        int x = 10, y = 20;
        int res = addition(x,y);
        cout << “result = ” << res;
        return 0;
}

int addition (int a, int b)
{
        int sum = a + b;
        return sum;
}
```

comment

Function call

- Pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

- **x & y** are actual parameters

- **a & b** are formal parameters

```cpp
#include<iostream>
using namespace std;

int addition (int, int);

int main()
{
      int x = 10, y = 20;
      int res = addition(x,y);
      cout << “result = “ << res;
      return 0;
}

int addition (int a, int b)
{
      int sum = a + b;
      return sum;
}



```

comment

Function definition

- The function body contains a collection of statements that define what the function does.

Types of user-defined functions

- No argument and no return value
- No argument but return value
- With argument but no return value
- With argument and return value

```
#include<iostream>
using namespace std;

void greatNum();

int main()
{
    greatNum();
    return 0;
}

void greatNum()
{
    int x, y;
    cin >> x >> y;
    if(x > y)
      cout <<“The greater number is”<< x;
    else
      cout <<“The greater number is”<< y;
}
```

comment

With no arguments and no return value

```
#include<iostream>
using namespace std;

int greatNum();

int main()
{
        int a = greatNum();
        cout <<“The greater number is”<< a;
        return 0;
}

int greatNum()
{
        int x, y, greaterNum;
        cin >> x >> y;
        if(x > y)
                greaterNum = x;
        else
                greaterNum = y;
        return greaterNum;
}
```

comment

With no arguments and a return value

```
#include<iostream>
using namespace std;

void greatNum(int, int);

int main()
{
    int x, y;
    cin >> x >> y;
    greatNum(x, y);
    return 0;
}

void greatNum(int x, int y)
{
    if(x > y)
      cout <<“The greater number is”<< x;
    else
      cout <<“The greater number is”<< y;
}
```

comment

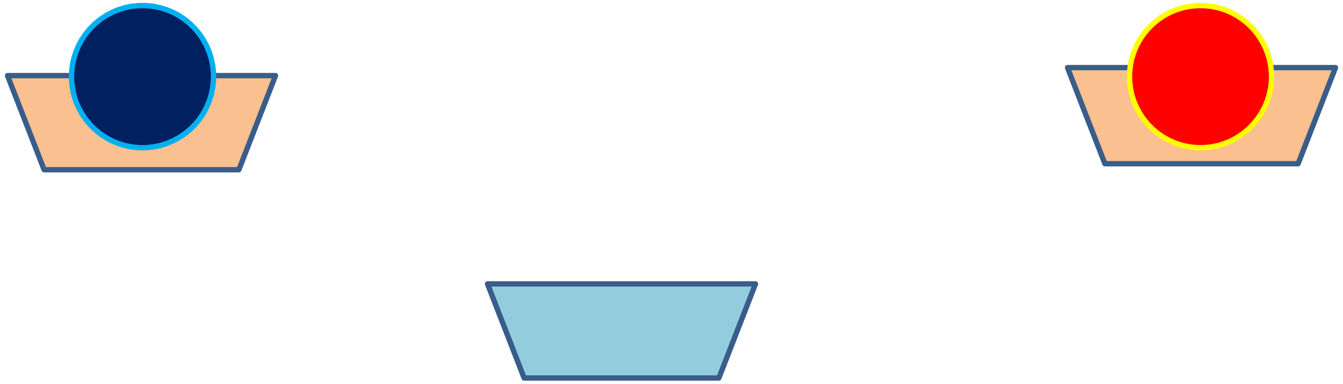
With arguments and no return value

```
#include<iostream>
using namespace std;

int greatNum(int , int);

int main()
{
        int x, y;
        cin >> x >> y;
        int a = greatNum(x, y);
        cout <<“The greater number is”<< a;
        return 0;
}

int greatNum(int x, int y)
{
        if(x > y)
                return x;
        else
                return y;
}

```

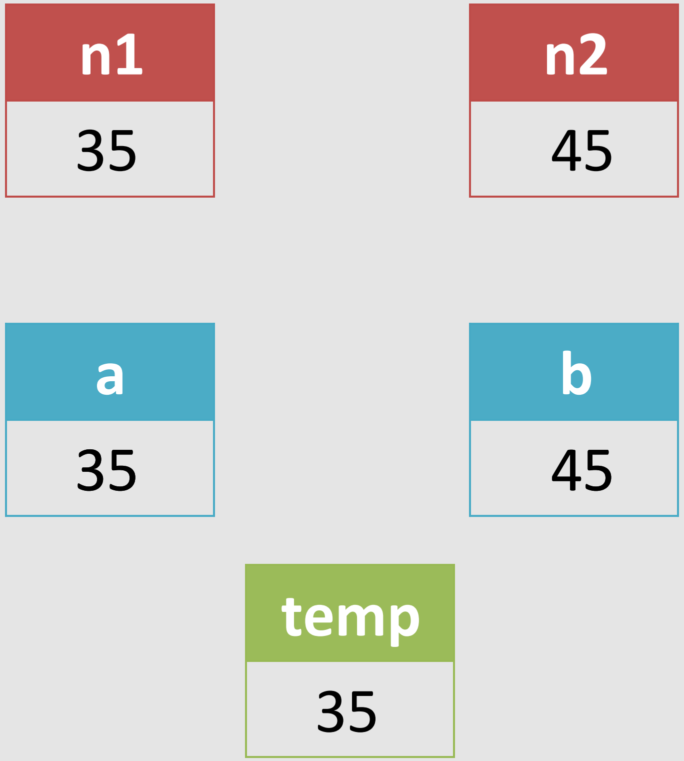
comment

With arguments and a return value

Swapping two Numbers



```
// Code to swap two numbers
#include<iostream>
using namespace std;

void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
int main()
{
    int n1 = 35, n2 = 45;
    cout <<"Before : “<< n1 << “,” << n2;
    swap(n1,n2);
    cout <<"After : “<< n1 “,” << n2;
    return 0;
}
```



n1 35

n2 45

a 35

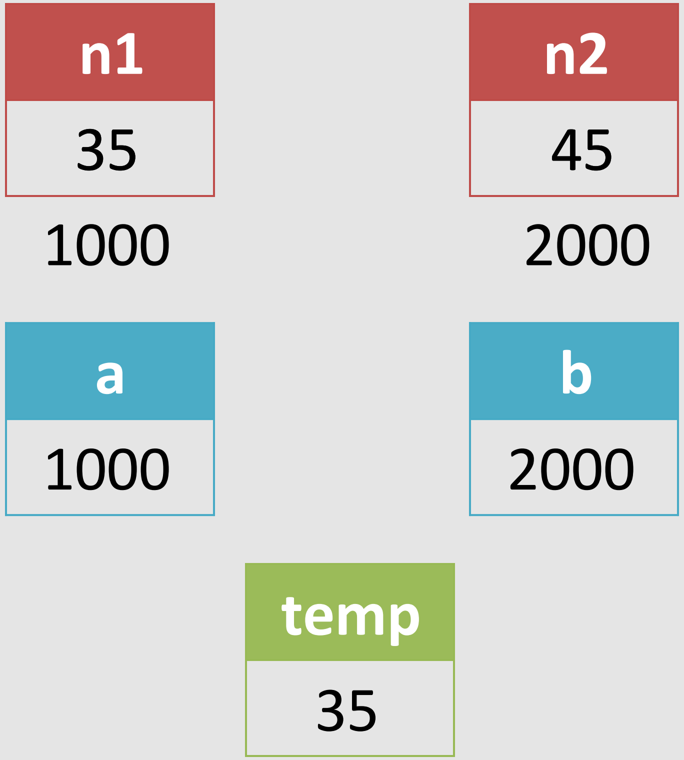
b 45

temp 35

Output:

```
Before : 35, 45
After : 35, 45
```

```cpp
// Code to swap two numbers
#include<iostream>
using namespace std;
void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
int main()
{
    int n1 = 35, n2 = 45;
    cout <<"Before : “<< n1 << “,” << n2;
    swap(&n1,&n2);
    cout <<"After : “<< n1 “,” << n2;
    return 0;
}
```



n1
35
1000

n2
45
2000

a
1000

b
2000

temp
35

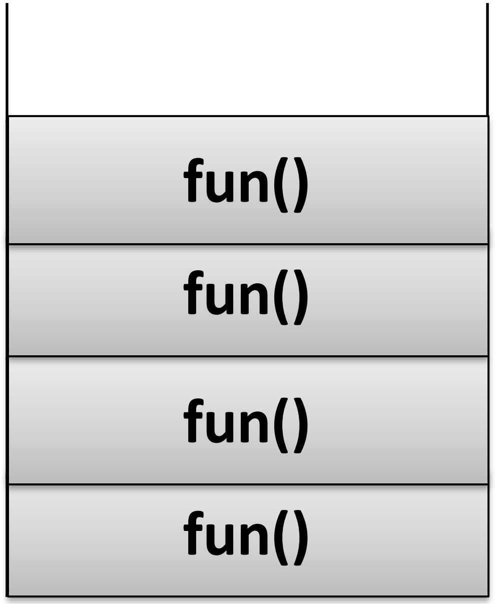
```
Output:

Before : 35, 45
After : 45, 35
```

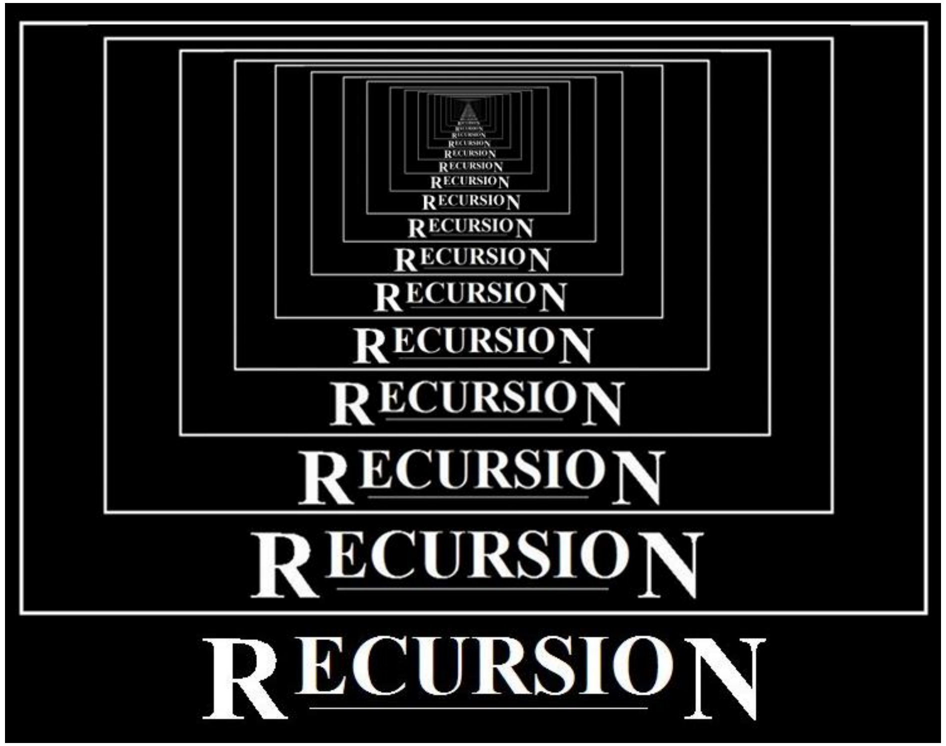
Recursion

- Reduce unnecessary calling of function
- To solve problems in easy way
- To reduce the code size
- **Example:** Tower of Hanoi

What is Recursion

A function which calls a copy of itself again and again





Recursion

Problem: finding out the number of layers in an onion.

Simple solution: As each layer is peeled off, layer-count is incremented by 1

Peeling the top layer reduces the problem to peeling (n-1) layers

$\text{onion\_peel\_count}(n) = 1 + \boxed{\text{onion\_peel\_count}(n-1)}$

$= 1 + \quad 1 + \boxed{\text{onion\_peel\_count}(n-2)}$

$= 1 + 1 + \quad \ldots \quad + \text{onion\_peel\_count}(0)$

Applying the procedure again

W.K.T onion_peel_count(0) = 0

onion_peel_count(n) = 1 + 1 + ...+ 1

This is recursion - repeatedly applying a procedure

Recursion

Peeling the top layer reduces the problem to peeling (n-1) layers

$$\text{onion\_peel\_count}(n) = \boxed{1 + \text{onion\_peel\_count}(n-1)}$$

$$= 1 + \quad 1 + \text{onion\_peel\_count}(n-2)$$

$$= 1 + 1 + \quad \dots \quad + \text{onion\_peel\_count}(0)$$

W.K.T $\boxed{\text{onion\_peel\_count}(0) = 0}$

$\text{onion\_peel\_count}(n) = 1 + 1 + \dots + 1$

Base Condition

Problem reduction in each step

Mathematically

$$\text{onion\_peel\_count}(n) = \begin{cases} 0 & n = 0 \\ 1 + \text{onion\_peel\_count}(n-1) & \text{Otherwise} \end{cases}$$

Recursion

Mathematically

$$\text{onion\_peel\_count}(n) = \begin{cases} 0 & n = 0 \\ 1 + \text{onion\_peel\_count}(n-1) & \text{Otherwise} \end{cases}$$

```
int onion_peel_count(int n) {
        if(n == 0)
                return 0;
        else
                return 1+onion_peel_count(n-1)
}
```

How should you proceed ?

Question 1

Write a program to find the sum of n natural numbers using recursion.

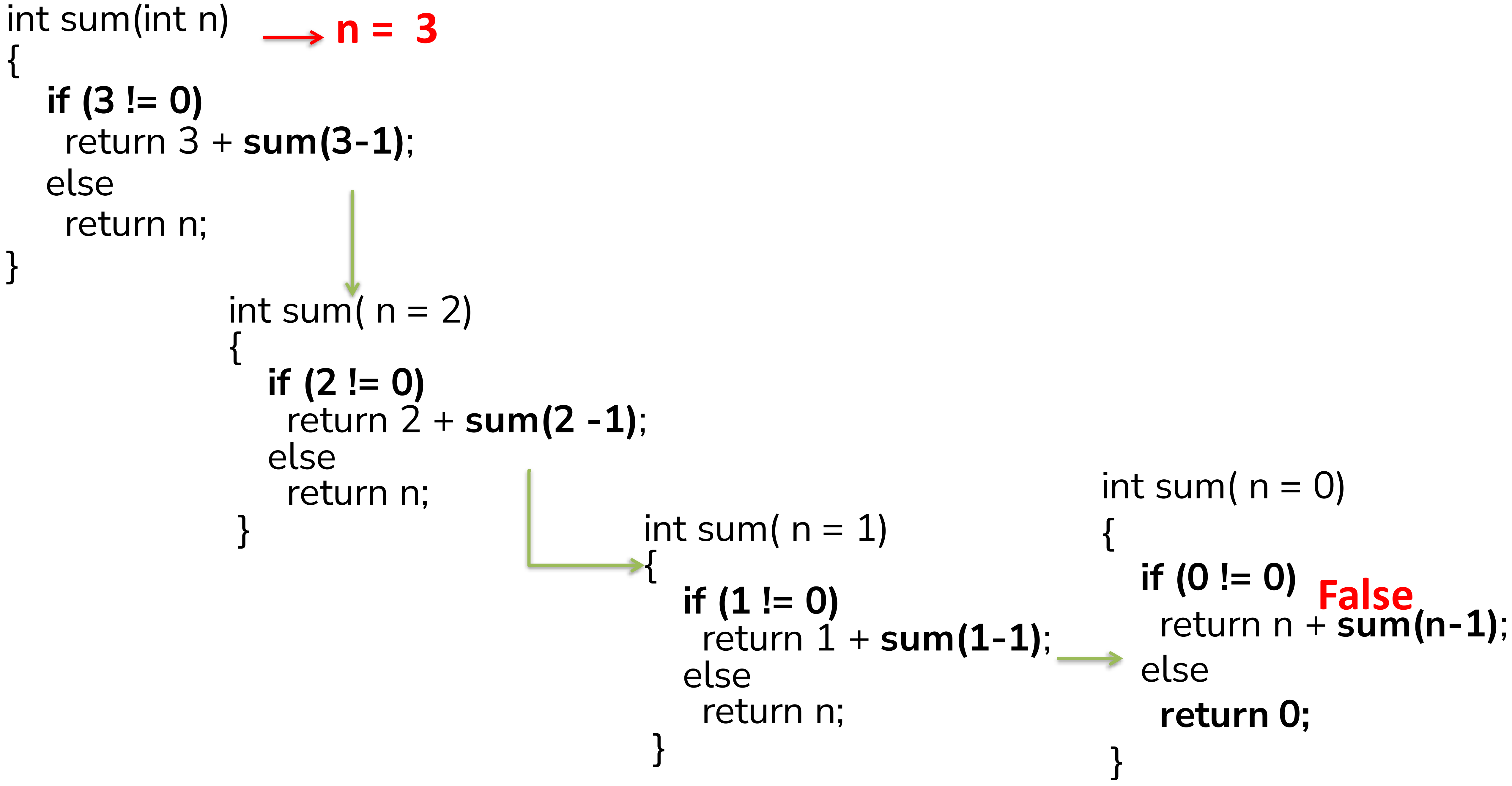
Sample Input:

3

Sample Output:

6

```cpp
#include<iostream>
using namespace std;

int main()
{
        int n, result;
        cin >> n;
        result = sum(n);
        return 0;
}

int sum(int n)
{
        if(n != 0)
        {
                return n + sum(n-1);
        }
        else
        {
                return n;
        }
}
```



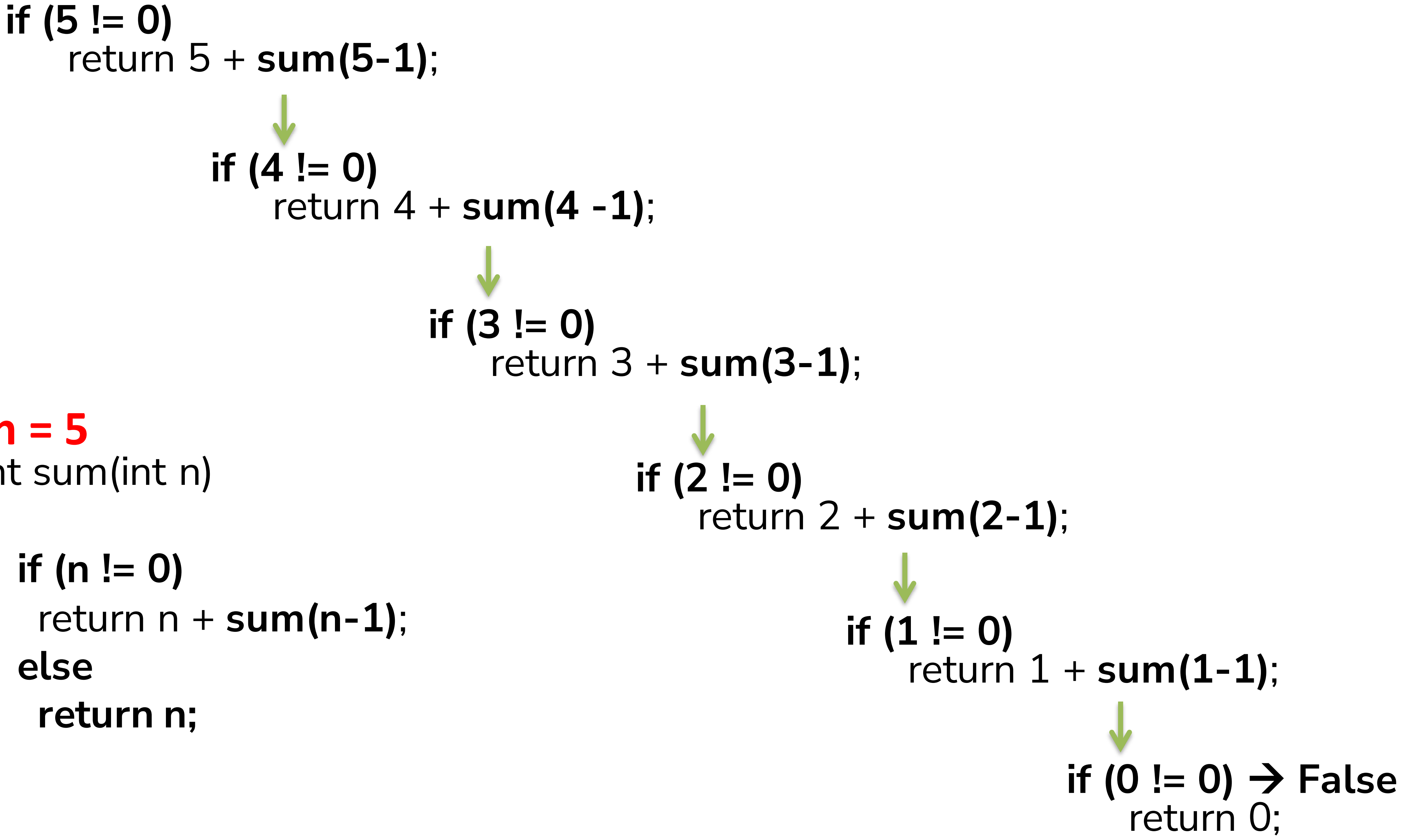
```
int sum(int n)        → n =  3
{
   if (3 != 0)
     return 3 + sum(3-1);
   else
     return n;
}
```

```
int sum( n = 2)
{
   if (2 != 0)
     return 2 + sum(2 -1);
   else
     return n;
}
```

```
int sum( n = 1)
{
   if (1 != 0)
     return 1 + sum(1-1);
   else
     return n;
}
```

```
int sum( n = 0)
{
   if (0 != 0)  False
     return n + sum(n-1);
   else
     return 0;
}
```

```
int sum(3)
{
    if (3 != 0)
      return 6;
    else
      return n;
}
```



```
if (5 != 0)
    return 5 + sum(5-1);
            ↓
            if (4 != 0)
                return 4 + sum(4 -1);
                        ↓
                        if (3 != 0)
                            return 3 + sum(3-1);
                                    ↓
                                    if (2 != 0)
                                        return 2 + sum(2-1);
                                                ↓
                                                if (1 != 0)
                                                    return 1 + sum(1-1);
                                                            ↓
                                                            if (0 != 0) → False
                                                                return 0;
```

n = 5

```
int sum(int n)
{
  if (n != 0)
    return n + sum(n-1);
  else
    return n;
}
```

```
if (5 != 0)
   return 5 + sum(4);
              ↕
          if (4 != 0)
             return 4 + sum(3);
                        ↕
                    if (3 != 0)
                       return 3 + sum(2);
                                  ↕
                              if (2 != 0)
                                 return 2 + sum(1);
                                            ↕
                                        if (1 != 0)
                                           return 1 + sum(0);
                                                      ↕
                                                  if (0 != 0) → False
                                                     return 0;
```

```
int sum(5)
{
  if (n != 0)
    return n + sum(n-1);
  else
    return n;
}
```

```
#include<iostream>
using namespace std;

int sum(int n);
int main()
{
    int n, result;
    cin >> n;
    result = 15 sum(5);
    cout << “sum = ‟ << result;
}
```

```
int sum( 5 )
{
    if (5 != 0)
      return 15;
    else
      return n;
 }
```

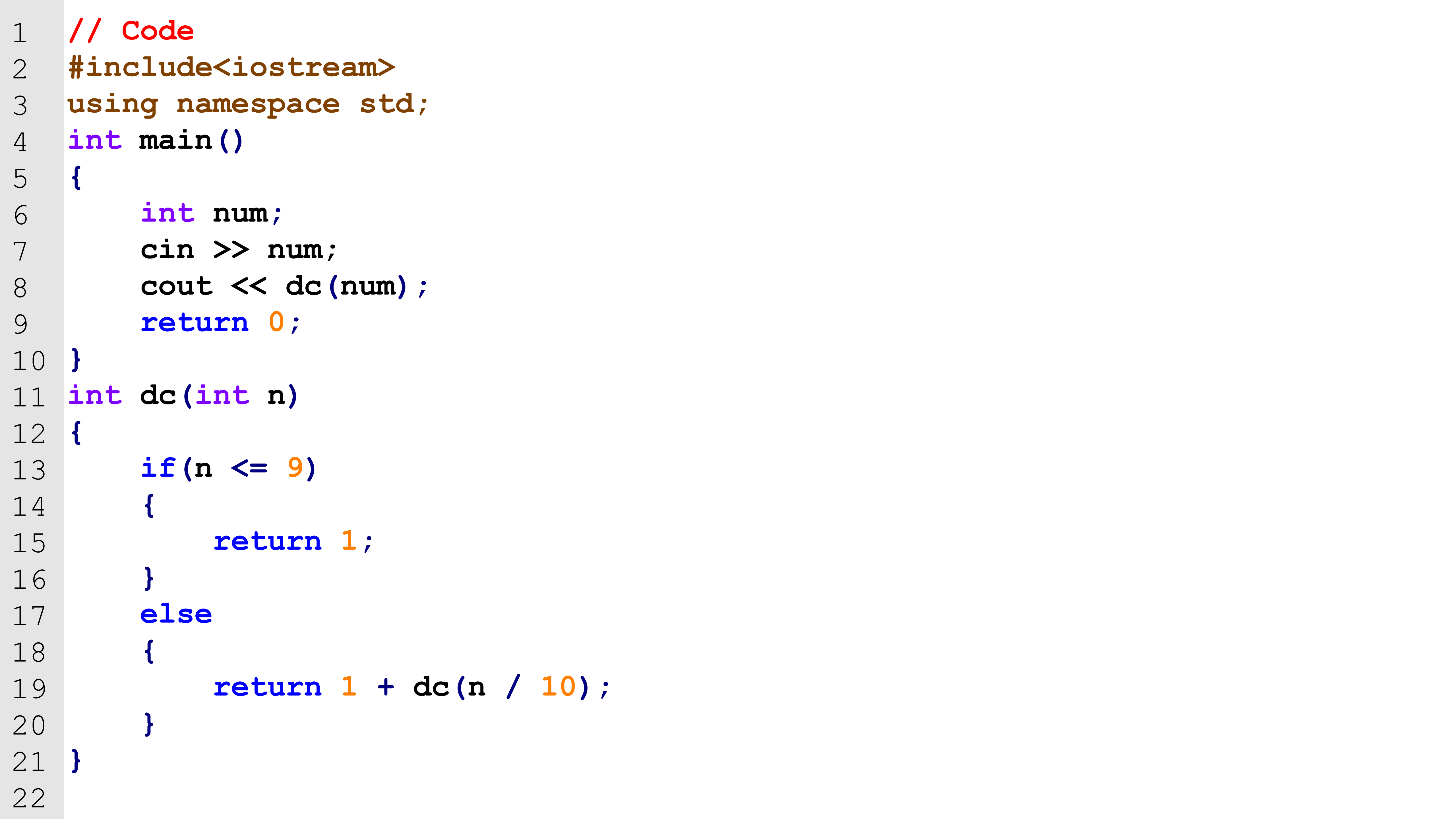
Compute Output

$$dc(n) = \begin{cases} \boxed{1} & n <= 9 \\ 1 + dc(n/10) & \text{Otherwise} \end{cases}$$

n	recursive_feeling_1 return value
98	2
784	3
47896	5

Intention of the Qn is ??

Mistake??



```cpp
// Code
#include<iostream>
using namespace std;
int main()
{
    int num;
    cin >> num;
    cout << dc(num);
    return 0;
}
int dc(int n)
{
    if(n <= 9)
    {
        return 1;
    }
    else
    {
        return 1 + dc(n / 10);
    }
}
```

Question 2

Print the digits from left to right using recursion.

Sample Input:

```
456
```

Sample Output:

```
4
5
6
```

Equation

$$\text{print\_digits}(n) = \begin{cases} \text{print } n & n <= 9 \\ \text{print\_digits}(n/10) & \text{Otherwise} \\ \text{print } (n \ \% \ 10) & \end{cases}$$

```cpp
#include<iostream>
using namespace std;
int print_digits(int n);
int main()
{
    int num;
    cin >> num;
    print_digits(num);
    return 0;
}
int print_digits(int n)
{
    if(n <= 9)
    {
        cout << n;
    }
    print_digits(n / 10);
    cout << n % 10 << endl;
}
```

```
Sample Input
456

Actual Output
40000000000000000.....

Expected Output:
4
5
6
```

```cpp
#include<stdio.h>
using namespace std;
int print_digits(int n);
int main()
{
    int num;
    cin >> num;
    print_digits(num);
    return 0;
}
int print_digits(int n)
{
    if(n <= 9)
    {
        cout << n;
        return;
    }
    print_digits(n / 10);
    cout << n % 10 << endl;
}
```

```
Sample Input
456

Sample Output
4
5
6
```

return (jump statement)

To terminate the execution of the function and returns the control to the calling function

Syntax :

```
return;
```

Question 1

Can any function call itself?

A) Yes

B) No

C) Compilation Error

D) Runtime Error

Strings in C

- In C, a string can be a specially terminated char array or char pointer
  - ✓ a char array, such as char str[ ]=“high”;
  - ✓ a char pointer, such as char *p = “high”;
- If a char array, the last element of the array must be equal to ‘\0’, signaling the end
- For example, the above str[] is really of length 5:
  - ✓ str[0]=‘h’ str[1]=‘i’ str[2]=‘g’ str[3]=‘h’ str[4]=‘\0’

Strings in C

- The same array could’ve been declared as:
  - ✓ char str[5] = {‘h’,’i’, ‘g’,’h’,’\0’};
- If you write char str[4] = {‘h’,’i’, ‘g’,’h’};, then str is an array of chars but not a string.
- In char *p=“high”; the system allocates memory of 5 characters long, stores “high” in the first 4, and ‘\0’ in the 5th.

Strings in C++

- C++ has a <string> library
- Include it in your programs when you wish to use strings: #include <string>
- In this library, a class string is defined and implemented
- It is very convenient and makes string processing easier than in C

Strings in C++

- Not necessarily null terminated
- String is not a pointer, but a class
- Many member functions take start position and length
  - ✓ If length argument is too large, max chosen

Creating string objects

```
#include <string>

string s;  //s contains 0 characters
string s1( "Hello" );        //s1 contains 5 characters
string s2 = “Hello”;        //s2 contains 5 characters
                             //implicitly calls the constructor
string s3( 8, 'x' );        //s3 contains 8  'x' characters
string s4 = s3;          //s4 contains 8  'x' characters
string s5(s2, 3, 2);  //s5 copies a substring of s2; it contains ”lo”
```

String I/O

- String can be input using the extraction operator >>, but one or more white spaces indicates the end of an input string.

```
char A_string[80], E_string[80];
cout << "Enter some words in a string:\n";
cin >> A_string >> E_string;
cout << A_string << E_string  << “\n END OF OUTPUT \n";
```

getline

- The function getline can be used to read an entire line of input into a string variable.
- The getline function has three parameters:
  - ✓ The first specifies the area into which the string is to be read.
  - ✓ The second specifies the maximum number of characters, including the string delimiter.
  - ✓ The third specifies an optional terminating character. If not included, getline stops at ‘\n’.

```
#include<iostream>
using namespace std;

int main()
{
      char A_string[80];
      cout << “Enter some words in a string:\n” ;
      cin.getline(A_string, 80);
      cout << A_string;
      return 0;
}
```

Output

```
Enter some words in a string:
This is a test.

This is a test.
```

Manipulate Null-terminated strings

C++ supports a wide range of functions that manipulate null-terminated strings

- strcpy(str1, str2)
  - ✓ Copies string str2 into string str1.
- strcat(str1, str2)
  - ✓ Concatenates string str2 onto the end of string str1.
- strlen(str1)
  - ✓ Returns the length of string str1.

Manipulate Null-terminated strings

- strcmp(str1, str2)
  - ✓ Returns 0 if str1 and str2 are the same; less than 0 if str1 < str2; greater than 0 if str1 > str2.
- strchr(str1, ch)
  - ✓ Returns a pointer to the first occurrence of character ch in string str1.
- strstr(str1, str2)
  - ✓ Returns a pointer to the first occurrence of string str2 in string str1.

Functions supported by string class

- append()
  - ✓ This functions appends a part of a string to another string
- assign()
  - ✓ This functions assigns a partial string
- at()
  - ✓ This function obtains the character stored at a specified location
- end()
  - ✓ This function returns a reference to the end of the string

Functions supported by string class

- begin()
  - ✓ This function returns a reference to the start of the string
- capacity()
  - ✓ This function gives the total element that can be stored
- compare()
  - ✓ This function compares a string against the invoking string
- empty()
  - ✓ This function returns true if the string is empty

Functions supported by string class

- erase()
  - ✓ This function removes character as specified
- find()
  - ✓ This function searches for the occurrence of a specified substring
- length()
  - ✓ It gives the size of a string or the number of elements of a string
- swap()
  - ✓ This function swaps the given string with the invoking one

Operators used for string objects

- = assignment
- + concatenation
- == Equality
- != Inequality
- <= Less than or equal
- >= Greater than or equal
- << Output
- >> Input

Void pointers

- Pointer which represents absence of data type.
- Void pointers have great flexibility as it can point to any data type
- Cannot be dereferenced

```
#include<iostream>
using namespace std;

int main()
{
	int x = 10;
	void *ptr;
	int *pt;
	ptr = &x;
	pt = ptr;
	cout <<  *pt;
	return 0;
}
```

Output

```
Error : invalid conversion from ‘void*’ to ‘int*’
Pt = ptr;
```

```cpp
#include<iostream>
using namespace std;

int main()
{
      int x = 10;
      void *ptr;
      int *pt;
      ptr = &x;
      pt = (int*)ptr;
      cout <<  *pt;
      return 0;
}
```

Output

```
10
```

Invalid pointers

- Pointer should point to valid address but not necessarily to valid elements.
- Uninitialized pointers are invalid pointers.

```
int *ptr;
int a[10];
int *p = a + 20
```

NULL pointers

- Pointer which point nowhere and not an invalid pointers.
- Following are 2 ways to assign NULL pointer

```
int *ptr = 0;
int *p = NULL;
```

Dynamic Memory Allocation

- It refers to performing memory allocation manually by programmer
- It creates memory at run time
- Dynamically allocated memory is allocated on heap
- Non-static and local variable get allocated on stack

```
// Using new to allocate storage

int * ptr = NULL;

ptr = new int;   // allocate an integer on the heap

cout << ptr << endl;   // address will be printed

cout << *ptr << endl; // garbage value will be printed

*ptr = 100;

cout << ptr << endl;  // 100 will be printed
```

Applications

- One use is to allocate memory of variable size which is not possible with compiler allocated memory except variable length arrays.
- We are free to allocate and deallocate memory whenever we need.

Memory leak

- For normal variables int a, char b, etc., memory is automatically allocated and deallocated
- For dynamically allocated memory like int *p = new int[10], it is programmers responsibility to deallocate memory when not needed
- If programmer doesn’t deallocated memory, it causes memory leak

new operator

- The new operator denotes a request for memory allocation on heap
- If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated
- syntax
  - ✓ int *p = new int(20);
  - ✓ int *q = new int [20];

delete operator

- It is programmer’s responsibility to deallocate dynamically allocated memory
- They are provided with delete operator
- syntax
  - ✓ delete p;

```
#include<iostream>
using namespace std;

int main()
{
        double *value = NULL; // pointer initialized with null
        value = new double;   // request memory for the variable
        *value = 29387.38;    // stores value at allocated address
        cout << “Value is “<< *value;
        delete value;         // free up the memory
        return 0;
}
```

Preprocessor

- The preprocessor include the instructions for the compiler
- These instructions are executed before the source code is compiled
- Preprocessor directives begin with hash sign(#)
- No semicolon(;) is expected at the end of a preprocessor directive

#define preprocessor directive

- It is used to define constant values in program

```
#include<iostream>
using namespace std;
#define PI 3.14

int main()
{
      int r;
      float a;
      cin >> r;
      cout << PI * r * r;
      return 0;
}
```

#include preprocessor directive

- It is used to include another source file in our source code
- syntax
  - ✓ #include “filename.h”
  - ✓ #include<filename.h>

Macros

- Macros are replaced with macro expression
- Macros runs programs faster but increase the program size
- It is better to use macros, when definition is smaller in size

Functions

- In function call, the control is passed to a function definition along with arguments, and definition is processed and value may return to call
- Functions runs programs slower but decrease the program size
- It is better to use functions, when definition is bigger in size

Question 1

Predict the output:

```
#include <iostream>
using namespace std;
#define a 10
int main()
{
    int a = 5;
    cout << "macro variable value: "<< a;
    return 1;
}
```

Question 1

A) 10

B) 5

C) Error

D) 50

Question 2

Predict the output:

```
#include<iostream>
using namespace std;
#define sqrt(x) (x*x)
int main()
{
    int a = 3, b;
    b = sqrt(a + 5);
    cout << b;
}
```

Question 2

A) 4

B) 20

C) 14

D) 23

Question 3

Predict the output:

```
#include < iostream >
using namespace std;
#define MIN(a,b) (((a)<(b)) ? a : b)
int main ()
{
      float i, j;
      i = 100.1;
      j = 100.01;
      cout << "The minimum is "  << MIN(i, j) << endl;
      return 0;
}
```

Question 3

A) 100.01

B) 100.1

C) 100.00

D) Compile time error

Question 4

Predict the output:

```
#include <iostream>
using namespace std;
#define SquareOf(x) x * x
int main()
{
    int x;
    cout << SquareOf(x + 4);
    return 0;
}
```

Question 4

A) 64

B) 16

C) 4

D) Compile time error

Question 5

What is the another name of the macro?

A) link directive

B) executed directive

C) scripted directive

D) executed & link directive