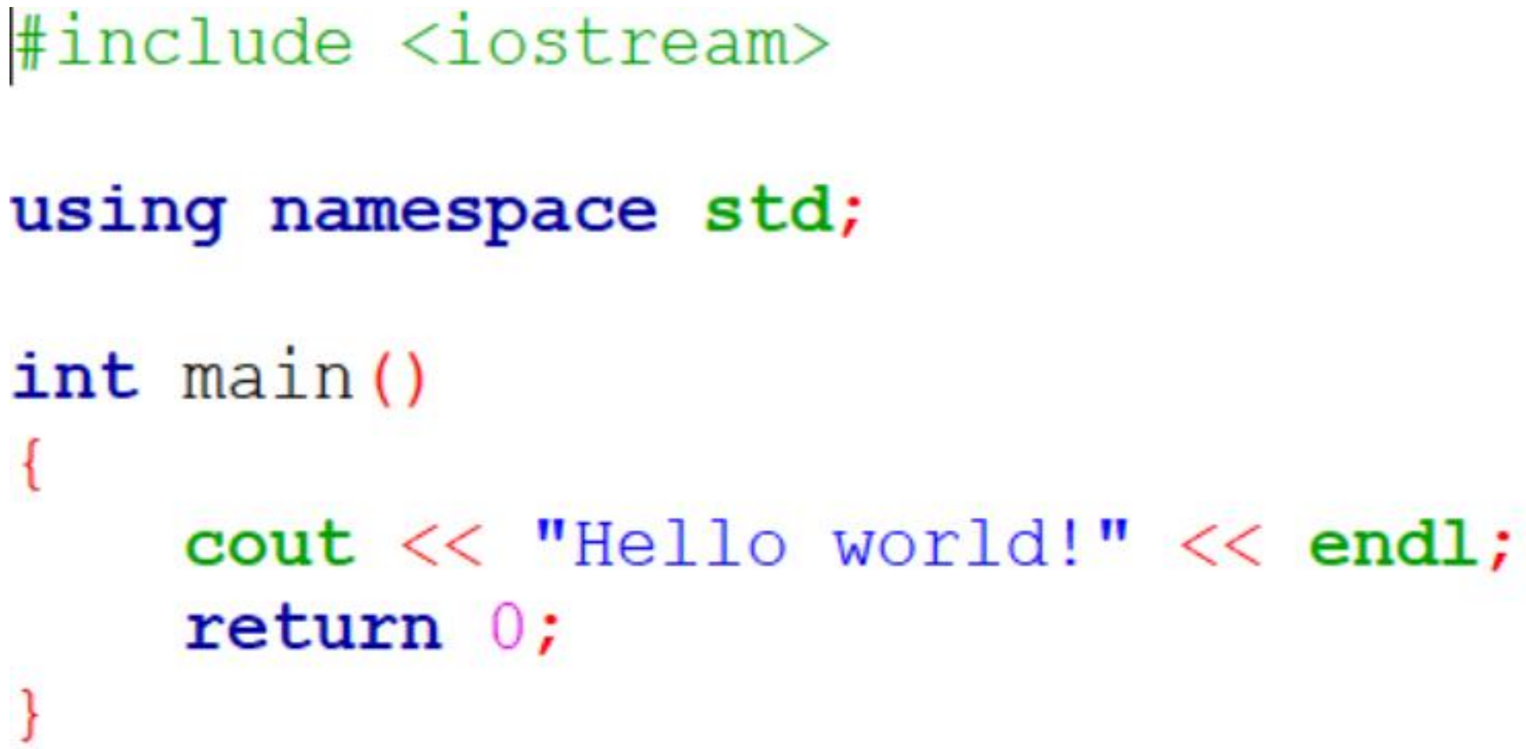
CSE2001 OBJECT ORIENTED PROGRAMMING WITH C++

Shreyasi

STRUCTURE OF C++ PROGRAM



```cpp
#include <iostream>

using namespace std;

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

CONSTANTS

- Their value does not change during the processing of all the instructions in a solution.
- They can be any type of data – numeric, alphabetical, or special symbols.
- We can declare the constants and store them in the memory.
- E.g. PI is a constant, its value cannot be changed.

VARIABLES

- Their values may change during the execution of a program.
- They are also referred to as identifiers.
- An identifier is a series of characters consisting of letters, digits and underscore(_) that does not begin with a digit.

NAMING CONVENTIONS

- Name a variable for what it represents. E.g. incomeTax for income tax.
- You cannot use spaces in the variable names.
- Start a variable name with a letter. It can be followed by digits.
- Do not use a symbol that is used as a mathematical operator.
- Follow a standard naming convention. For named constants, you can use all upper case characters.

INTEGER DATA TYPES

Data Type	Memory (Bytes)	Range
short int	2	-32768 to 32767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2147483648 to 2147483647
long	8	-9223372036854775808 to 9223372036854775807

FLOATING-POINT TYPES

TYPE	Memory(bytes)	APPROX MIN VALUE	APPROX MAX VALUE
float	4	$1.2 * 10^{-38}$	$3.4 * 10^{38}$
double	8	$2.3 * 10^{-308}$	$1.7 * 10^{308}$

TEXT AND BOOLEAN TYPES

TYPE	ALLOWED VALUES
char	Single character
bool	Boolean value, true or false
string	A sequence of characters

ADDING COMMENTS

- Comments enable you to add a description to your code, which can serve as reminders for what the code is trying to accomplish.
- There are two ways of adding comments.
- Single line comments can be added by prefixing the statement with //
- Multiple line comments can be enclosed within /* and */

ESCAPE SEQUENCES

ESCAPE SEQUENCE	CHARACTER PRODUCED
\’	Single quotation mark
\”	Double quotation mark
\\	Backslash
\b	Backspace
\n	New line
\t	Horizontal tab
\v	Vertical tab
\a	Alert

OPERATORS

- They are the data connectors within expressions and equations.
- They tell the computer how to process the data.
- Different types of operators used – mathematical, relational, and logical operators.
- Operands are the data that the operator connects and processes. E.g. in 2 + 4, + is the operator, 2 and 4 are the operands, and 6 is the resultant.

OPERATOR PRECEDENCE

PRECEDENCE	OPERATORS
Highest	() ++, -- +,- (unary)
	*,/,%
	+,-
	=,*=,/=,+=,-=
Lowest	,(Comma)

EXPRESSIONS

- C++ contains a number of operators.
- By combining the operators with variables and literal values, you can create expressions to perform computations.
- An **expression** processes data, the operands through the use of operators. E.g.

$$grossSalary * taxRate$$

- An equation stores the resultant of an expression in a memory location in the computer through the equal (=) sign. E.g.

$$netSalary = grossSalary * taxRate$$

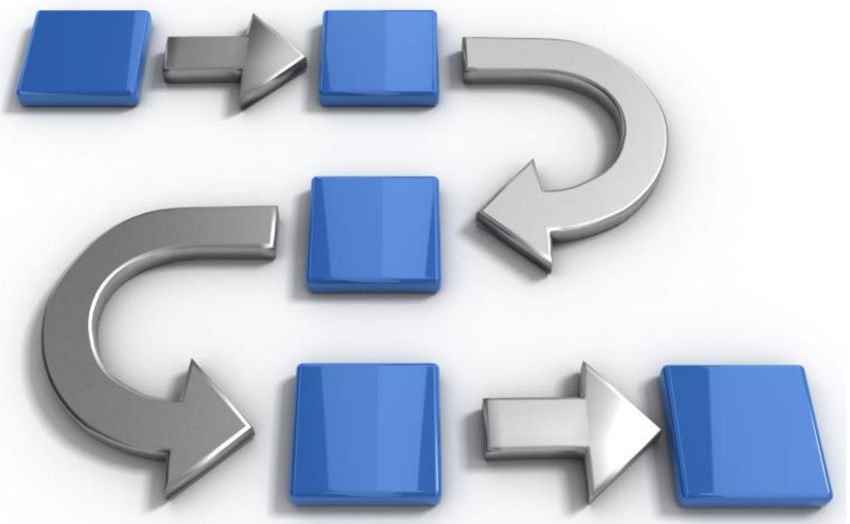
FUNCTIONS

- They are small set of instructions that perform specific tasks and return values.
- Each language has a set of functions within it.
- Most of the languages allow programmers to write their own functions too.
- Data can be listed as part of the function in the form of **parameters**.

TYPES OF ERRORS

- There are two types of errors –syntax errors and logical errors.
- A syntax error is caused when the compiler cannot recognize a statement. They are also referred to as compile-time errors.
- Logical errors are more difficult to identify. The program with logical errors can execute successfully but it will provide incorrect result.

CONTROL FLOW STRUCTURES



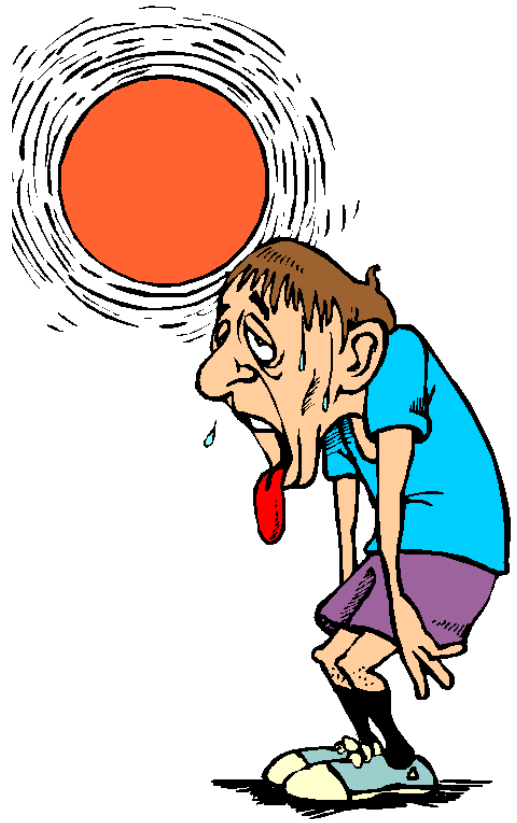
WHAT IS FLOW CONTROL?

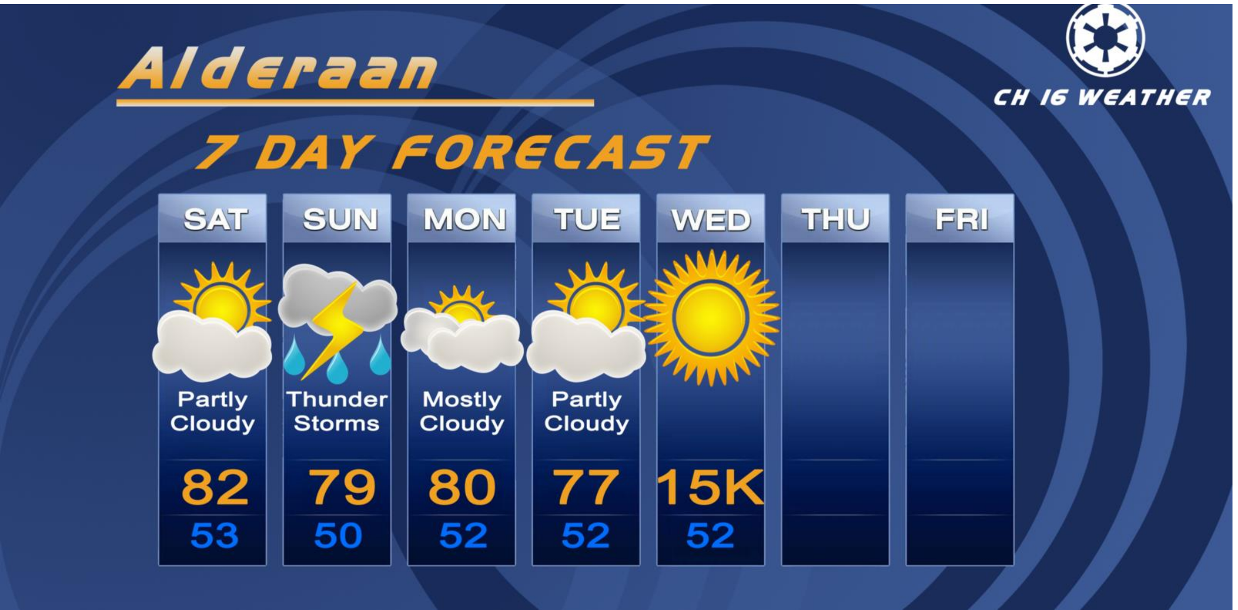
FLOW CONTROL

- Controls
  - Which parts of your code run or get skipped each time
  - What processes to execute

DECISION-MAKING LOGIC

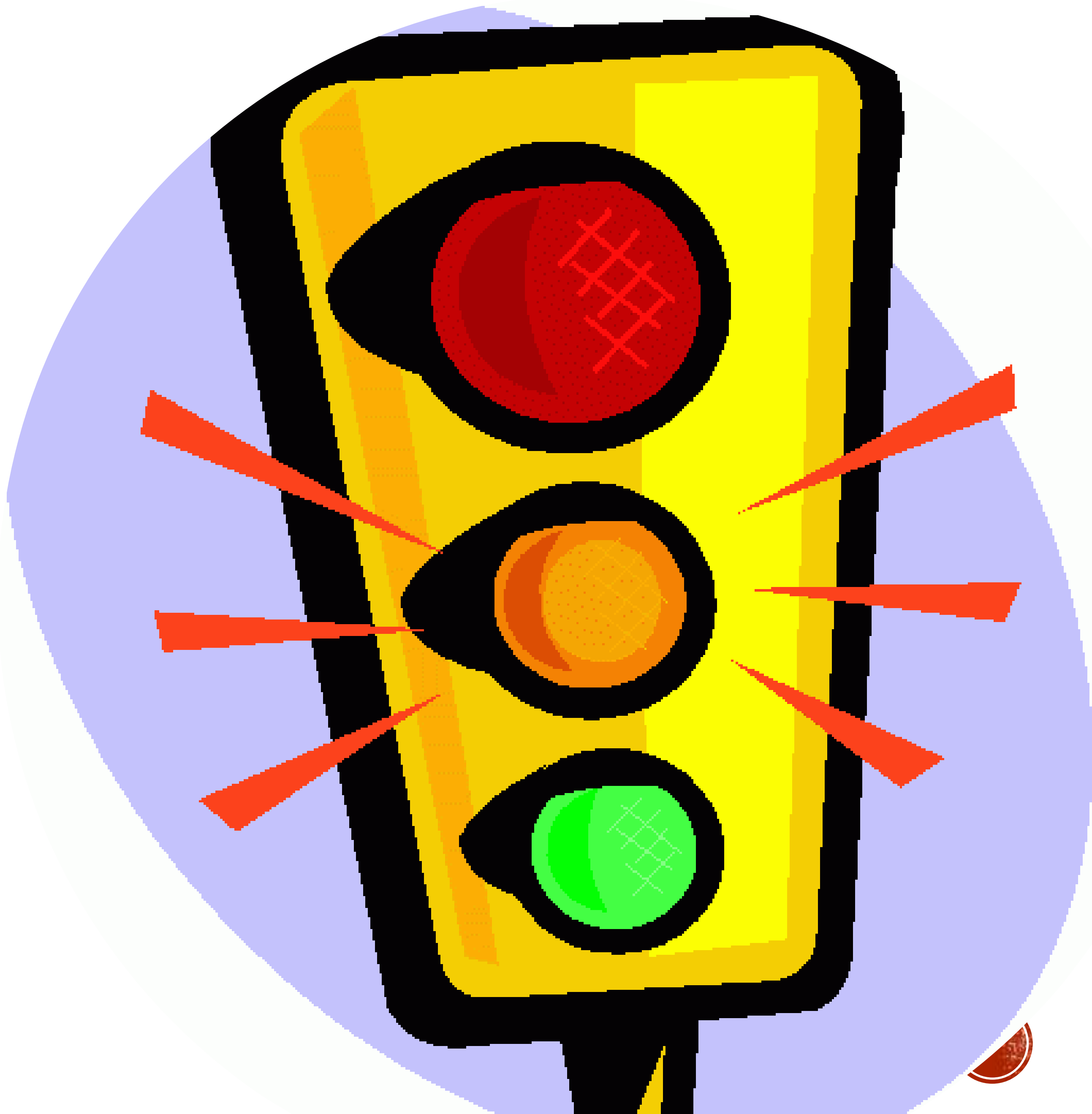
- Decision structures:
  - They are used to choose among alternative courses of action.
- All the decisions made by the computer are either yes or no.
- The decisions are indicated by Boolean values – true or false.





CONDITIONAL STATEMENTS

YOU MUST MAKE A DECISION BASED ON THE VALUE AND CIRCUMSTANCES

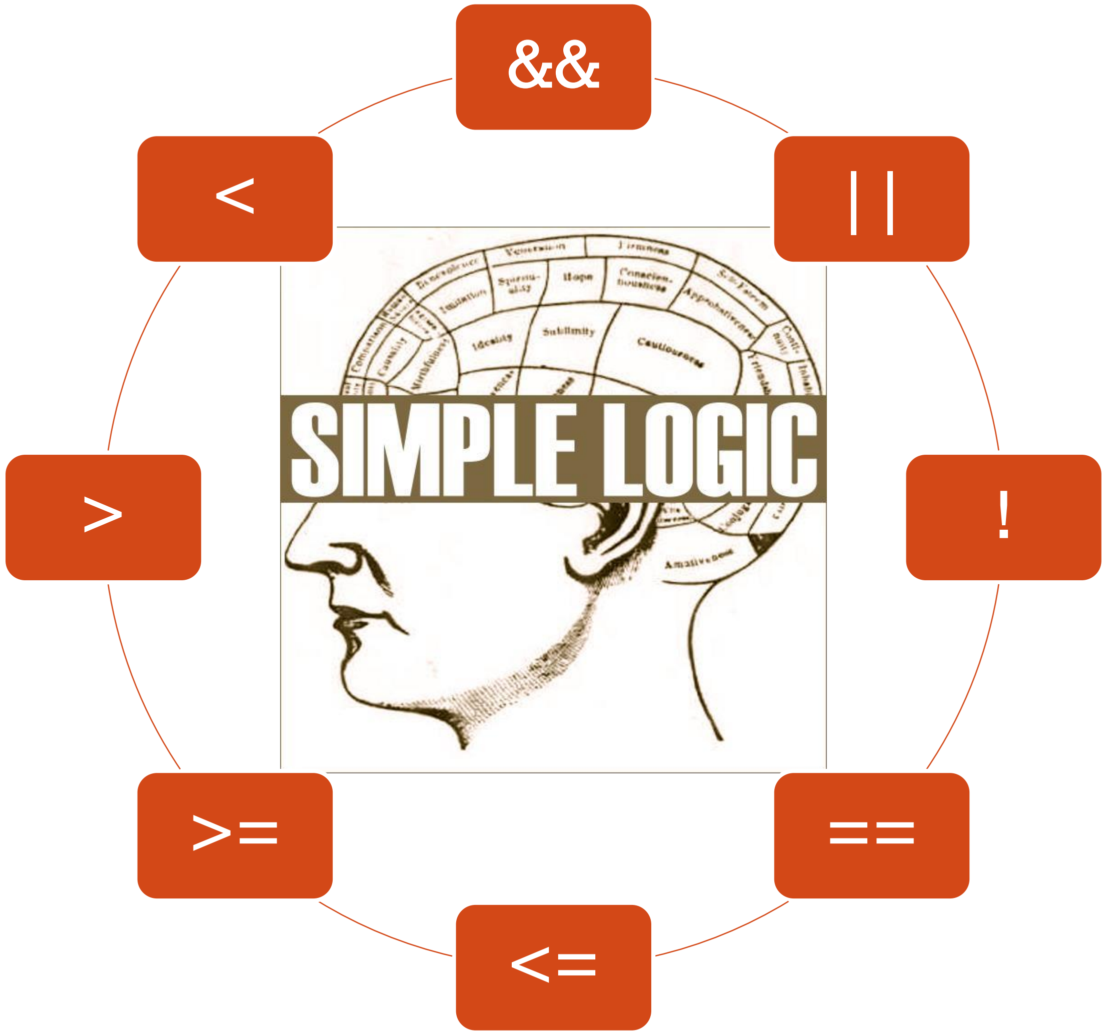


EXAMPLES

STOP
PERMISSION REQUIRED BEYOND THIS POINT

STOP
START
CONTINUE

You Must Be
18
to Enter
SmartSign.com • 800-952-1457 • S-9126-18



CONDITIONAL OPERATORS

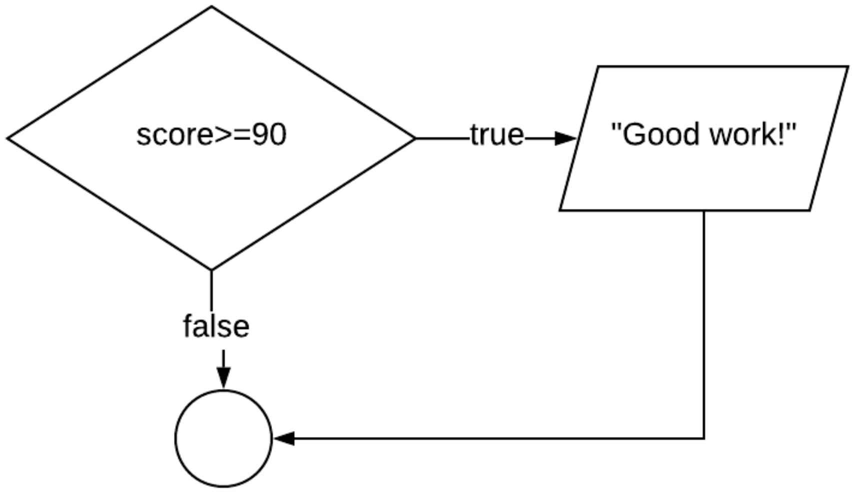
BRANCHING

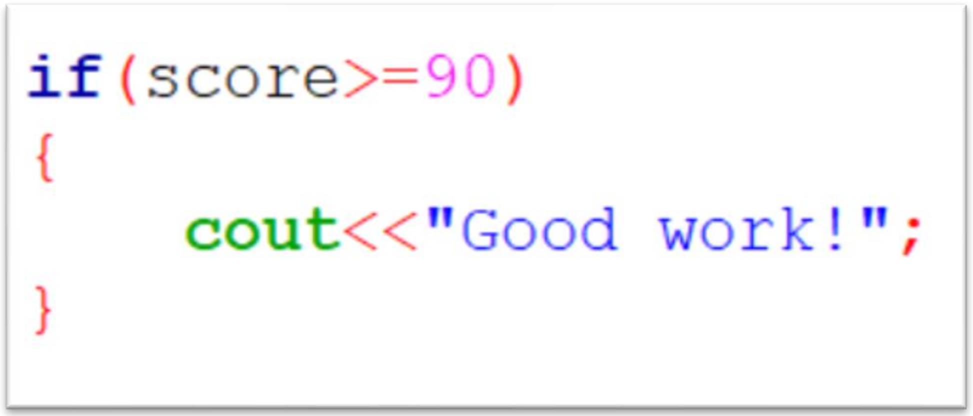
The act of controlling which line(s) of code should be executed

THE IF STATEMENT

- It is the simplest structure to make a decision.
- A Boolean expression is written in parentheses to represent the decision making logic.
- Multiple conditions can be evaluated using logical operators.
- We can create nested if structures to check interdependent values.
- The compiler ignores white-space characters like blanks, tabs, and new lines used for indentation and vertical spacing.

EXAMPLE



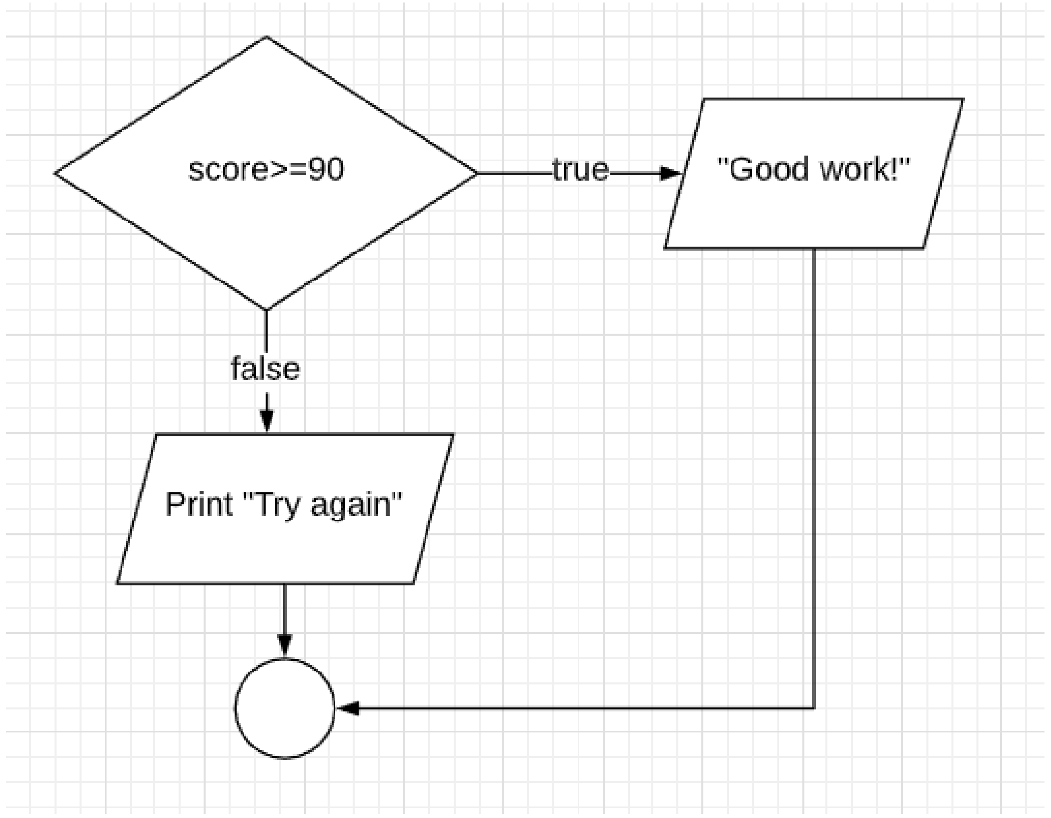


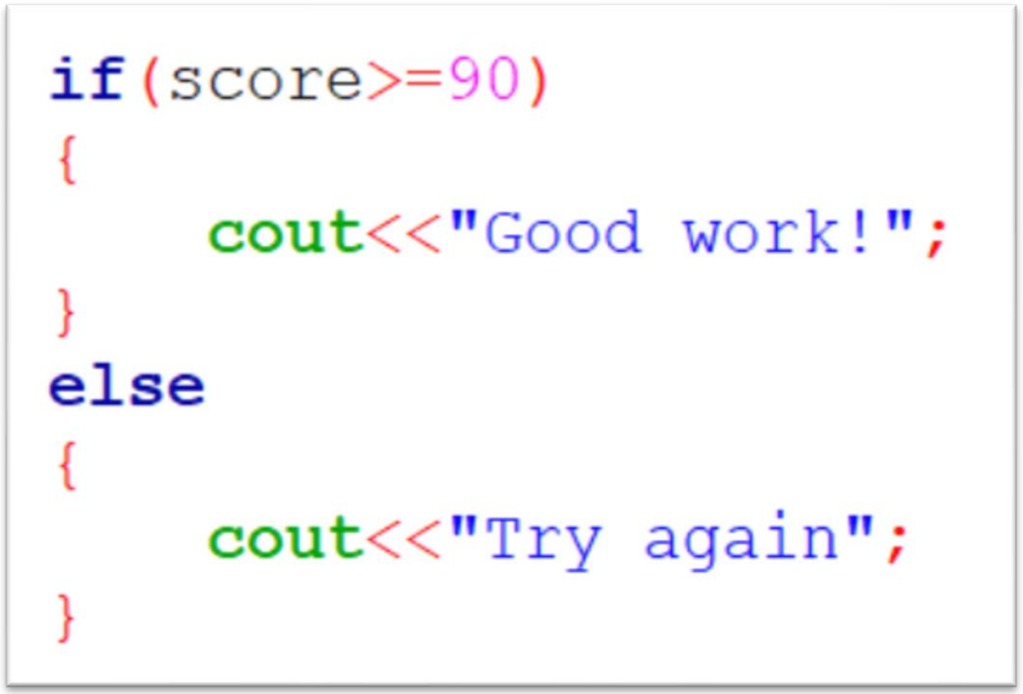
```
if(score>=90)
{
    cout<<"Good work!";
}
```

IF…ELSE… STRUCTURE

- If the condition is true, the statements following the if structure will execute, otherwise the else structure will execute.
- It is not possible to code an else without an if condition.
- Only one of the sections will execute depending on the validity of the tested condition.

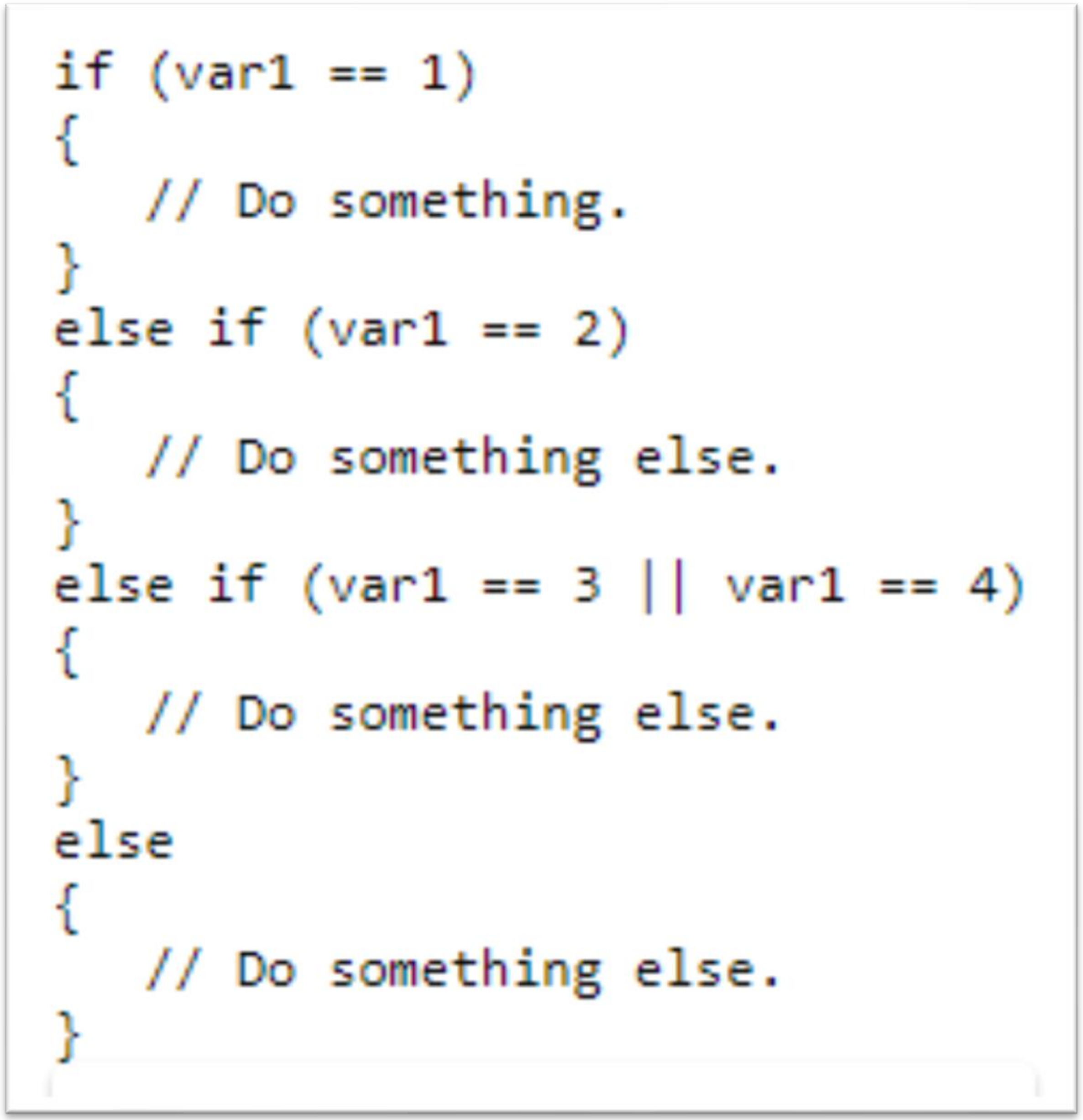
EXAMPLE





```
if(score>=90)
{
      cout<<"Good work!";
}
else
{
      cout<<"Try again";
}
```

IF…ELSE IF… TEMPLATE

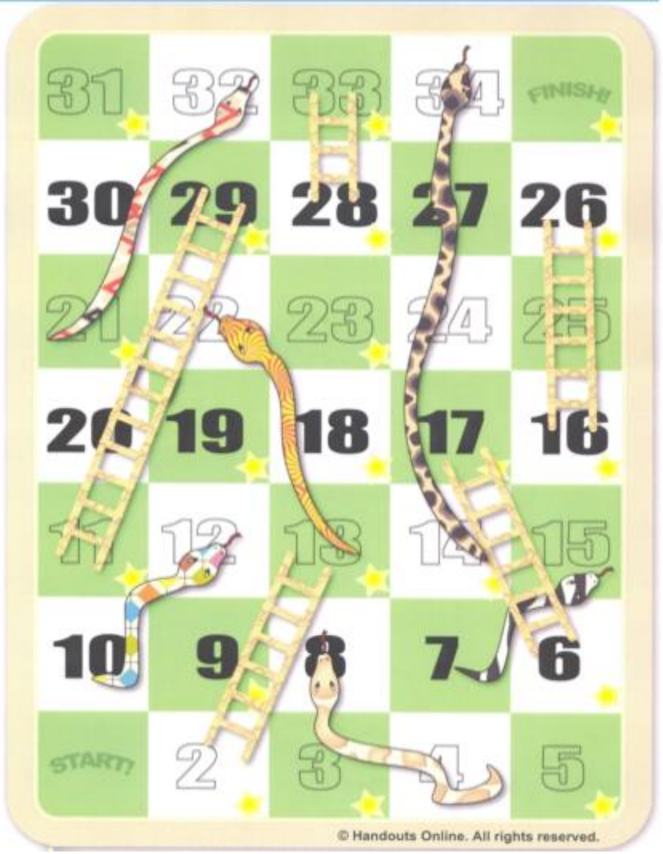


```
if (var1 == 1)
{
   // Do something.
}
else if (var1 == 2)
{
   // Do something else.
}
else if (var1 == 3 || var1 == 4)
{
   // Do something else.
}
else
{
   // Do something else.
}
```

WHAT IS A LOOP?

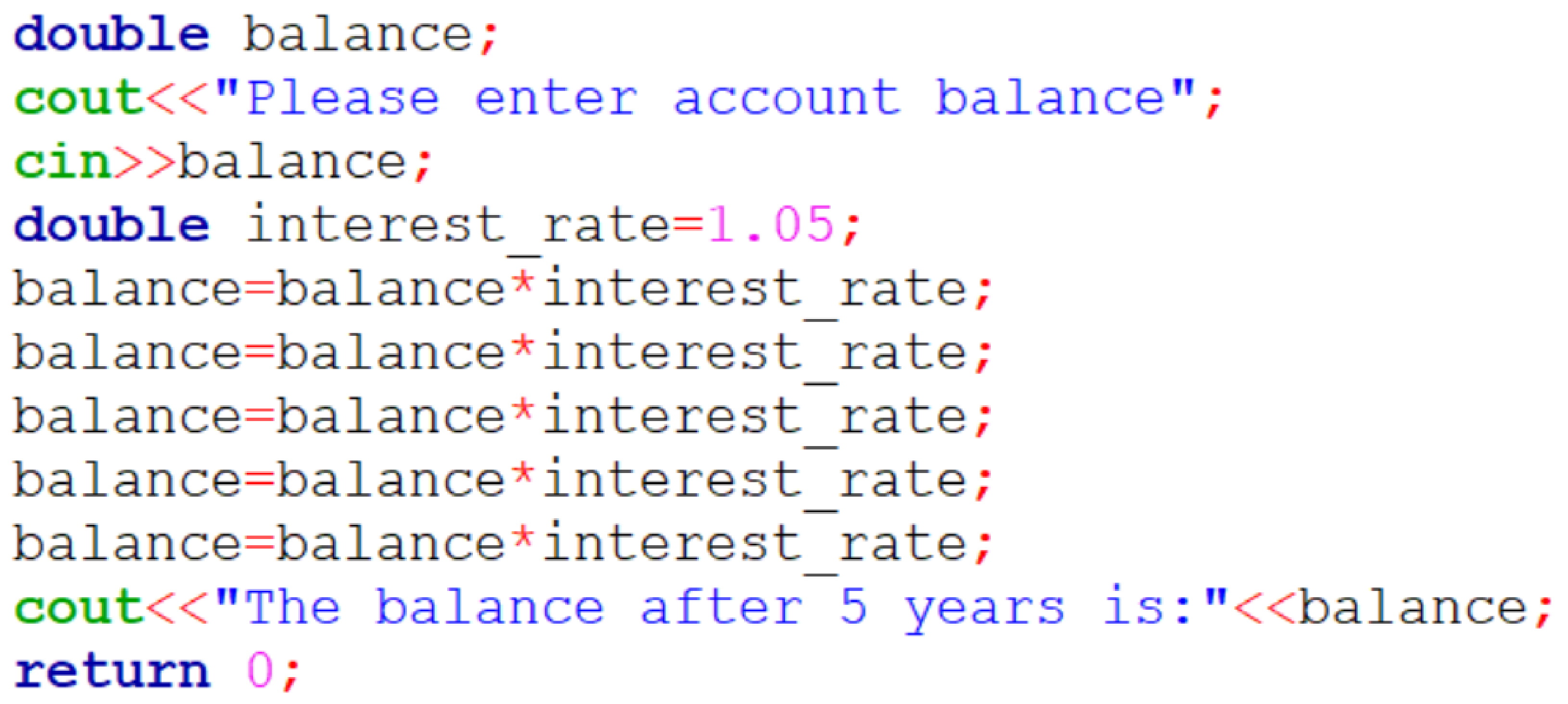
- Repeating the same block of code 0 to many times
- Cyclical
- Allows ability to iterate as many times as you want

WHY WOULD YOU EVER WANT TO LOOP?



EXAMPLE

Calculate the amount of money in a bank account after 5 years, assuming that an interest of 5% is paid each year and no other money flows into or out of the account and a starting balance of ₹1,000.



```
double balance;
cout<<"Please enter account balance";
cin>>balance;
double interest_rate=1.05;
balance=balance*interest_rate;
balance=balance*interest_rate;
balance=balance*interest_rate;
balance=balance*interest_rate;
balance=balance*interest_rate;
cout<<"The balance after 5 years is:"<<balance;
return 0;
```

This Photo by Unknown Author is licensed under CC BY

This Photo by Unknown Author is licensed under CC BY-NC

VARIATIONS

2 main variations

- while (indefinite)
- for (definite)

WHILE LOOP

- Most versatile
- Great if you are waiting for something to occur
  - User to guess random number
  - User to run out of lives in video game
  - Data to show up from a client
- Variations
  - do while loop

WHILE EXAMPLE

```
double balance;
cout<<"Please enter account balance";
cin>>balance;
double interest_rate=1.05;
int total_years=5;
int year=1;
while(year<=total_years)
{
    balance=balance*interest_rate;
    year=year+1;
}
cout<<"The balance after 5 years is:"<<balance;
return 0;
```

DO … WHILE LOOP

“BOTTOM-CONTROLLED WHILE LOOP”

CONDITION IS CHECKED AFTER THE BODY OF THE LOOP

DO WHILE EXAMPLE

```
double balance;
cout<<"Please enter account balance";
cin>>balance;
double interest_rate=1.05;
int total_years=5;
int year=1;
do
{
    balance=balance*interest_rate;
    year=year+1;
}while(year<=total_years);
cout<<"The balance after 5 years is:"<<balance;
return 0;
```

FOR EXAMPLE

```
double balance;
cout<<"Please enter account balance";
cin>>balance;
double interest_rate=1.05;
int total_years=5;
int year;
for(year=1;year<=total_years;year++)
{
    balance=balance*interest_rate;
}
cout<<"The balance after 5 years is:"<<balance;
return 0;
```

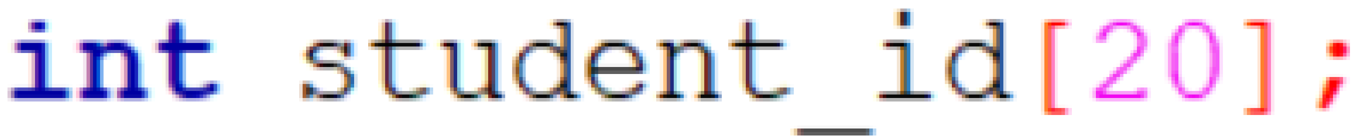
- **Great if you know the exact number of iterations you need**
  - This value can be dynamic (e.g. change at run time)

ARRAYS

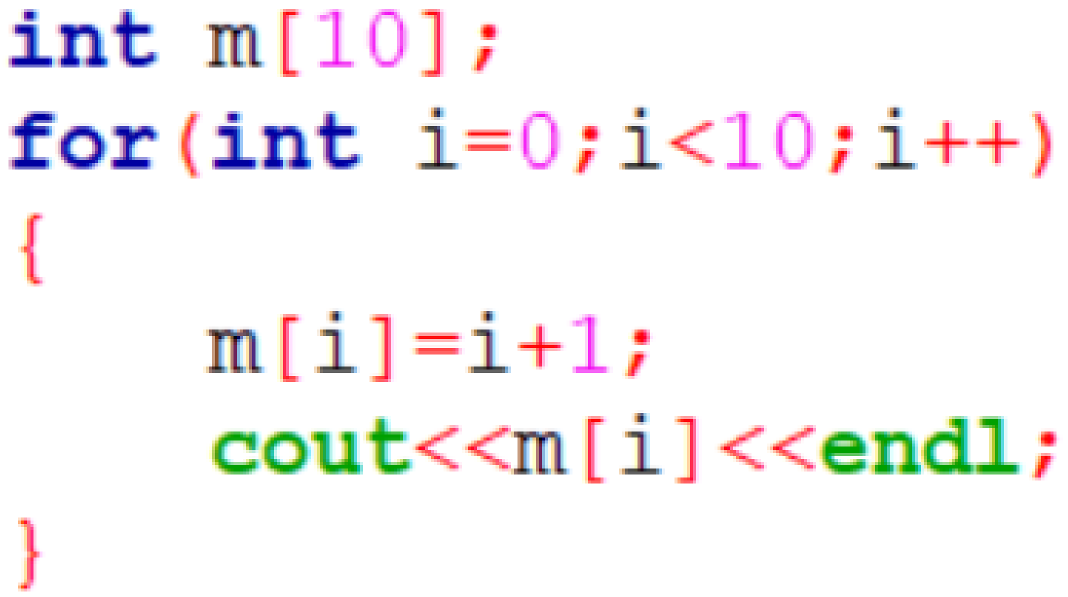
- Array is a collection of items.
- An element in the array is referred to by giving the name of the array followed by the position number of the particular element in square brackets.
- The first element in every array is the zeroth element.
- The position number contained within square brackets is also called subscript or index.

ARRAY DECLARATION

- To create an array you need to mention the data type, name of the array, and the number of items it can hold.



```
int student_id[20];
```



```
int m[10];
for(int i=0;i<10;i++)
{
    m[i]=i+1;
    cout<<m[i]<<endl;
}
```

ARRAY INITIALIZATION

- In the given code, we are using a for statement to initialize the elements of a 10-element integer array to ascending values and print the array in a tabular format.

ARRAY INITIALIZATION WITH A LIST OF VALUES

- We can also initialize an array using a list of values.

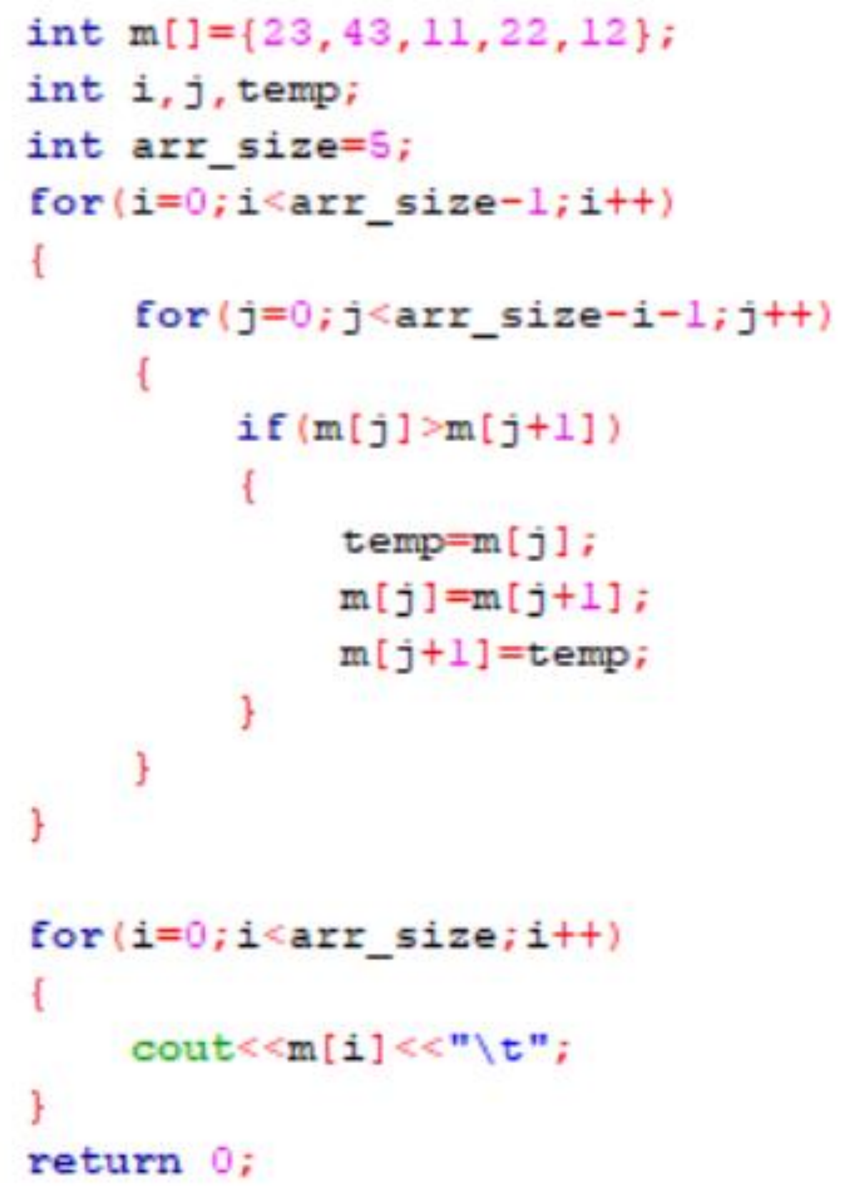
```
int n[5]={23,43,22,12,11};
```

- If we want to initialize all the values in an array to a single value, we can do the following:

```
int n[5]={0};
```

SORTING ARRAYS

- Sorting is an important technique to arrange data in an organized manner.
- There are different techniques of sorting an array.
- One of the techniques we will study is bubble sort.
- In bubble sort, the smaller values gradually “bubble” their way upward to the top of the array like air bubble rising in water, while the larger values sink to the bottom of the array.



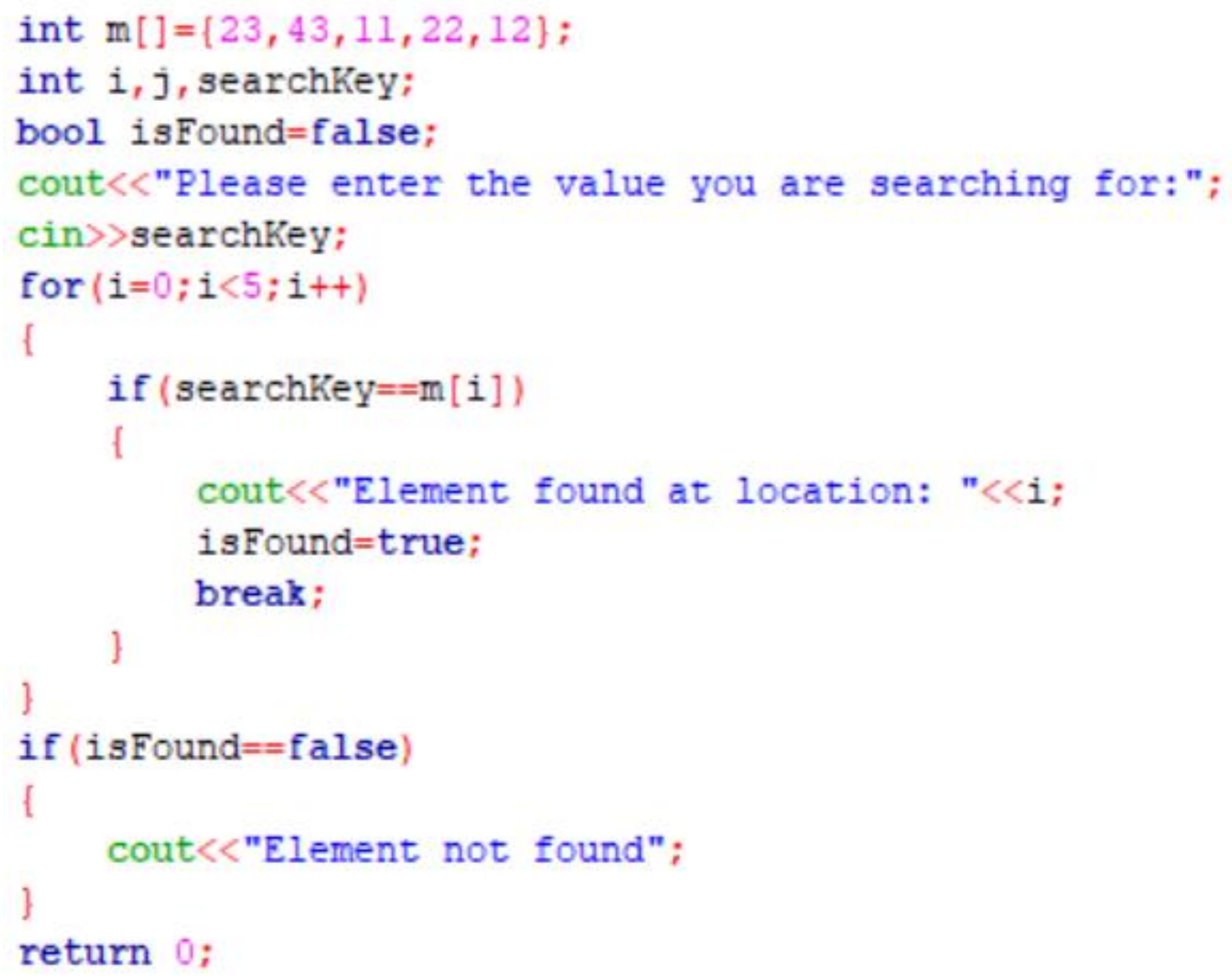
```
int m[]={23,43,11,22,12};
int i,j,temp;
int arr_size=5;
for(i=0;i<arr_size-1;i++)
{
    for(j=0;j<arr_size-i-1;j++)
    {
        if(m[j]>m[j+1])
        {
            temp=m[j];
            m[j]=m[j+1];
            m[j+1]=temp;
        }
    }
}

for(i=0;i<arr_size;i++)
{
    cout<<m[i]<<"\t";
}
return 0;
```

SEARCHING ARRAYS

- Sometimes, you will work with huge amounts of data.
- You may need to search for a particular value in a dataset.
- We can use searching techniques to lookup a value in an array.
- There are two searching techniques:
  - Linear search
  - Binary search

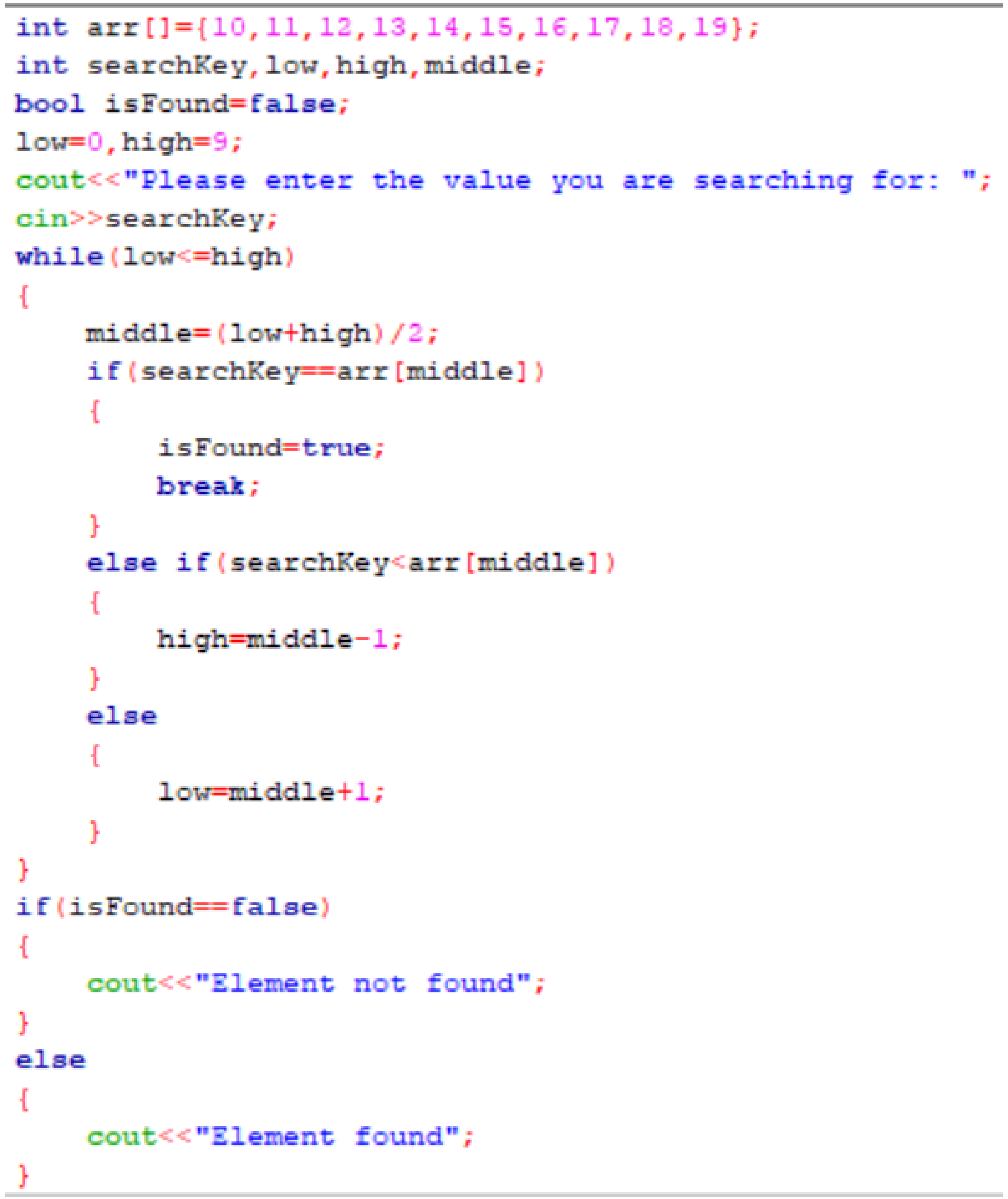
LINEAR SEARCH



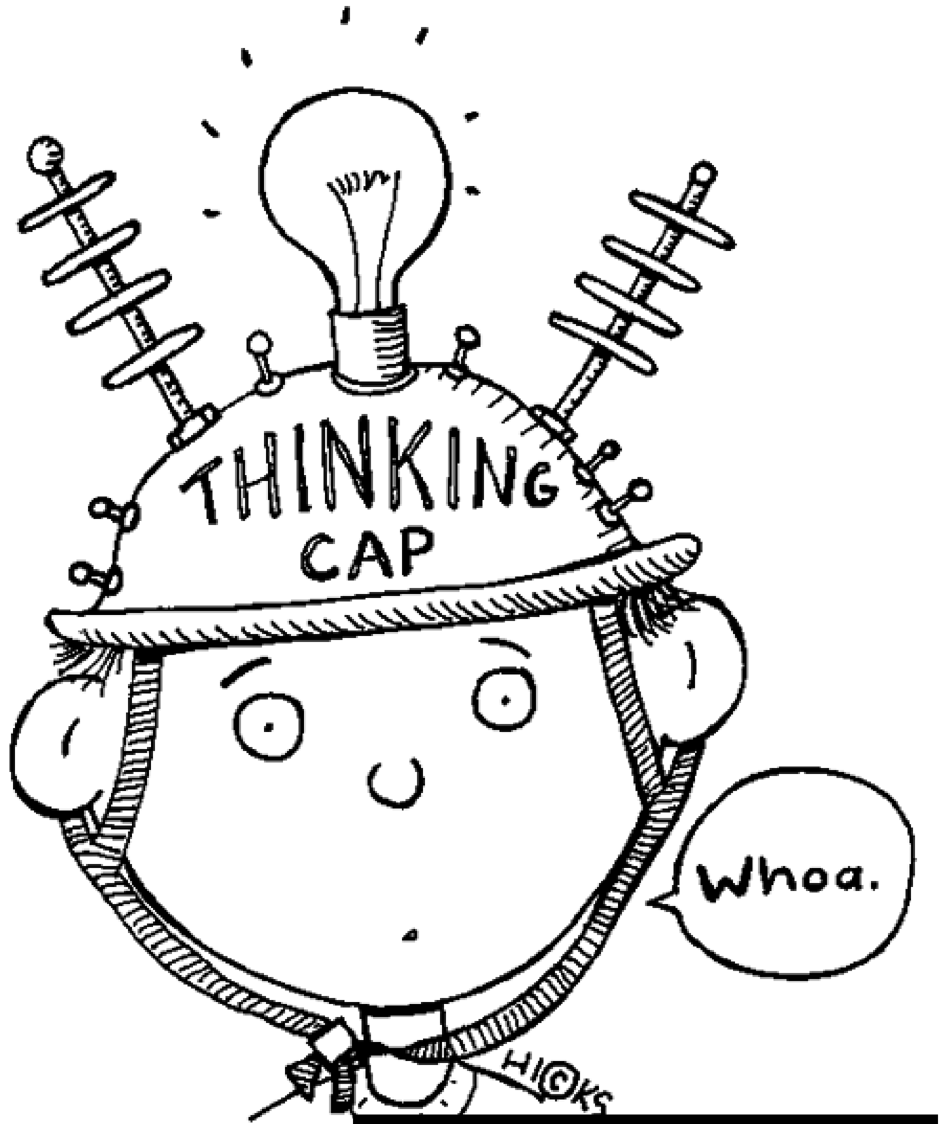
```
int m[]={23,43,11,22,12};
int i,j,searchKey;
bool isFound=false;
cout<<"Please enter the value you are searching for:";
cin>>searchKey;
for(i=0;i<5;i++)
{
    if(searchKey==m[i])
    {
        cout<<"Element found at location: "<<i;
        isFound=true;
        break;
    }
}
if(isFound==false)
{
    cout<<"Element not found";
}
return 0;
```

BINARY SEARCH

- For small and unsorted arrays, linear search is a useful approach.
- But for large arrays, linear search is inefficient.
- If the array is sorted, we can use binary search.
- Binary search algorithm eliminates one-half on the elements in an array from consideration after each comparison.



```cpp
int arr[]={10,11,12,13,14,15,16,17,18,19};
int searchKey,low,high,middle;
bool isFound=false;
low=0,high=9;
cout<<"Please enter the value you are searching for: ";
cin>>searchKey;
while(low<=high)
{
    middle=(low+high)/2;
    if(searchKey==arr[middle])
    {
        isFound=true;
        break;
    }
    else if(searchKey<arr[middle])
    {
        high=middle-1;
    }
    else
    {
        low=middle+1;
    }
}
if(isFound==false)
{
    cout<<"Element not found";
}
else
{
    cout<<"Element found";
}
```



QUESTIONS?

THANK YOU