



Al-Albayt University
Computer Science Department

C++ Programming 1 (901133)

Instructor: Eng. Rami Jaradat

rjaradat@aabu.edu.jo

Subjects

1. [Introduction to C++ Programming](#)
2. [Control Structures](#)
3. [Functions](#)
4. [Arrays](#)
5. [Pointers](#)
6. [Strings](#)

1 - Introduction to C++ Programming

What is computer?

- Computers are programmable devices capable of performing computations and making logical decisions.
- Computers can store, retrieve, and process data according to a list of instructions
- Hardware is the physical part of the compute: keyboard, screen, mouse, disks, memory, and processing units
- Software is a collection of computer programs, procedures and documentation that perform some tasks on a computer system

Computer Logical Units

- Input unit
 - obtains information (data) from input devices
- Output unit
 - outputs information to output device or to control other devices.
- Memory unit
 - Rapid access, low capacity, stores information
- Secondary storage unit
 - cheap, long-term, high-capacity storage, stores inactive programs
- Arithmetic and logic unit (ALU)
 - performs arithmetic calculations and logic decisions
- Central processing unit (CPU):
 - supervises and coordinates the other sections of the computer

Computer language

- Machine languages: machine dependent, it consists of strings of numbers giving machine specific instructions:

+1300042774

+1400593419

+1200274027

- Assembly languages: English-like abbreviations representing elementary operations, assemblers convert assembly language to machine language:

load basepay

add overpay

store grosspay

- High-level languages: Similar to everyday English, use mathematical notations, compilers convert high-level language into machine language, C++ is a high level language:

grossPay = basePay + overTimePay

Program Design

- Programming is a creative process
- Program Design Process
 - Problem Solving Phase
 - Result is an algorithm that solves the problem
 - Implementation Phase
 - Result is the algorithm translated into a programming language

Problem Solving Phase

- Be certain the task is completely specified
 - What is the input?
 - What information is in the output?
 - How is the output organized?
- Develop the algorithm before implementation
 - Experience shows this saves time in getting program to run.
 - Test the algorithm for correctness

Problem Solving Phase

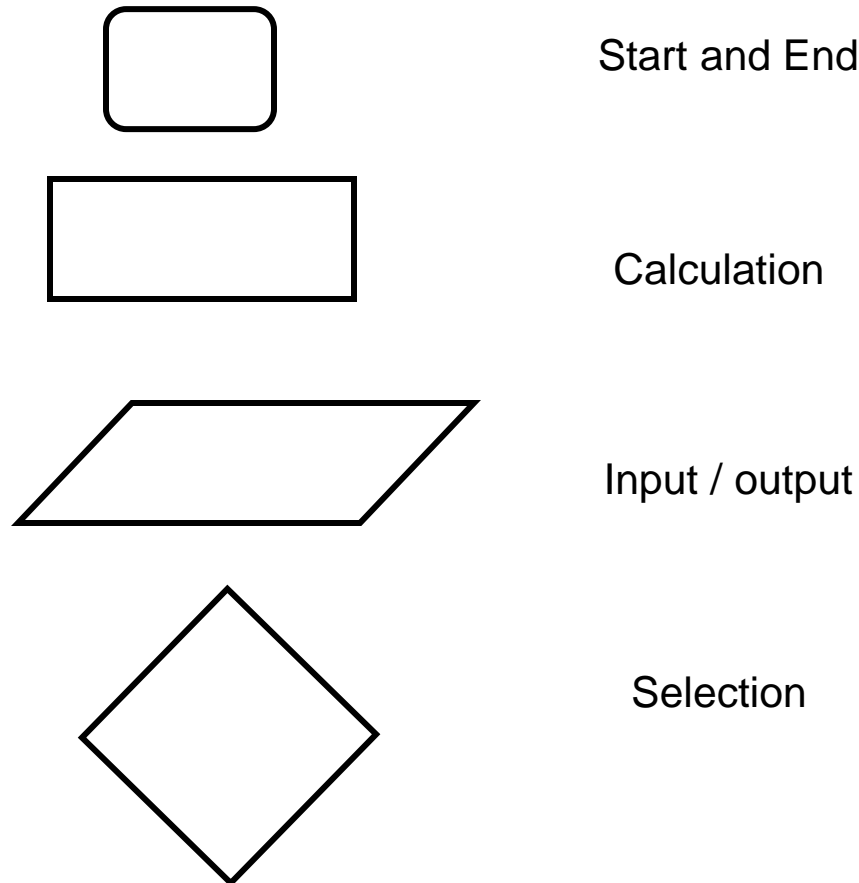
- Algorithm
 - A sequence of precise instructions (written is **pseudo** code or represented as a **flowchart**) which leads to a solution
- Pseudo code
 - Artificial, informal language similar to everyday English
 - Used to develop algorithms and not executed on computers
 - Only executable statements, no need to declare variables
- Program
 - An algorithm expressed in a language the computer can understand

Implementation Phase

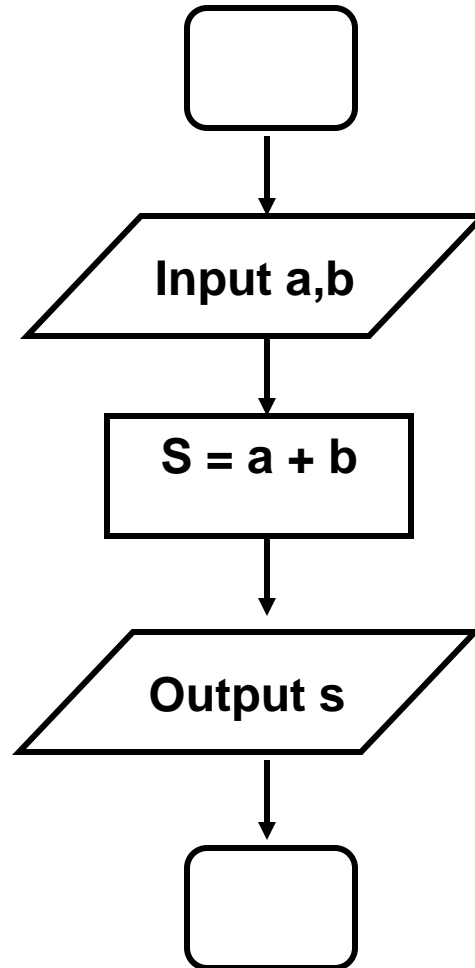
- Translate the algorithm into a programming language
 - Easier as you gain experience with the language
- Compile the source code
 - Locates errors in using the programming language
- Run the program on sample data
 - Verify correctness of results
- Results may require modification of the algorithm and program

Flowchart

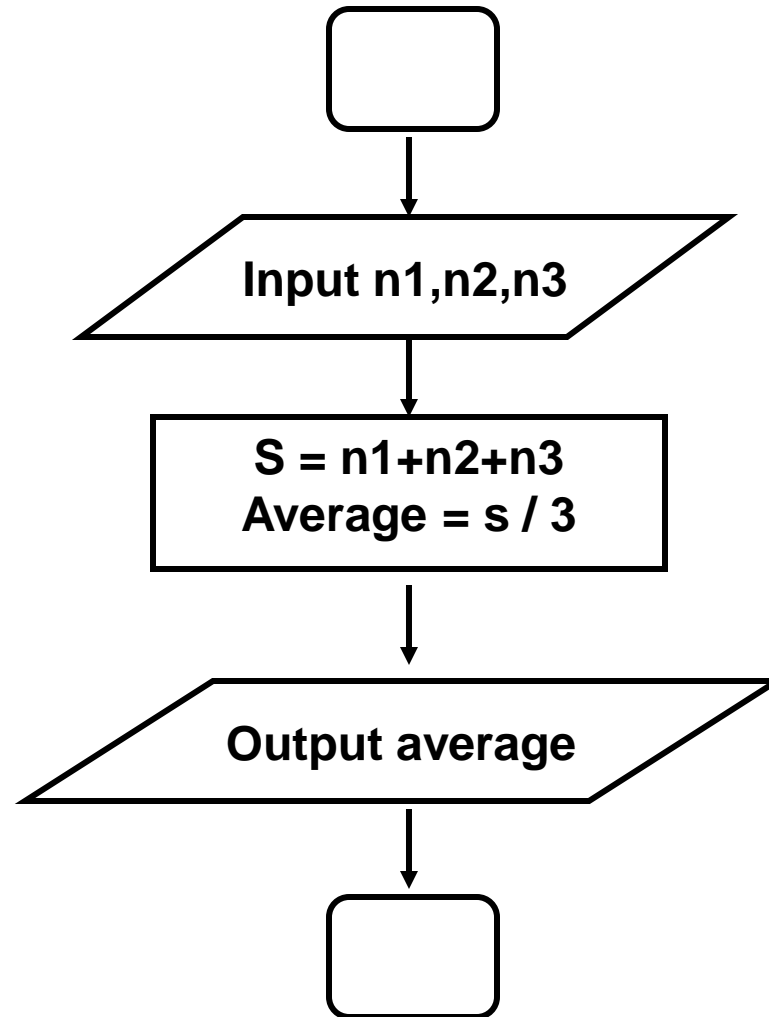
- Graphical representation of an algorithm or a portion of algorithm
- Drawn using certain special-purpose symbols connected by arrows called flow lines:



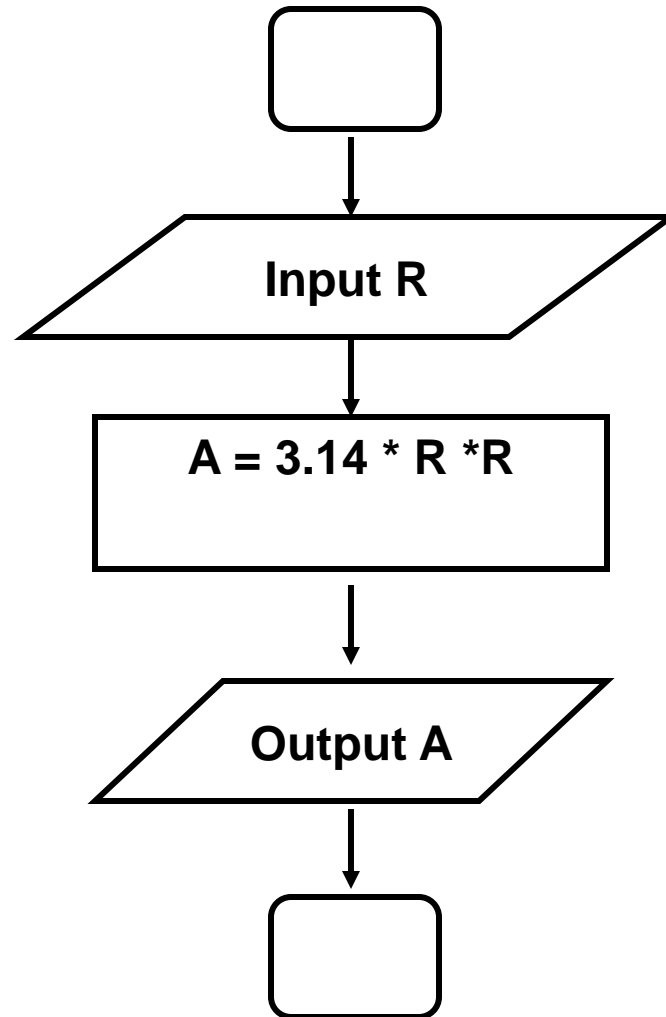
Compute and print the summation of two numbers



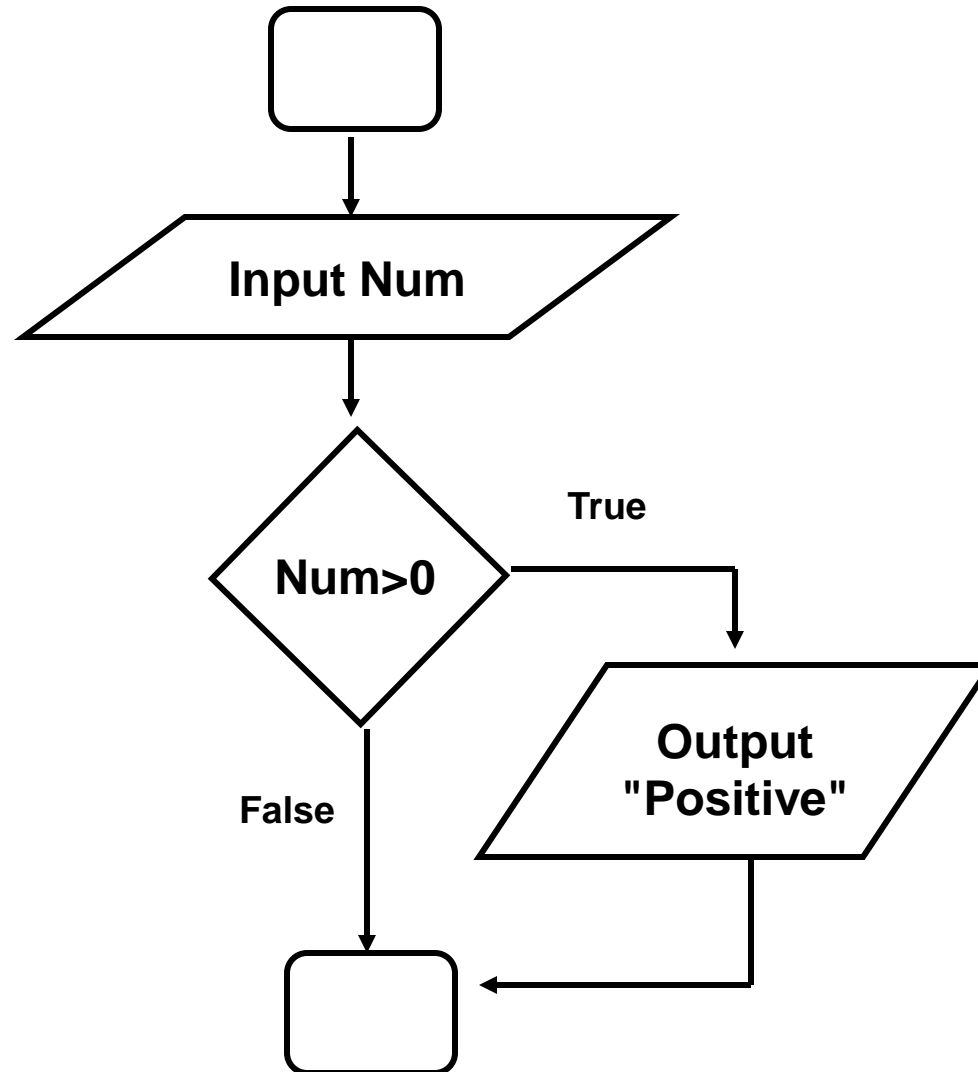
Compute and print the average of three numbers



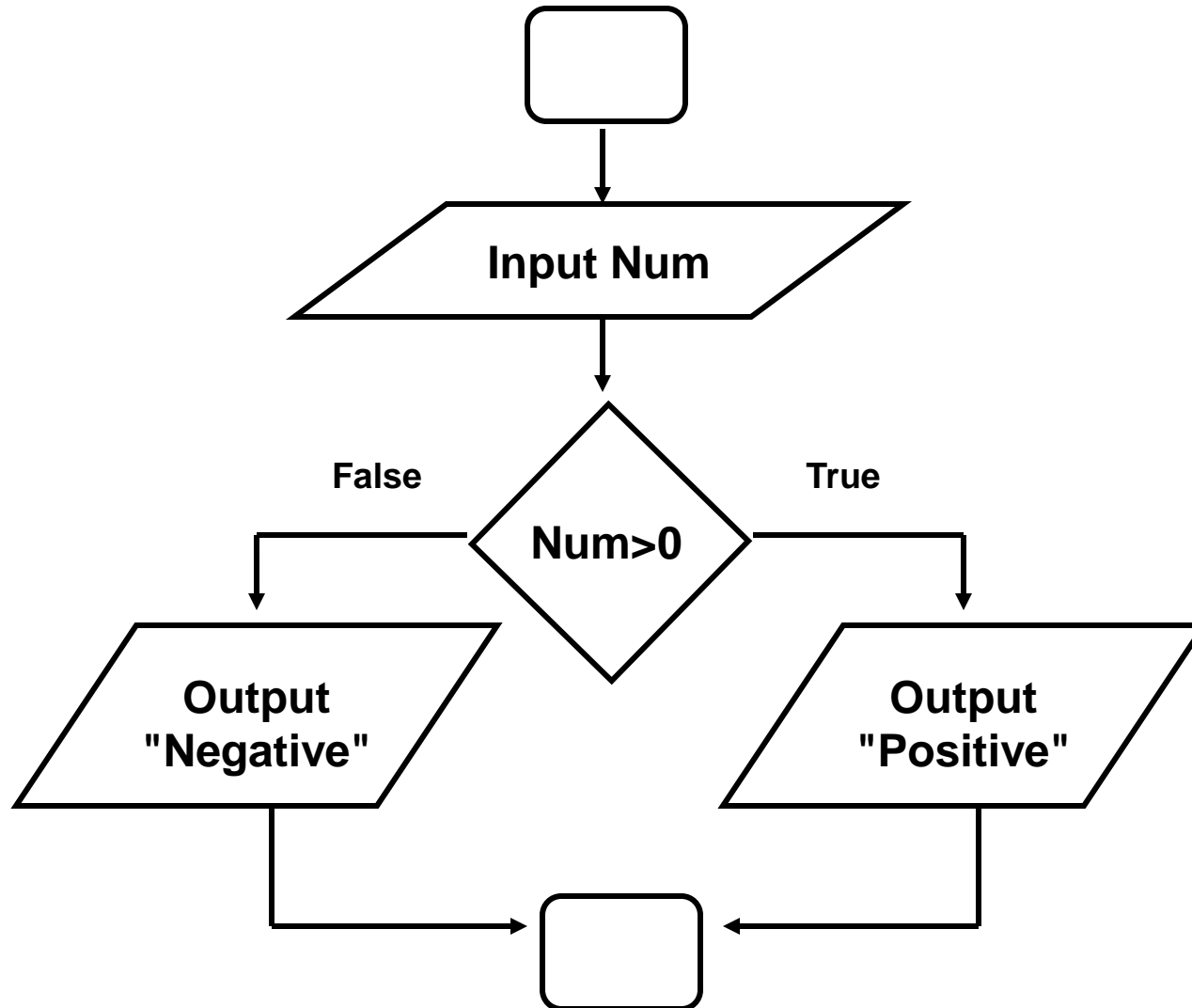
Compute the area of the circle
Where $\text{area} = 3.14 \times R^2$



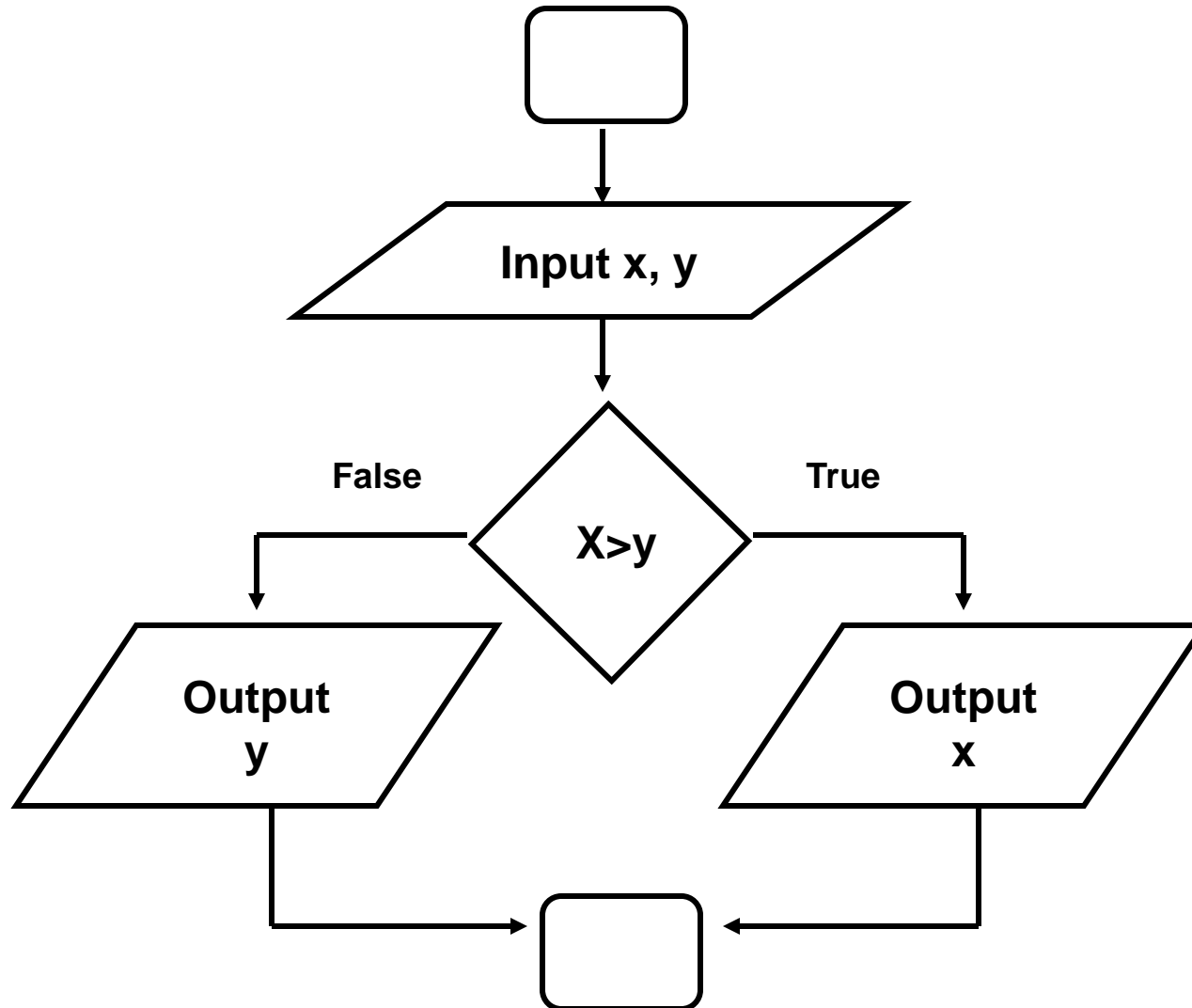
Read a number then print positive if it is positive



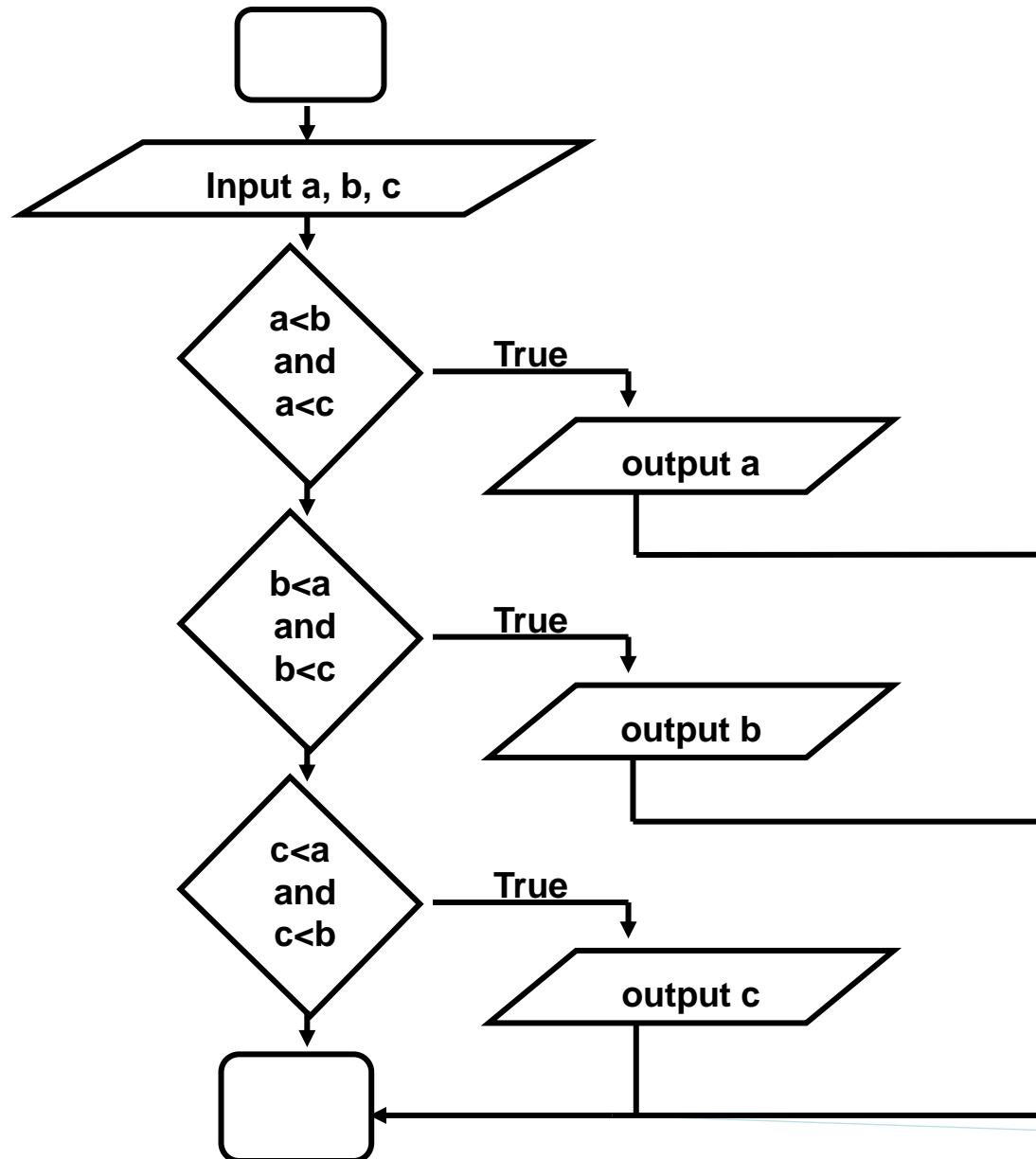
Read a number then print positive if it is positive and print negative otherwise.



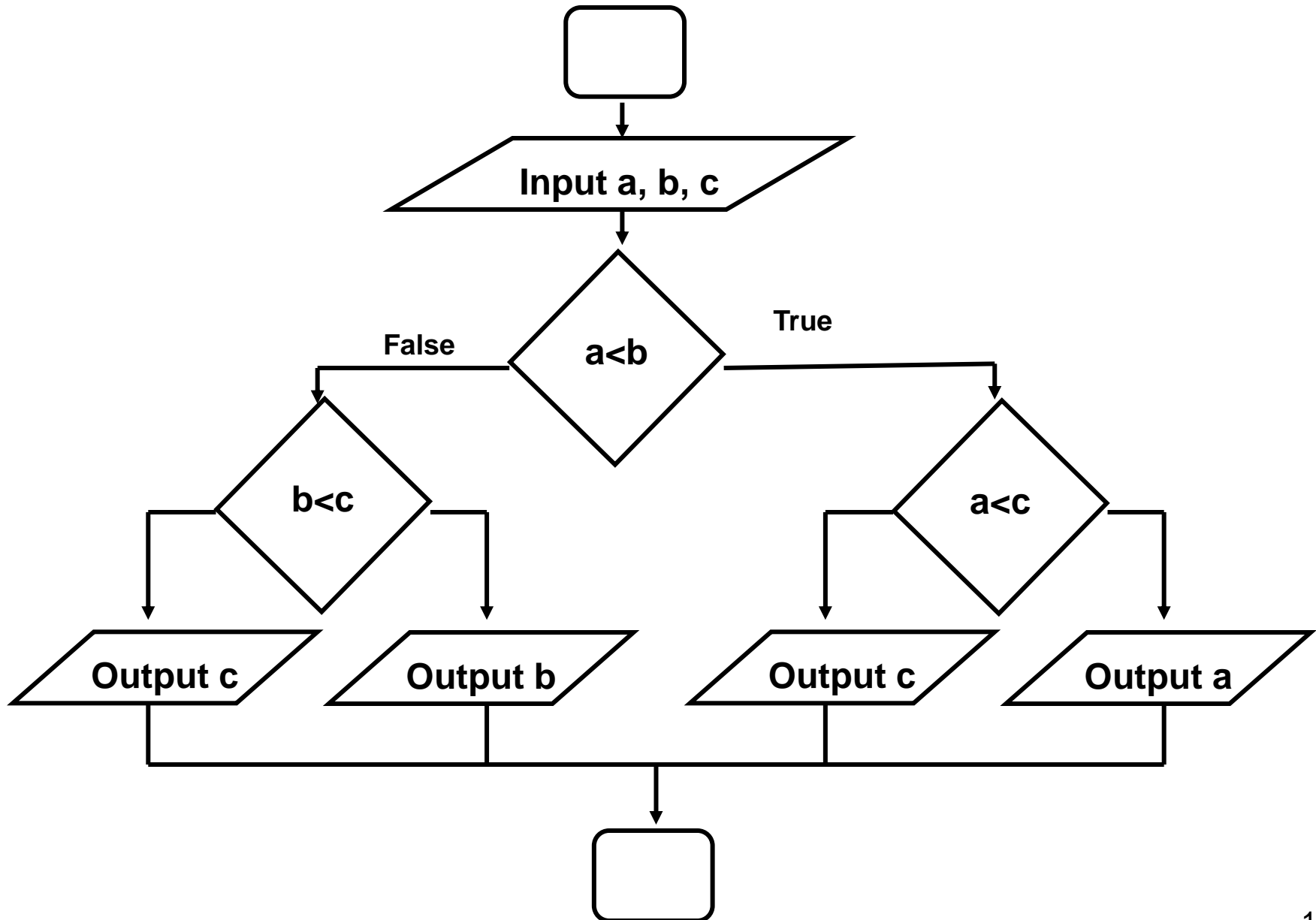
Read Two numbers then print the greatest one



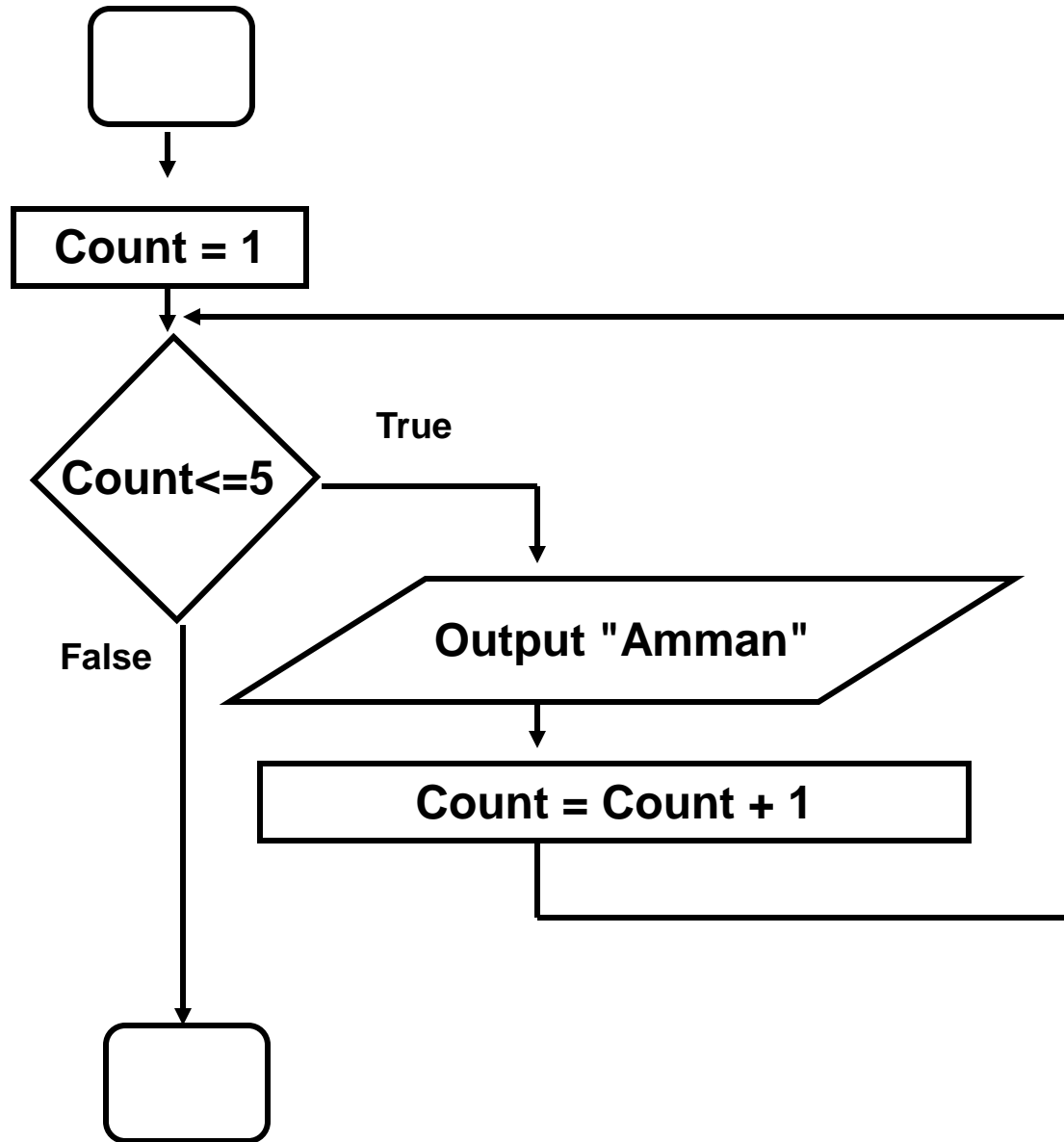
Read three numbers and print the smallest one



Another Solution

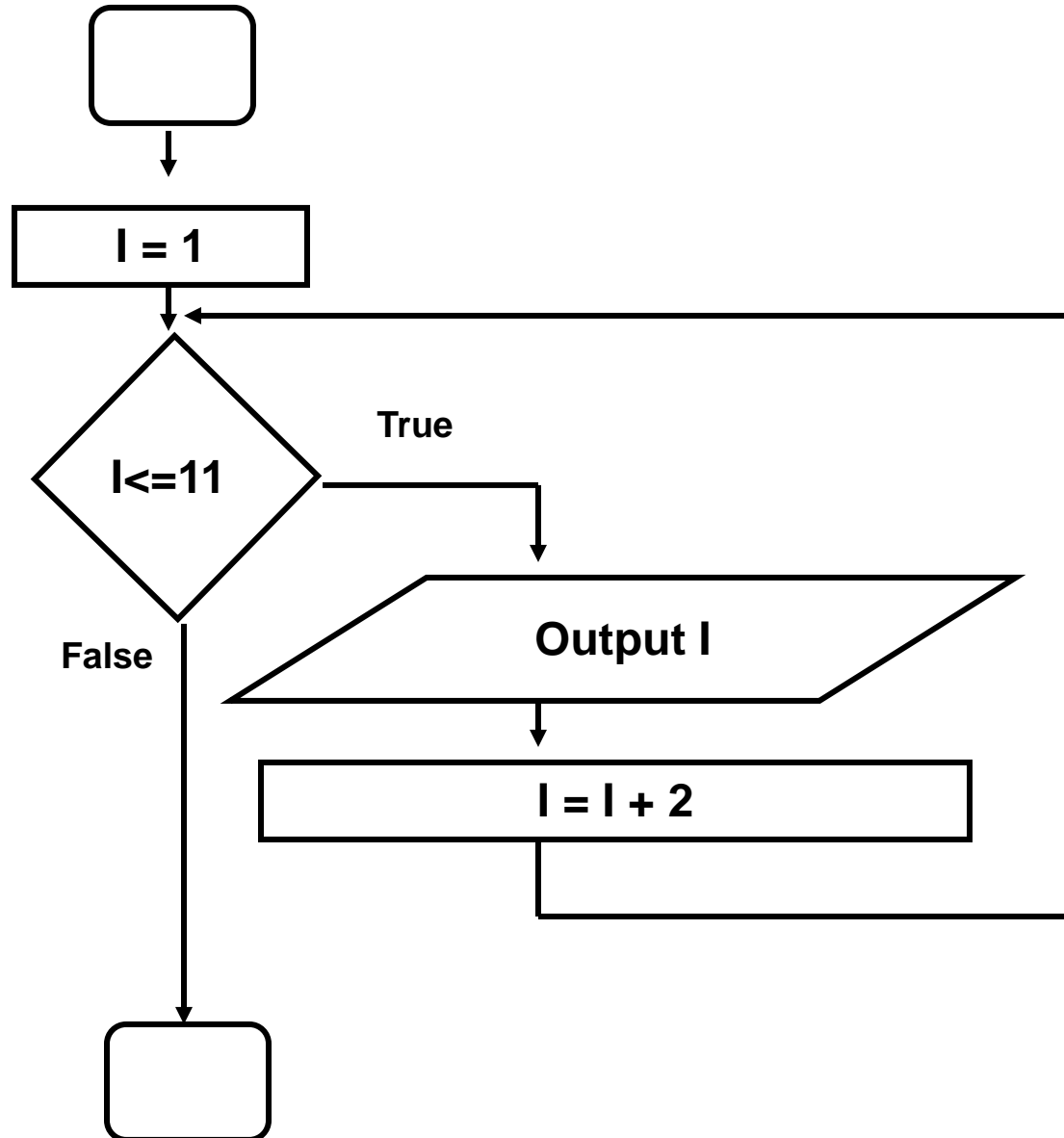


Print the word "Amman" five times.



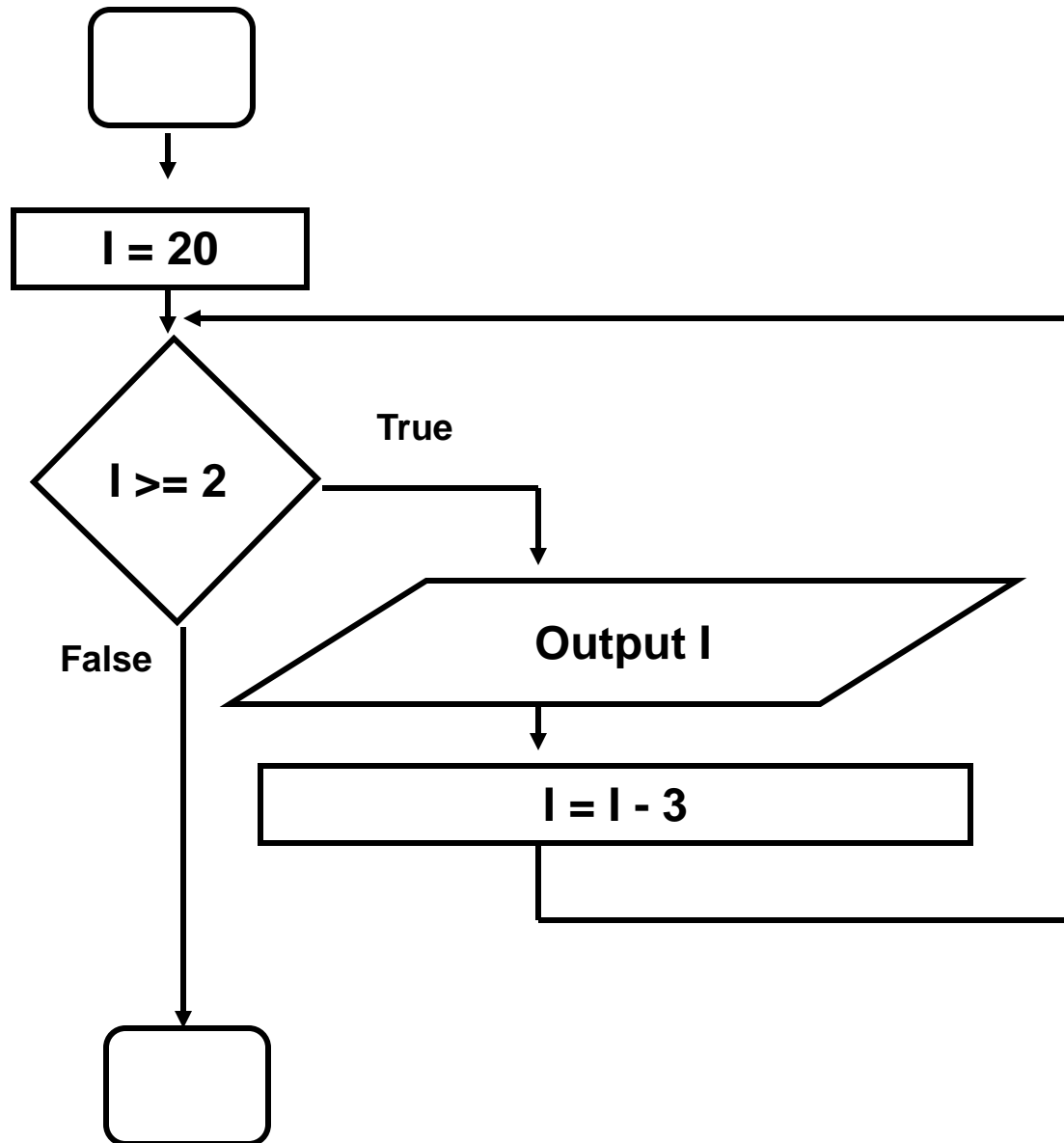
Print the following numbers

1 3 5 7 9 11

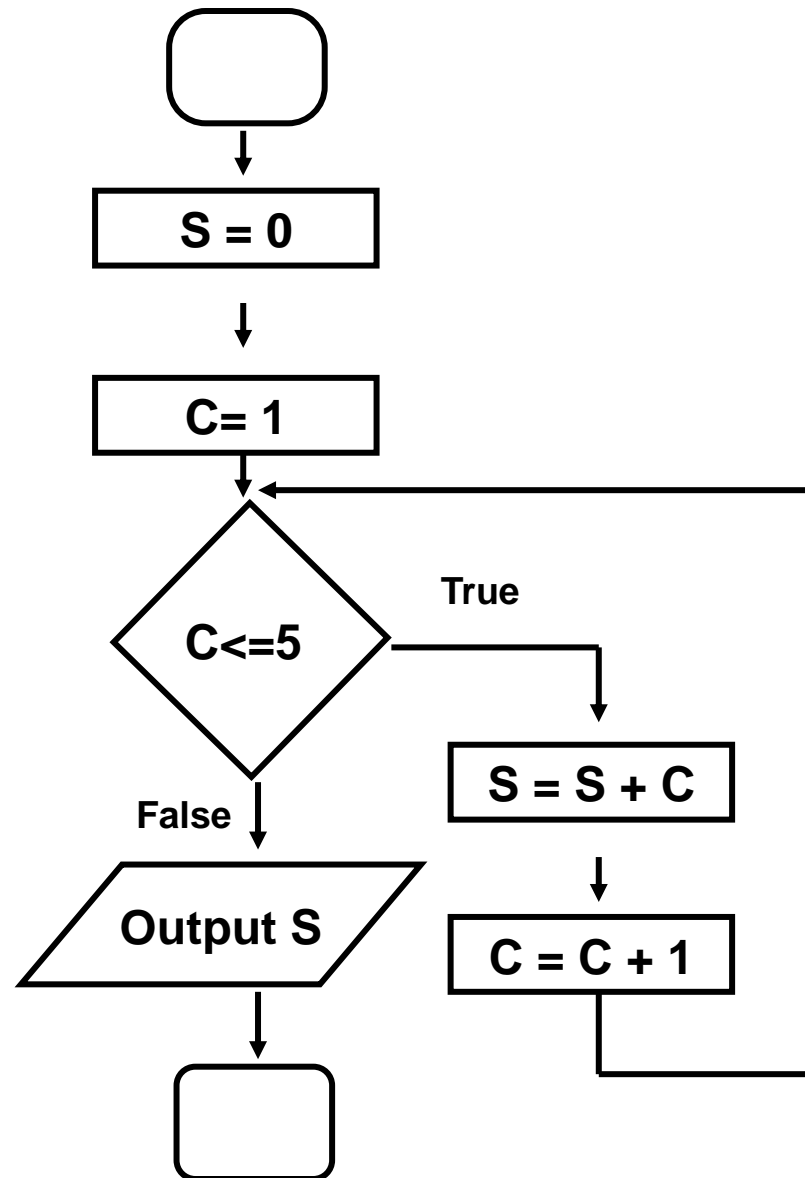


Print the following numbers

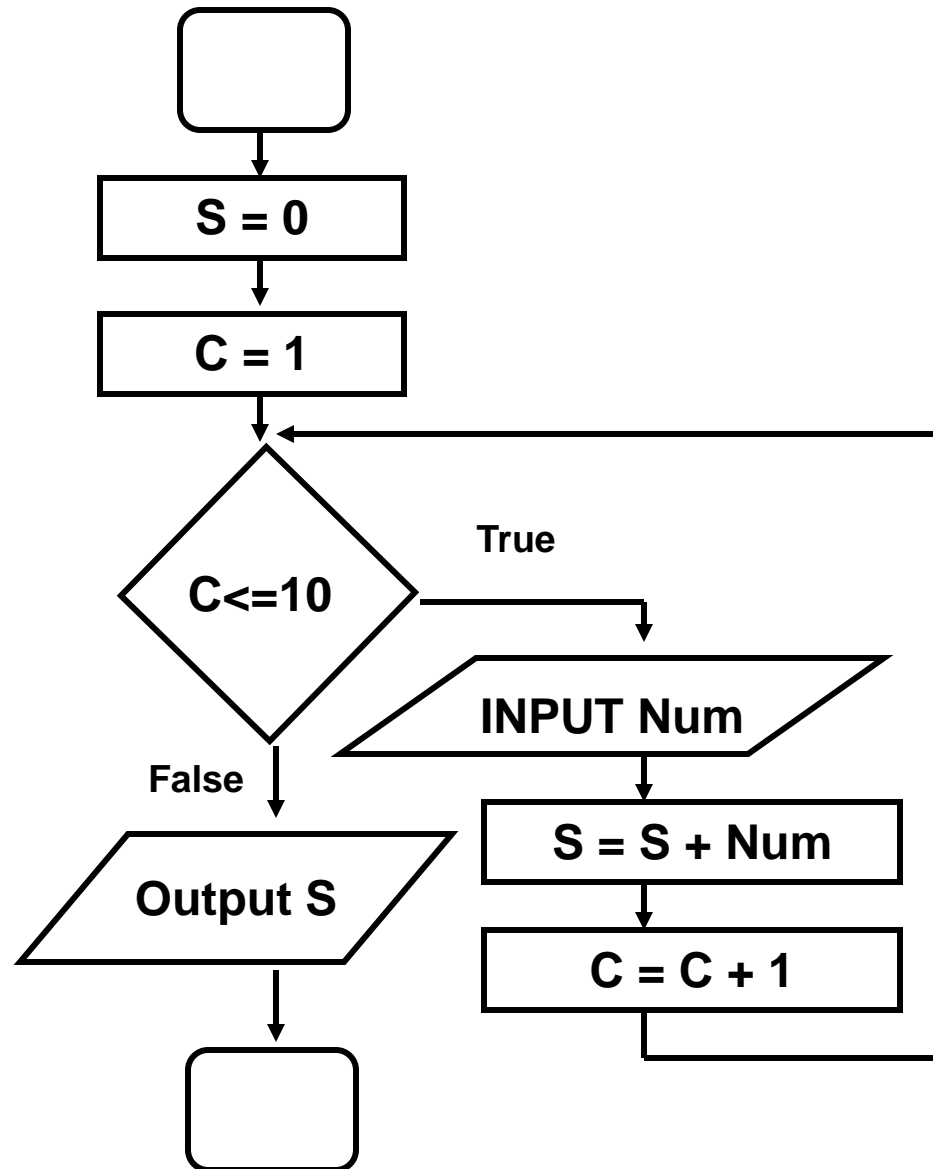
20 17 14 11 8 5 2



Compute and print S
Where $S = 1 + 2 + 3 + 4 + 5$

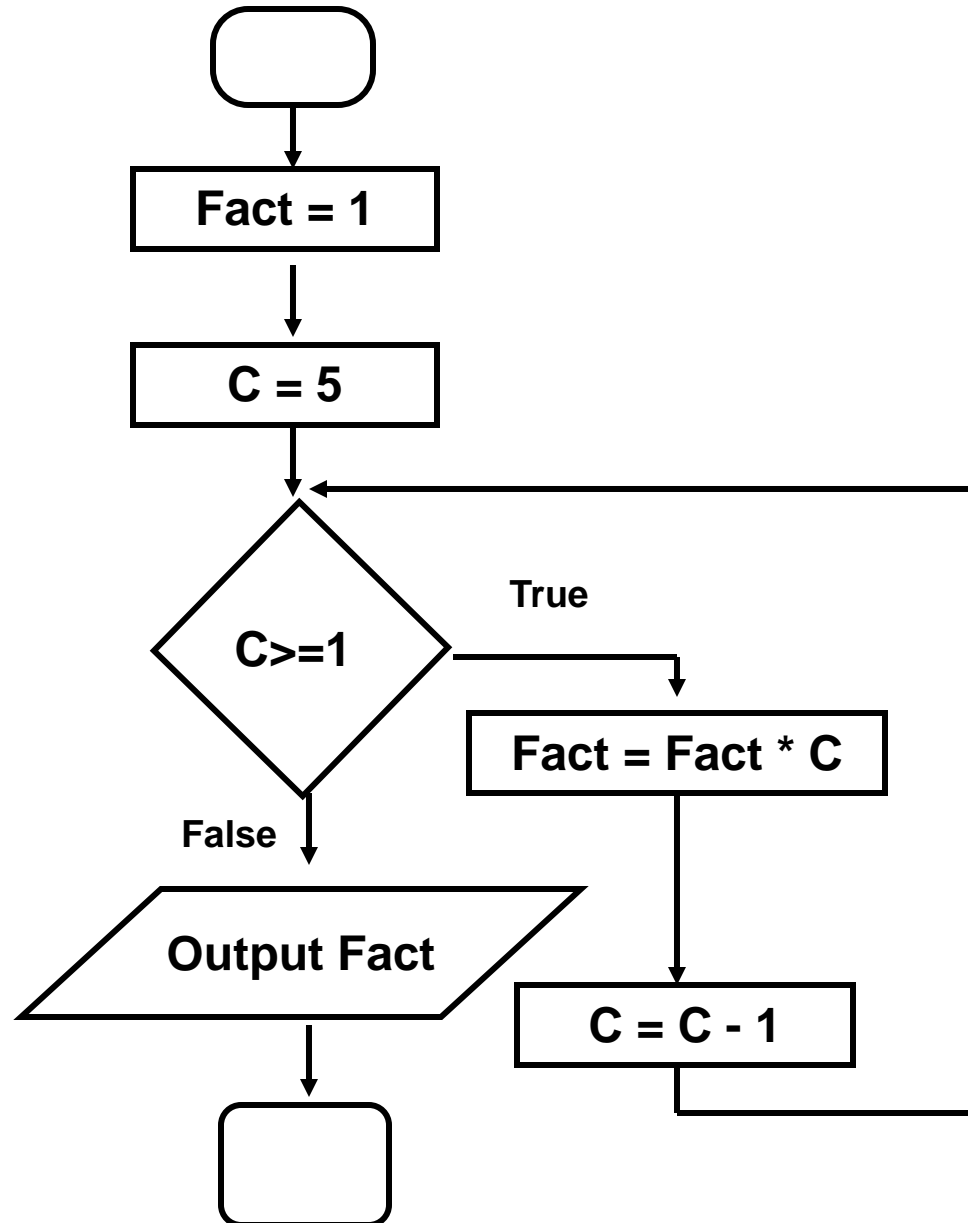


Print the Sum of 10 numbers entered by the user



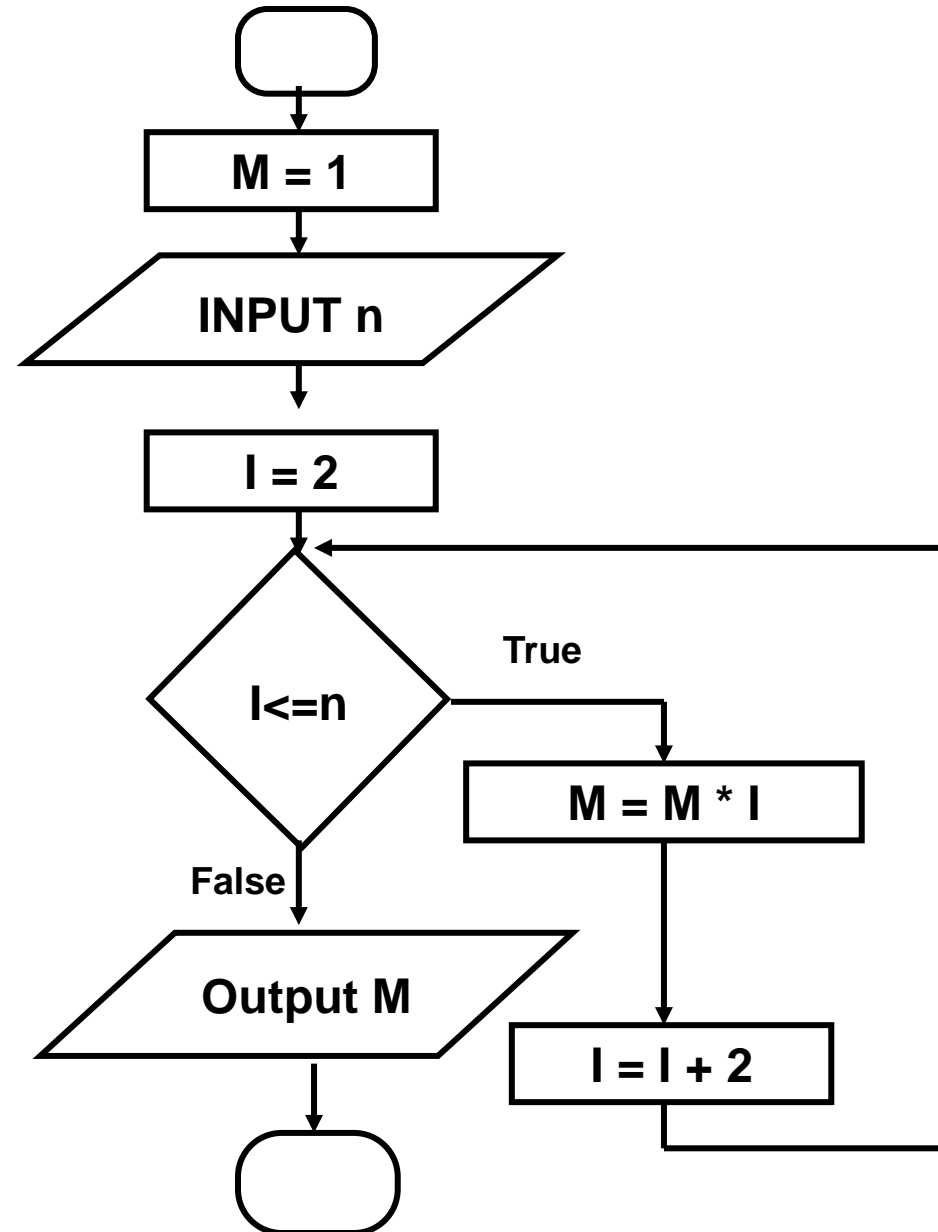
Compute and Print the factorial of 5, where:

$$fact(5) = 5 \times 4 \times 3 \times 2 \times 1$$



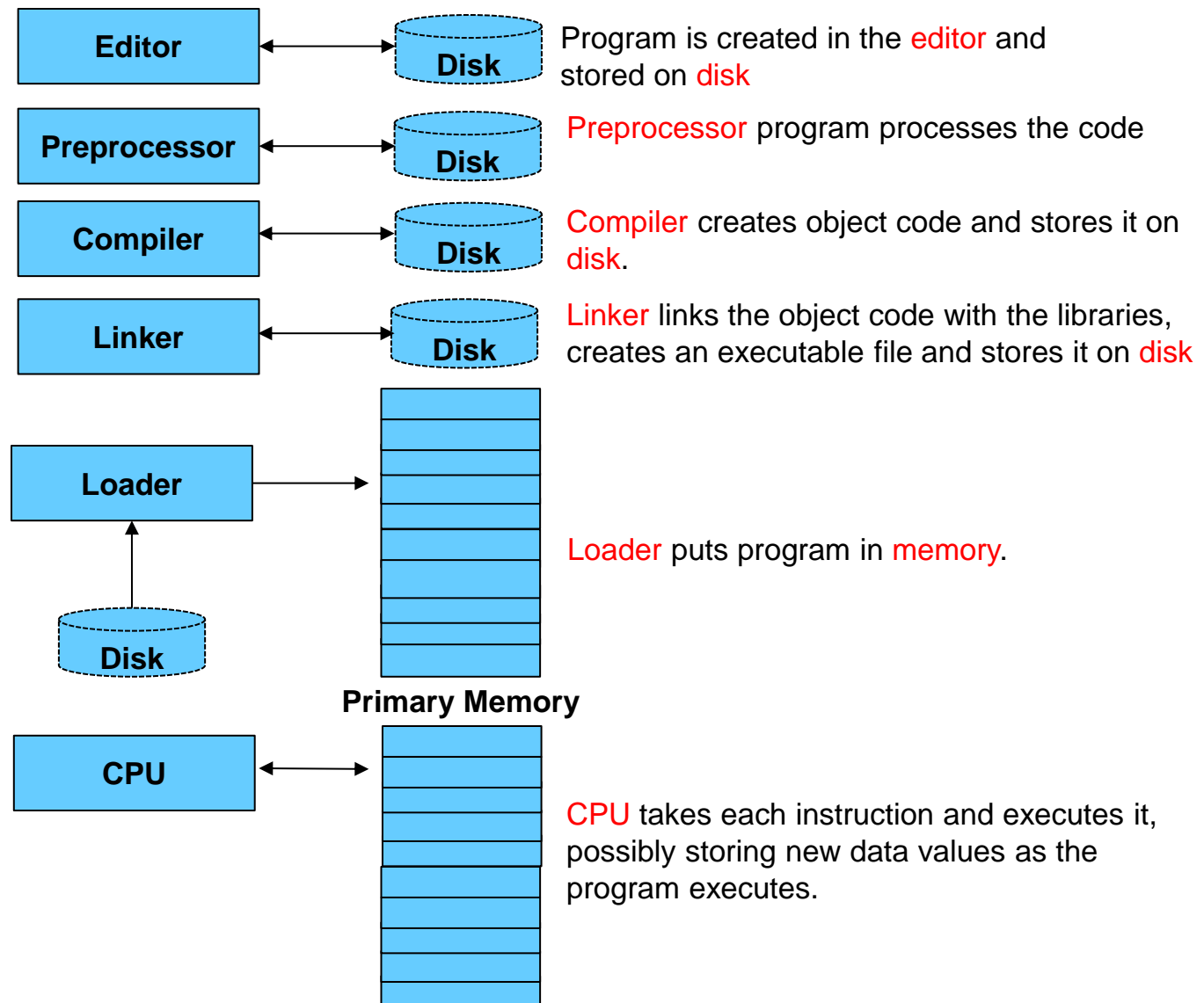
Compute and Print the value of M where:

$$M = 2 \times 4 \times 6 \times \dots \times n$$



Phases of C++ Programs Execution

- Edit
- Preprocess
- Compile
- Link
- Load
- Execute



C++ Programming Language

- C++ standard library
 - Rich collections of existing classes and functions which are written in the core language
 - Can be used at any C++ program
- C++ programs
 - Built from pieces called **classes** and **functions which** can span multiple files
 - Structured into small understandable units to reduce the complexity and decrease program size
 - "Building block approach" to creating programs help in software reuse
- C++ is case sensitive

First C++ Program: Printing a Line of Text

```
// A first program in C++.  
#include<iostream>  
  
//function main begins program execution  
int main()  
{  
    std::cout << "Welcome to C++!\n";  
}
```



Welcome to C++!

First C++ Program: Printing a Line of Text

`// A first program in C++.`

- Comments are ignored by compiler, used to document programs and improve readability
 - Single line comment begin with `//`, and multiple line comments begin with `/*` and end with `*/`

`#include <iostream>`

- Preprocessor directives begin with `#`
 - Processed by **preprocessor** before compiling
 - Causes a copy of the specified header file (`iostream`) to be included in place of the directive
 - `iostream` is standard library header file that must be included if because `cout` is to be used

First C++ Program: Printing a Line of Text

```
int main()
```

- Part of every C++ Program
- **main()** is a function, which begins with left brace ({) and ends with right brace (})

```
std::cout << "Welcome to C++!\n";
```

- **cout** is a standard output stream object found in **iostream**
- **cout** is connected to the screen
- **<<** is the stream insertion operator
 - Value to right (right operand) inserted into output stream (which is connected to the screen)
- **std::** specifies using name that belongs to "**namespace**" **std**
- Escape characters (\): indicates "special" character output

Escape Character

Escape Sequence	Description
<code>\n</code>	Newline. Position the screen cursor to the beginning of the next line.
<code>\t</code>	Horizontal tab. Move the screen cursor to the next tab stop.
<code>\r</code>	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
<code>\a</code>	Alert. Sound the system bell.
<code>\\</code>	Backslash. Used to print a backslash character.
<code>\"</code>	Double quote. Used to print a double quote character.

Example

```
#include <iostream>
int main()
{
    std::cout << "Welcome ";
    std::cout << "to C++!\n";
}
```

Welcome to C++!

Example

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

int main()
{
    cout << "Welcome\n\nC++!\n";
}
```

Welcome

To

C++!

Testing and Debugging

- Bug
 - A mistake in a program
- Debugging
 - Eliminating mistakes in programs

Program Errors

- Syntax errors
 - Violation of the grammar rules of the language
 - Discovered by the compiler
 - Error messages may not always show correct location of errors
- Run-time errors
 - Error conditions detected by the computer at run-time
- Logic errors
 - Errors in the program's algorithm
 - Most difficult to diagnose
 - Computer does not recognize an error

Stream extraction operator (>>)

- When used with `cin`, waits for the user to input a value and stores the value in the variable to the right of the operator
- The user types a value, then presses the *Enter* (Return) key to send the data to the computer
- Example:

```
int myVariable;  
cin >> myVariable;
```

- Waits for user input, then stores input in `myVariable`

Compute and print the summation of two numbers

```
#include <iostream>
using namespace std;
int main() {

    int num1, num2, sum;
    cout << "Please Enter two numbers:\n";
    cin >> num1 >> num2;
    sum = num1 + num2;
    cout << "sum = " << sum << endl;
}
```

```
Please Enter two numbers:
2
3
sum = 5
```

Fundamental C++ Objects

- Integer objects

`short`

`int`

`long`

- Floating-point objects

`float`

`double`

`long double`

– represent real numbers

- Character objects

`char`

– may hold only a single letter, a single digit, or a single special character like `a`, `$`, `7`, `*`.

- Different types allow programmers to use resources more efficiently

Character object type

- ASCII is the dominant encoding scheme
 - ' ' encoded as 32 '+' encoded as 43
 - 'A' encoded as 65 'a' encoded as 97
- Explicit (literal) characters within single quotes:
 'a' 'D' '*'
- Special characters - delineated by a backslash \
 '\n' '\t' '\\'

Memory Concepts

- Variables are names of memory locations
- Correspond to actual locations in computer's memory
- Every variable has name, type, size and value
- When new value placed into variable, overwrites previous value
- Reading variables from memory is nondestructive

```
int num1 = 4;
```

```
int num2 = 7;
```

```
int sum = num1 + num2;
```

num1	4
num2	7
Sum	11

Names (Naming Entities)

- Used to denote program values or components
- A valid name is a sequence of
 - Letters (upper and lowercase)
 - A name cannot start with a digit
- Names are case sensitive
 - MyVar is a different name than MYVAR
- There are two kinds of names
 - Keywords
 - Identifiers

Keywords

- Keywords are words reserved as part of the language
- They cannot be used by the programmer to name things
- They consist of lowercase letters only
- They have special meaning to the compiler

C++ Keywords

and	continue	goto	public	try
and_eq	default	if	register	typedef
asm	delete	inline	reinterpret_cast	typeid
auto	do	int	return	typename
bitand	double	long	short	union
bitor	dynamic_cast	mutable	signed	unsigned
bool	else	namespace	sizeof	using
break	enum	new	static	virtual
case	explicit	not	static_cast	void
catch	export	not_eq	struct	volatile
char	extern	operator	switch	wchar_t
class	false	or	template	while
compl	float	or_eq	this	xor
const	for	private	throw	xor_eq
const_cast	friend	protected	true	

Identifiers

- Used to name entities in C++
- Consists of letters, digits or underscore
 - Starts with a letter or underscore
 - Can not start with a digit
- Identifiers should be:
 - Short enough to be reasonable to type
 - Long enough to be understandable
- Examples
 - **Grade**
 - **Temperature**
 - **CameraAngle**
 - **IntegerValue**

Definitions/declaration

- All variable that are used in a program must be defined (declared)
- A variable definition specifies Type and Identifier
- General definition form: **Type Id;**
- Examples:

```
Char   Response;  
int    MinElement;  
float  Score;  
float  Temperature;  
int    i;  
char   c;  
double x;
```

- Value of a variable is whatever in its assigned memory location
- Memory location is where a variable value can be stored for program use

Type compatibilities

- Store the values in variables of the same type
- This is a type mismatch:

```
int x;  
x = 2.99;
```
- Variable x will contain the value 2, not 2.99

Arithmetic

- Arithmetic is performed with operators.
- Arithmetic operators are listed in following table

C++ operation	Arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	<code>f + 7</code>
Subtraction	-	$p - c$	<code>p - c</code>
Multiplication	*	bm	<code>b * m</code>
Division	/	x / y	<code>x / y</code>
Modulus	%	$r \bmod s$	<code>r % s</code>

- Modulus operator returns the remainder of integer division
 $7 \% 5$ evaluates to 2
- Integer division truncates remainder
 $7 / 5$ evaluates to 1

Results of Arithmetic operators

- Arithmetic operators can be used with any numeric type.
- An **operand** is a number or variable used by the operator
e.g.
 - integer1 + integer2
 - **+** is operator
 - **integer1** and **integer2** are operands
- Result of an operator depends on the types of operands
 - If both operands are **int**, the result is **int**
 - If one or both operands are **double**, the result is **double**

Integer Division

$$\begin{array}{r} 4 \longleftarrow \\ 3 \overline{) 12} \\ \underline{12} \\ 0 \end{array} \quad \begin{array}{r} 12 \\ \hline 3 \end{array}$$

$0 \longleftarrow 12 \% 3$

$$\begin{array}{r} 4 \longleftarrow \\ 3 \overline{) 14} \\ \underline{12} \\ 2 \end{array} \quad \begin{array}{r} 14 \\ \hline 3 \end{array}$$

$2 \longleftarrow 14 \% 3$

Examples on integer division

```
#include <iostream>
using namespace std;
int main( ) {

    cout<< 10/4 <<endl;
    cout<< 10.0/4 <<endl;
    cout<< 10/4.0 <<endl;
}
```



2
2.5
2.5

Comparing mathematical and C++ expressions

Mathematical formula	C++ Expression
$x^2 - 5yz$	$x * x - 5 * y * z$
$x(y + 2z)$	$x * (y + 2 * z)$
$\frac{1}{x^2 + 4y + 3}$	$1 / (x * x + 4 * y + 3)$
$\frac{w + x}{y + 2z}$	$(w + x) / (y + 2 * z)$

Operator precedence

- The order in which an operator is executed
- For example, the multiplication operator (*) is executed before addition operator (+)
- To find the average of three variables a, b and c
 - **Incorrect:** $a + b + c / 3$
 - **Correct:** $(a + b + c) / 3$

Rules of operator precedence

Operator(s)	Operation(s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
*, /, or %	Multiplication Division Modulus	Evaluated second. If there are several, they are evaluated left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Example on Operator Precedence

Evaluate the following arithmetic expression:

$$20 - 10 / 5 * 2 + 3 * 5 \% 4$$

$$1) 10/5 = 2 \rightarrow 20 - 2 * 2 + 3 * 5 \% 4$$

$$2) 2*2 = 4 \rightarrow 20 - 4 + 3 * 5 \% 4$$

$$3) 3*5 = 15 \rightarrow 20 - 4 + 15 \% 4$$

$$4) 15\%4 = 3 \rightarrow 20 - 4 + 3$$

$$5) 20 - 4 = 16 \rightarrow 16 + 3$$

$$6) 16 + 3 = 19$$

Assignment operator (=)

- The (=) operator in C++ is not an equal sign. It assigns a value to a variable
- An assignment statement changes the value of the variable on the left of the assignment operator (=)
- General Form: ***identifier = expression;***
- On the right of the assignment operator can be
 - Constant: **x = 21;**
 - Variable: **x = y;**
 - Expression: **x = y * 2 + z;**
- The following statement is not true in algebra: **i = i + 3;**
 - In C++ it means the **new** value of **i** is the **previous** value of **i** plus 3

Assignment expression abbreviations

- C++ provides several assignment operators for abbreviating assignment expressions, as shown in the table below:

Assignment operator	Sample expression	Explanation	Assigns
Assume: <code>int c = 3, d = 5, e = 4, f = 6, g = 12;</code>			
<code>+=</code>	<code>c += 7</code>	<code>c = c + 7</code>	10 to <code>c</code>
<code>-=</code>	<code>d -= 4</code>	<code>d = d - 4</code>	1 to <code>d</code>
<code>*=</code>	<code>e *= 5</code>	<code>e = e * 5</code>	20 to <code>e</code>
<code>/=</code>	<code>f /= 3</code>	<code>f = f / 3</code>	2 to <code>f</code>
<code>%=</code>	<code>g %= 9</code>	<code>g = g % 9</code>	3 to <code>g</code>

Print the average of three numbers

```
int main() {  
    int n1 , n2 , n3;  
    float s , average;  
    cout << "Please Enter three integers:\n";  
    cin >> n1 >> n2 >> n3;  
    s = n1 + n2 + n3;  
    average = s / 3;  
    cout << "Average = " << average << endl;  
}
```

Please Enter three integers:

1

6

2

Average = 3

Compute the area of a circle, where $\text{area} = \pi \times r^2$

```
int main() {  
    double Pi = 3.14;  
    int r;  
  
    cout<<"Please enter r : ";  
    cin>>r;  
  
    double area;  
    area = Pi * r * r;  
  
    cout<<"Circle's Area = "<< area <<endl;  
}
```

Increment and Decrement Operators

- Increment and decrement operators are unary operators as they require only one operand.
 - ++ unary increment operator: Adds 1 to the value of a variable
 - -- unary decrement operator
 - **x++** is equivalent to **x = x + 1**
 - **x--** is equivalent to **x = x - 1**
- Pre-increment
 - When the operator is used before the variable (**++c**), Variable is changed, then the expression it is in is evaluated
- Post-increment
 - When the operator is used after the variable (**c++**), Expression the variable is in executes, then the variable is changed.

Increment and Decrement Operators

- Example: If `c = 5`, then
 - `cout << ++c;`
 - `c` is changed to **6**, then printed out
 - `cout << c++;`
 - Prints out **5** (`cout` is executed before the increment)
 - `c` then becomes **6**
- When variable not in expression
 - Preincrement and postincrement have same effect
 - `++c;`
 - `cout << c;`
 - and
 - `c++;`
 - `cout << c;`

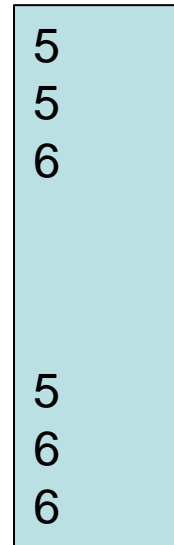
are the same

Increment and decrement operators

Operator	Sample expression	Explanation
++ Preincrement	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++ Postincrement	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
-- Predecrement	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
-- postdecrement	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Understand the effect of pre and post-increment

```
int main() {  
  
    int c;  
  
    c = 5;  
    cout << c << endl;  
    cout << c++ << endl  
    cout << c << endl << endl;  
  
    c = 5;  
    cout << c << endl;  
    cout << ++c << endl;  
    cout << c << endl;  
}
```



5
5
6
5
6
6

Operators Precedence

Operators	Associativity	Type
()	left to right	parentheses
++ --	left to right	unary (postfix)
++ -- + -	right to left	unary (prefix)
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
= += -= *= /= %=	right to left	assignment

Example

```
int main() {  
    int X = 5 , Y = 7 , Z;  
  
    cout<< X++ <<endl;  
    cout<< ++X <<endl;  
  
    Z = X++ + Y;  
    cout <<X<<"\t"<<Y<<"\t"<<Z<<endl;  
  
    Z = ++X + Y;  
    cout <<X<<"\t"<<Y<<"\t"<<Z<<endl;  
  
    Z = X++ + Y++;  
    cout <<X<<"\t"<<Y<<"\t"<<Z<<endl;  
}
```

5		
7		
8	7	14
9	7	16
10	8	16

2 - Control Structures

Control Structures

- Sequence structure: C++ programs executed sequentially by default.
- Selection structures
 - **if** selection structure
 - Perform an action if condition is true.
 - Skip the action if condition is false.
 - **if/else** selection structure
 - Perform an action if condition is true.
 - Performs a different action if the condition is false.
 - **switch** selection structure
 - Perform one of many different actions depending on the value of an integer expression.

Control structures

- Repetition structures
 - **while** repetition structure
 - An action to be repeated while some conditions remains true.
 - **do/while** repetition structure
 - Similar to while structure.
 - Tests the loop continuation after the loop body is performed.
 - **while** tests the loop continuation condition before the loop body is performed.
 - **for** repetition structure
 - used when the number of times to be repeated is fixed/known
 - It handles all the details of the counter controlled repetition
 - Execution continues as long as the condition is true

Condition

- **Condition** is a logical expression that evaluates to **true** or **false**
- Specifies the decision you are making
- Conditions can be formed by using the equality (==) and relational operators (< , > , >= , <= , !=)
- Equality operators precedence is lower then precedence of relational operators.

Arithmetic Expressions

- Composed of operands and arithmetic operations (+ , - , * , / , %)
- evaluates to a numeric value
 - (e.g. $3 + 4$ gives 7)
- Operands may be numbers and/or identifiers that have numeric values

Relational Expressions

- Composed from operands and operators.
- Operands may be numbers and/or identifiers that have numeric values
- Result is a logical value (**true** or **false**).
- Operators are relational operators: $<$, $>$, \leq , \geq , $=$, \neq
- Example:
 - $(a < b)$ gives **true** if value of **a** is less than value of **b**, or gives **false** if value of **a** is not less than value of **b**
 - $(x \neq y)$ gives **true** if **x** does not equal **y** or gives **false** if **x** equal **y**

Equality and relational operators

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
$>$	<code>></code>	<code>x > y</code>	x is greater than y
$<$	<code><</code>	<code>x < y</code>	x is less than y
\geq	<code>>=</code>	<code>x >= y</code>	x is greater than or equal to y
\leq	<code><=</code>	<code>x <= y</code>	x is less than or equal to y
Equality operators			
$=$	<code>==</code>	<code>x == y</code>	x is equal to y
\neq	<code>!=</code>	<code>x != y</code>	x is not equal to y

Boolean variables and relational operations

```
int main( ){  
    bool x , y;  
    x = 5 > 7;  
    cout << "x = " << x << endl;  
    y = 5 < 7;  
    cout << "y = " << y << endl;  
    x = true;  
    cout << "x = " << x << endl;  
    y = false;  
    cout << "y = " << y << endl;  
    x = 5;  
    cout << "x = " << x;  
}
```

x = 0

y = 1

x = 1

y = 0

x = 1

Logical Expressions

- Also called **Boolean expressions**
- Result is a logical value **true** or **false**
- Composed from operands and operators.
- Operands are identifiers that have logical values
- Operators are logical operators:
 - **&&** (**AND**)
 - **||** (**OR**)
 - **!** (**NOT**)
- Example:
 - **x && y**
 - **a && b || c**

Evaluating Logical Expressions

- AND truth table

&&		
True	True	True
True	False	False
False	True	False
False	False	False

- OR truth table

True	True	True
True	False	True
False	True	True
False	False	False

- NOT truth table

!	
True	False
False	True

Arithmetic, Relational and Logical Expressions

- Relational expression may contain arithmetic sub expressions:
 - $(3 + 7) < (12 * 4)$
- Logical expression may contain relational and arithmetic subexpressions:
 - $x \ \&\& \ y \ \&\& \ (a > b)$
 - $(2 + t) < (6 * w) \ \&\& \ (p == q)$

Operators Precedence

Operators	Associativity	Type
()	left to right	parentheses
++ --	left to right	unary (postfix)
++ -- + -	right to left	unary (prefix)
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
&&		
= += -= *= /= %=	right to left	assignment

Operators Precedence

```
int main( ){  
  
    int a = 10, b = 3;  
  
    cout<<"a+b="<<a+b <<endl;  
  
    cout<<"a+b*2= " <<a+b*2 <<endl;  
  
    cout<<" (a+b) *2= " << (a+b) *2<<endl;  
  
    cout<<a<<"<"<<b<<" is " <<(a<b)<<endl;  
  
    cout<<"a+b != a+3 is " <<(a+b != a+3) ;  
}
```

a+b=13

a+b*2= 16

(a+b)*2= 26

10<3 is 0

a+b != a+3 is 0

`if` selection Structure

```
if ( Condition )  
    statement;
```

```
if ( Condition ) {  
    statement1;  
    statement1;  
    statement1;  
    ...  
}
```

Read any number from user, then print positive if it is positive

```
int main()  
{  
    int Num;  
  
    cout<<"Enter an integer Number:";  
  
    cin >> Num;  
  
    if (Num > 0)  
        cout<<" Positive\n";  
}
```


Another Version

```
int main() {  
  
    int Num;  
    bool w;  
  
    cout<<"Enter an integer number:";  
    cin >> Num;  
  
    w = Num > 0;  
  
    if (w)  
        cout<<" Positive\n";  
}
```

if/else selection Structure

```
if ( Condition )  
    statement;  
else  
    statement;
```

```
if ( Condition ){  
    statement1;  
    statement2;  
    ...  
}  
else{  
    statement1;  
    statement2;  
    ...  
}
```

Read a mark, then print "PASS" if it is greater than or equal 50, or print "FAIL" otherwise

```
int main() {  
    int mark;  
  
    cout<<"Please Enter your mark: ";  
    cin >> mark;  
  
    if (mark >= 50)  
        cout<<" PASS\n";  
    else  
        cout<<"FAIL\n";  
}
```

Ternary conditional operator

- Ternary conditional operator (`? :`)
 - Three arguments (condition, value if **true**, value if **false**)

```
cout <<( mark >= 50 ? "PASS\n" : "FAIL\n" );
```

↑	↑	↑
Condition	Value if true	Value if false

- Equivalent to:

```
if (mark >= 50)
    cout<<" PASS\n";
else
    cout<<"FAIL\n";
```

More than one statement in `if`

```
int main() {  
  
    int mark;  
    cout << "Please Enter your mark: ";  
    cin >> mark;  
  
    if (mark >= 50) {  
        cout<<"PASS\n";  
        cout<<"You can take the next course\n";  
    }  
  
    else {  
        cout<<"FAIL\n";  
        cout<<"You must take this course again\n";  
    }  
  
}
```

Write a program to print the fraction a/b in the form $c \ d/b$

```
int main() {  
  
    int a,b,c,d;  
    cout<<"To convert from a/b to c d/b, Enter a,b";  
    cin >> a >> b;  
  
    c = a / b;  
    d = a % b;  
    cout<< a << "/" << b << "=";  
  
    if ( c != 0)  
        cout<<c;  
  
    if (d!=0)  
        cout<<" "<<d<<"/"<<b;  
}
```

Read any number, then print "positive" if it is positive and "negative" otherwise.

```
int main() {  
  
    int Num;  
    cout<<"Please Enter Number:";  
    cin>>Num;  
  
    if (Num < 0)  
        cout<<"Negative\n";  
    else  
        cout<<"Positive\n";  
}
```

Equality (==) and Assignment (=) Operators

```
int main() {  
  
    int x = 0;  
  
    if (x = 0)  
        cout<<"condition is true\n";  
  
    else  
        cout<<"condition is false\n";  
}
```

condition is false

Read two numbers and print the largest

```
int main() {  
    int x,y;  
    cout<<"Enter two numbers:";  
    cin>>x>>y;  
    cout<<"Max = ";  
  
    if (x > y)  
        cout<<x<<endl;  
    else  
        cout<<y<<endl;  
}
```

```
Enter two numbers:15  
4  
Max = 15
```

Read three numbers and print the smallest

```
int main() {  
  
    int a, b, c;  
    cout<<"Enter three numbers:\n";  
    cin>>a>>b>>c;  
    cout<<"Min = ";  
    if ((a < b) && (a < c))  
        cout<<a;  
    if ((b < a) && (b < c))  
        cout<<b;  
    if ((c < a) && (c < b))  
        cout<<c;  
}
```

```
Please Enter three numbers: 8  
3  
6  
Min = 3
```

Read three numbers and print the smallest

```
int main() {  
    int a, b, c;  
    cout<<"Please Enter three numbers:";  
    cin>> a >> b >> c;  
    cout<< "Min = ";  
    int min = a;  
    if (b < min)    min = b;  
    if (c < min)    min = c;  
    cout<<min;  
}
```

```
Please Enter three numbers: 8  
3  
6  
Min = 3
```

Read three numbers and print the smallest, use nested **if**

```
int main() {  
    int a, b, c;  
    cout<<"Please Enter three numbers: ";  
    cin>>a>>b>>c;  
    cout<<"Min = ";  
    if (a < b)  
        if (a < c)    cout<<a;  
        else          cout<<c;  
    else  
        if (b < c)    cout<<b;  
        else          cout<<c;  
}
```

```
Please Enter three numbers: 5  
11  
9  
Min = 5
```

Read a number, if it is positive add **10** to it and print Number "is positive", otherwise, subtract 10 and print Number "is negative"

```
int main() {  
    int Number;  
    cout<<"Please enter Number:";  
    cin>>Number;  
  
    if (Number>0) {  
        Number = Number + 10;  
        cout<<Number<<" is Positive\n";  
    }  
  
    else {  
        Number = Number - 10;  
        cout<<Number<<" is Negative\n";  
    }  
}
```

Dangling else

```
int main() {  
  
    int x = 2 , y = 5 , z = 10;  
    if ( x > y)  
        if ( x < z)  
            cout << " Hello";  
        else  
            cout << "Hi";  
}
```



Nothing is printed

Multiple Selection Structure (**switch**)

- Test variable for multiple values
- Series of **case** labels and optional **default** case

```
switch ( variable ) {  
    case value1: // taken if variable = value1  
        statements  
        break;    // necessary to exit switch  
  
    case value2:  
    case value3: //taken if variable = value2 or = value3  
        statements  
        break;  
  
    default:    //taken if variable matches no other case  
        statements  
        break;  
}
```

Example 1

```
int main() {
    int a;
    cout<<" Enter an Integer between 0 and 10: ";
    cin>>a;

    switch(a) {
        case 0: case 1:  cout<<"hello ";
        case 2:  cout<<"there ";
        case 3:  cout<<"Welcome to ";
        case 4:  cout<<"C++ "<< endl; break;
        case 5:  cout<<"How ";
        case 6: case 7: case 8:  cout<<"are you "<<endl; break;
        case 9: break;
        case 10: cout<<"Have a nice day. "<<endl; break;
        default: cout<<"the number is out of range"<<endl;
    }

    cout<< "Out of switch structure."<<endl;
}
```


Example 2

```
int main( ) {  
    int score;  
    char grade;  
    cin>>score;  
    switch(score/10) {  
        case 0:case 1:case 2:case 3:case 4:case 5: grade='F';  
        break;  
        case 6: grade = 'D'; break;  
        case 7: grade = 'C'; break;  
        case 8: grade = 'B'; break;  
        case 9: case 10: grade = 'A'; break;  
        default: cout<<"Invalid test score."<<endl;  
    }  
  
    cout<<"Grade is"<<grade<<endl;  
}
```

Example 3

```
int main( ) {  
  
    char grade;  
    cout <<" Enter grade as a letter : " ;  
    cin>>grade;  
  
    switch (grade) {  
  
        case 'A': cout<<"The Grade is A"; break;  
        case 'B': cout<<"The Grade is B"; break;  
        case 'C': cout<<"The Grade is C"; break;  
        case 'D': cout<<"The Grade is D"; break;  
        case 'F': cout<<"The Grade is F"; break;  
        default: cout<< "The Grade is invalid";  
  
    }  
}
```

Example 4

```
int main( ){
    int age;
    cout<<"Enter your age: ";
    cin>>age;

    switch (age >= 18){

    case 1:
        cout<<"old enough to drive"<<endl;
        cout<<"old enough to vote."<<endl;
        break;

    case 0:
        cout<<"Not old enough to drive"<<endl;
        cout<<"Not old enough to vote."<<endl;
    }
}
```

Quiz

- Write a program to read two numbers (a and b) and one character (op). The program then uses switch statement to print the output according to the table below:

op	output
+	a+b
-	a-b
*	a*b
/	a/b
otherwise	"Invalid Operation"

for Repetition Structure

- General format:

```
for ( initialization; condition; increment )  
    statement;
```
- Statements will be executed repeatedly while condition is true.
- When the condition become false, the loop will be terminated and the execution sequence will go the first statement after for loop.
- If the loop body contains only one statement, there is no need to begin { and end } the loop body.

Examples

```
for( int c = 1; c <= 5; c++ )  
    cout << c << endl;
```

1
2
3
4
5

```
for (int i = 1; i <= 5; i++)  
    cout<<"Amman\n";
```

Amman
Amman
Amman
Amman
Amman

```
for (int i = 5; i >=1 ; i--)  
    cout<<"Amman\n";
```

Amman
Amman
Amman
Amman
Amman

Print the following numbers:

1 3 5 7 9 11

```
for (int k=1; k<=11; k+=2)
    cout<<k<<"\t";
```

Print the following numbers

20 17 14 11 8 5 2

```
for (int m=20; m>=2; m-=3)
    cout<<m<<"\t";
```

Print the following numbers

1 2 3 4 ... n (entered by user)

```
int main( ) {  
    int n;  
    cout<<"Enter the upper limit:";  
    cin >> n;  
  
    for (int i=1; i<=n; i++)  
        cout<<i<<"\t";  
  
    cout<<endl;  
}
```


Print the following numbers

a (a + 1) (a + 2) ... b (**a** and **b** are entered by user)

```
int main( ) {  
  
    int a,b;  
    cout<<"Enter the start value:";  
    cin>>a;  
  
    cout<<"Enter the end value:";  
    cin>>b;  
  
    for (int i=a; i<=b; i++)  
        cout<<i<<"\t";  
  
}
```

Read five numbers from user and print the positive numbers only

```
int main( ) {  
    int num;  
  
    for (int i=1; i<=5; i++){  
        cout<<"Please Enter No "<<i<<':';  
        cin>>num;  
  
        if (num > 0)  
            cout<<num<<" is positive\n";  
    }  
}
```

Compute and print S , Where $S = 1 + 2 + 3 + 4 + 5$

```
int S=0;
for (int i=1; i<=5; i++)
    S += i;
cout<<"Sum is "<<S<<endl;
```

Compute and print S , Where $S = 1 + 3 + 5 + 7 + \dots + n$

```
int Sum=0, n;
cout<<"Please Enter n";
cin>>n;
for (int i=1; i<=n; i += 2)
    Sum += i;
cout<<"Sum="<<Sum<<endl;
```

Compute and print the summation of any 10 numbers entered by the user

```
int main() {  
  
    int S=0, N;  
    for (int i = 10; i >= 1; i--) {  
        cout<<"Enter the next number:";  
        cin>>N;  
        S += N;  
    }  
  
    cout<<"Sum = "<< S <<endl;  
}
```

Compute and Print the factorial of 5, where:

$$fact(5) = 5 \times 4 \times 3 \times 2 \times 1$$

```
int main( ) {  
  
    int Fact=1;  
  
    for (int j = 5; j >= 1; j--)  
        Fact *= j;  
  
    cout<<"5!    = "<<Fact<<endl;  
}
```

Compute and Print the factorial of n, where

$$fact(n) = n \times n - 1 \times n - 2 \times \dots \times 1$$

```
int main( ) {  
  
    int Fact = 1, n;  
    cout<<"Enter an integer: ";  
    cin>>n;  
  
    for (int j=n; j>=1; j--)  
        Fact *= j;  
  
    cout<< n <<"! = " << Fact << endl;  
}
```

Compute and Print the value of M where:

$$M = 2 \times 4 \times 6 \times \dots \times n$$

```
int main( ) {  
  
    long M = 1;  
    int n;  
    cout<<"please enter the upper Limit:";  
    cin>>n;  
  
    for (int i=2; i<=n; i += 2)  
        M *= i;  
  
    cout<<"M = "<< M <<endl;  
}
```

Quiz

- Write a program that prints the numbers from X to Y, with step Z , using **for** statement. The program should read X, Y, Z then start the loop

Compute and Print M^n

```
int main() {  
  
    long Result = 1;  
    int M, n;  
    cout<<"Enter the Base number:";  
    cin>>M;  
    cout<<"Enter the exponent:";  
    cin>>n;  
  
    for (int i=1; i<=n; i++)  
        Result *= M;  
  
    cout<<"Result= "<<Result<<endl;  
}
```

Quiz

- Write a program that finds M^n for positive and negative n

While Repetition Structure

```
initialization;  
while (Condition) {  
    statements;  
    increment;  
}
```

- Statements will be executed repeatedly while condition is true
- When the condition become false, the loop will be terminated and the execution sequence will go to the first statement after While loop
- If the loop body contains only one statement, there is no need to begin { and end } the loop body.

Print the word "Amman" five times

```
int main( ) {  
  
    int i=1;  
  
    while (i<=5) {  
        cout<<"Amman\n";  
        i++;  
    }  
}
```

Print the word "Amman" five times

```
int main() {  
  
    int i=1;  
  
    while (i++ <= 5)  
        cout<<"Amman\n";  
  
    cout<<i<<endl;  
}
```

```
Amman  
Amman  
Amman  
Amman  
Amman  
7
```

Print the following numbers

1 3 5 7 9 11

```
int main() {  
  
    int i=1;  
  
    while (i <= 11) {  
        cout<<i<<'\\t';  
        i+=2;  
    }  
}
```

Print the following numbers

20 17 14 ... n

```
int main() {  
  
    int n, k=20;  
    cout<<"Enter the lower limit:";  
    cin>>n;  
  
    while ( k >= n) {  
        cout<<k<<'\\t';  
        k -= 3;  
    }  
  
    cout<<endl;  
}
```

Read five numbers from the user and print the positive numbers only

```
int main() {  
  
    int num, j=0;  
  
    while ( j++ < 5 ) {  
        cout<<"Enter a number:";  
        cin>>num;  
  
        if (num > 0)  
            cout<<num<<endl;  
    }  
  
}
```


Sum of numbers from x to y

```
int main() {  
    int sum = 0, i, x, y;  
    cout<<"Enter First Number: ";  
    cin >> x;  
    cout<<"Enter Second Number: ";  
    cin >> y;  
  
    i = x;  
    while ( i <= y) {  
        sum = sum + i;  
        i = i+1;  
    }  
    cout<<"Sum from "<<x<<" to "<<y<<" = "<<sum;  
}
```

```
Enter First Number: 5  
Enter Second Number: 8  
Sum from 5 to 8 = 26
```

Compute and print *sum*, Where
$$sum = 1 + 3 + 5 + 7 + \dots + n$$

```
int main() {  
  
    int n, Sum=0, i=1;  
    cout<<"Enter the upper limit:";  
    cin>>n;  
  
    while ( i <= n ) {  
        Sum += i;  
        i += 2;  
    }  
  
    cout<<"Sum="<<Sum<<endl;  
}
```

Read 10 numbers and compute the sum of numbers divisible by 3

```
int main() {  
  
    int Num, Sum=0, i=1;  
  
    while ( i <= 10 ) {  
        cout<<"Enter a number:";  
        cin>>Num;  
  
        if (Num % 3 == 0)  
            Sum += Num;  
        i++;  
    }  
  
    cout<<"\nSum="<<Sum;  
}
```

Compute and Print the value of M where:

$$M = 2 \times 4 \times 6 \times \dots \times n$$

```
int main() {  
  
    int N, M=1, i=2;  
    cout<<"Enter the upper limit:";  
    cin>>N;  
  
    while ( i <= N ) {  
        M *= i;  
        i += 2;  
    }  
  
    cout<<"\nM="<<M;  
}
```

Do While Repetition Structure

```
initialization  
do {  
    Statement(s) ;  
} while (Condition) ;
```

- Statements will be executed repeatedly while condition is true
- When condition become false, the loop will be terminated and the execution sequence will go to the first statement after the loop
- The loop body will be executed at least once.

Print the word "Amman" five times

```
int main( ) {  
  
    int i = 1;  
  
    do {  
        cout<<"Amman\n";  
        i++;  
    } while (i <= 5);  
  
}
```

Program to read an integer then prints if it is *Even* or *Odd*.
The program keeps running until number 1 is entered

```
int main() {
    int Choice, Num;
    do {
        cout <<"\nEnter a Number: ";
        cin >> Num;

        if (Num%2 == 0)
            cout<<Num<<" is Even\n";
        else
            cout<<Num<<" is Odd\n";

        cout<<"Enter 1 to Exit program\n";
        cout<<"Enter any other number to repeat\n";
        cin>>Choice;
    } while (Choice != 1); }
```

Modifying previous program such that 'Y' is entered to continue program and any other character to end

```
int main() {  
    int Num;  
    char Choice;  
    do {  
        cout<<"\nEnter a Number: ";  
        cin >> Num;  
  
        if (Num%2 == 0)  
            cout<<Num<<" is Even\n";  
        else  
            cout<<Num<<" is Odd\n";  
        cout<<"Enter Y to continue\n";  
        cout<<"Enter any other character to end program\n";  
        cin>>Choice;  
    } while (Choice == 'Y');  
}
```


Modify previous program such that 'Y' or 'y' is entered to continue

```
int main() {
    int Num;
    char Choice;
    do {
        cout<<"\nEnter a Number";
        cin >> Num;
        if (Num%2 == 0)
            cout<<Num<<" is Even\n";
        else
            cout<<Num<<" is Odd\n";
        cout<<"Enter Y to continue\n";
        cout<<"Enter any other character to end program\n";
        cin>>Choice;
    } while ((Choice == 'Y') || (Choice == 'y'));
}
```

break Statement

- Immediate exit from **while**, **for**, **do/while**, **switch**
- Program continues with first statement after structure
- Used to escape early from a loop
- Skip the remainder of **switch**

Example

```
int main() {  
    int x;  
    for (x = 1; x <= 10; x++) {  
        if (x == 5)  
            break;  
        cout << x << " ";  
    }  
    cout<<endl;  
    cout<<"Broke out of loop when x became "<<x <<endl;  
}
```

1 2 3 4

Broke out of loop when x became 5

Read a number and print "Prime" if it is a prime number, or "Not prime" otherwise

```
int main() {
    bool Prime = true;
    int i, num;
    cout<<"Please enter the number:";
    cin>>num;

    for ( i=2; i<num; i++)
        if (num%i==0) {
            Prime = false;
            break;
        }
    if (Prime)
        cout<<num<<" is a Prime number\n";
    else
        cout<<num<<" is not a Prime number\n";
}
```

`continue` Statements

- Used in `while`, `for`, `do/while`
- Skips remainder of loop body
- Proceeds with next iteration of loop

Example

```
int main() {  
    for (int x = 1; x <= 10; x++) {  
        if (x == 5)  
            continue;  
        cout << x << " ";  
    }  
  
    cout<<endl;  
    cout<<"skipped printing the value 5";  
}
```

```
1 2 3 4 6 7 8 9 10  
skipped printing the value 5
```

Read five numbers from user then print the positive numbers only (use **continue**)

```
int main( ) {  
    int num;  
  
    for (int i=1; i<=5; i++){  
        cout<<"Please Enter No "<<i<<':';  
        cin>>num;  
  
        if (num < 0)  
            continue;  
        cout<<num<<" is positive\n";  
    }  
}
```

Nested for

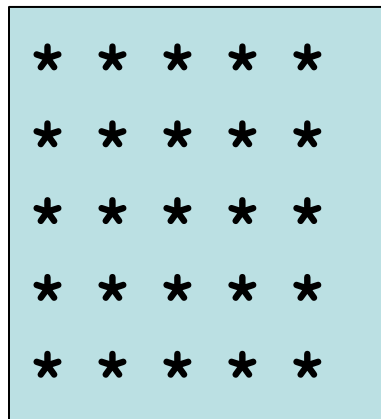
- **for** repetition structure that rests entirely within another **for** repetition structure

Outer loop \longrightarrow **for**(initialization; condition; increment)
 Inner loop \longrightarrow **for**(initialization; condition; increment)
 statement

- If the outer loop will repeat m times and the inner loop will repeat n times, then each statement in the inner loop will be executed $m \times n$ times

Nested for Example 1

```
for ( int i = 1; i <= 5 ; i++) {  
    for (int j = 1 ; j <= 5 ; j++)  
        cout << "*" ;  
    cout << endl ;  
}
```

A 5x5 grid of asterisks, representing the output of the nested for loop. The grid is contained within a light blue rectangular box with a thin black border. Each row contains five asterisks, and there are five rows in total.

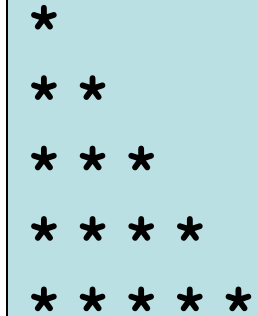
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*

Nested for Example 2

```
int main() {  
  
    for(int i=1;i<=5;i++) {  
        for (int j=1;j<=5;j++)  
            cout<<i<<" , "<<j<<"    ";  
        cout<<endl;  
    }  
}
```

1,1	1,2	1,3	1,4	1,5
2,1	2,2	2,3	2,4	2,5
3,1	3,2	3,3	3,4	3,5
4,1	4,2	4,3	4,4	4,5
5,1	5,2	5,3	5,4	5,5

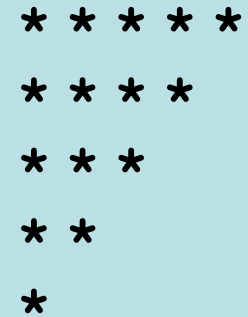
Draw the following shape:



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

```
for (int r = 1 ; r <= 5; r++) {  
    for (int C = 1; C <= r; C++)  
        cout<<'*';  
    cout<<endl;  
}
```

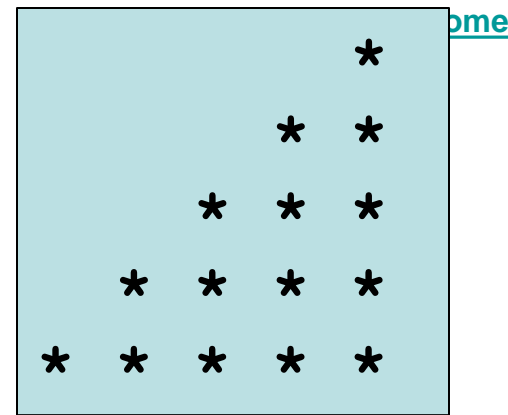
Draw the following shape:



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

```
for (int r = 1; r <= 5; r++) {  
    for (int c = r; c <= 5; c++)  
        cout<<'*';  
    cout<<endl;  
}
```

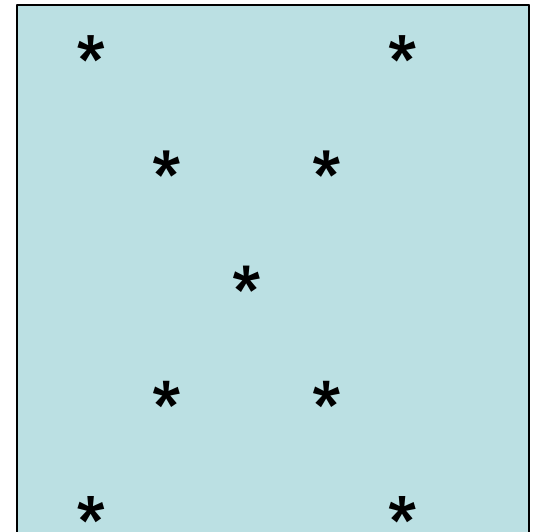
Draw the following shape:



```
for (int i = 1 ; i <= 5; i++) {  
    for(int k = i ; k < 5 ; k++)  
        cout<<" ";  
    for (int j = 1; j <= i; j++)  
        cout<<'*';  
    cout<<endl;  
}
```

What is the output for the following program

```
for (int i = 1 ; i <= 5 ; i++) {  
    for (int j = 1; j <= 5; j++)  
        if (i == j)  
            cout<<"*";  
        else  
            if (i+j == 6)  
                cout<<"*";  
            else  
                cout<<" ";  
    cout<<endl;  
}
```



Using nested **for**, display the multiplication table for the number 3

```
for (int i=1; i<=10; i++)  
    cout<<"3 x "<<i<<" = "<<3*i<<endl;
```

calculate S , where $S = m^0 + m^1 + \dots + m^n$

```
int main() {
    int s=0,n,m,t;
    cout<<"Enter m please ";
    cin>>m;
    cout<<"Enter n please ";
    cin>>n;

    for (int i = 0 ; i <= n ; i++) {
        t = 1;
        for (int j = 1 ; j <= i ; j++)
            t = t * m;
        s = s + t;
    }
    cout<<s<<endl;
}
```

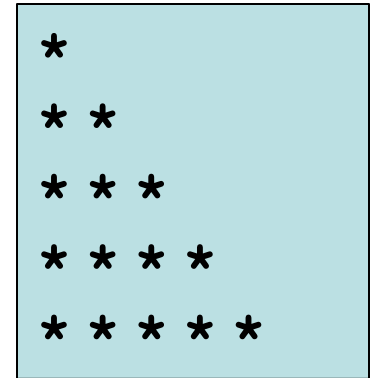

Nested while

```
int main() {  
    int j = 1;  
    while (j <= 4) {  
        int i = 1;  
        while (i <= 4) {  
            cout<<"*"<<"\t";  
            i++;  
        }  
        j++;  
        cout<<endl;  
    }  
}
```



*	*	*	*
*	*	*	*
*	*	*	*
*	*	*	*

Draw the following shape using nested while



```
int main() {  
    int i=1;  
    while (i<=5) {  
        int j=1;  
        while (j<=i){  
            cout<<'*';  
            j++;  
        }  
        cout<<endl;  
        i++;  
    }  
}
```

3 - Functions

Modular programming with Functions

- Large program can be constructed from smaller pieces (modules) which are more manageable than the original program
- Modules in C++ are functions and classes
- Function is a group of statements that is executed when it is called from some point of the program
- Function can be called multiple times in a program

Advantages of modular programming

- Functions in C++, make programs
 - Easier to understand
 - Easier to change
 - Easier to write
 - Easier to test
 - Easier to debug
 - Easier for teams to develop

Pre-packaged and Programmer defined functions

- Pre- packaged functions are provided as part of the C++ programming environment
- available in standard C++ library which provides rich collection of functions for performing:
 - Common mathematical calculations
 - String manipulations
 - Character manipulations
 - Input/output, error checking and many other useful operations
- Programmer defined functions
 - Programmer can write functions to define specific tasks
 - Could be used at many points in a program
 - Actual statements defining the function are written only once
 - These statements are hidden from other functions

Function Libraries

- Predefined functions are found in libraries
- The library must be "included" in a program to make the functions available
- An include directive tells the compiler which library to include.
- To include the math library containing `sqrt()`:

```
#include <cmath>
```

Math library functions

- Allow the programmer to perform common mathematical calculations
- Example: `cout << sqrt(49.0) ;`
 - Calls the sqrt (square root) function and print 7
 - The sqrt function takes an argument of type double and returns a result of type double, as do all functions in the math library
- Function arguments can be
 - Constants: `sqrt(4) ;`
 - Variables: `sqrt(x) ;`
 - Expressions: `sqrt(3y + 6) ;`

Commonly used math library functions

Function	Description	Example
<code>ceil(x)</code>	rounds x to the smallest integer not less than x	<code>ceil(9.2)</code> returns 10.0 <code>ceil(-9.8)</code> returns -9.0
<code>cos(x)</code>	trigonometric cosine of x (x in radians)	<code>cos(0.0)</code> returns 1.0
<code>exp(x)</code>	exponential function e^x	<code>exp(1.0)</code> returns 2.71828 <code>exp(2.0)</code> returns 7.38906
<code>fabs(x)</code>	absolute value of x	<code>fabs(5.1)</code> returns 5.1 <code>fabs(-8.76)</code> returns 8.76
<code>floor(x)</code>	rounds x to the largest integer not greater than x	<code>floor(9.2)</code> returns 9.0 <code>floor(-9.8)</code> returns -10.0
<code>fmod(x,y)</code>	remainder of x/y as a floating-point number	<code>fmod(13.657, 2.333)</code> returns 1.992
<code>log(x)</code>	natural logarithm of x (base e)	<code>log(2.718282)</code> returns 1.0 <code>log(7.389056)</code> returns 2.0
<code>log10(x)</code>	logarithm of x (base 10)	<code>log10(10.0)</code> returns 1.0 <code>log10(100.0)</code> returns 2.0
<code>pow(x,y)</code>	x raised to power y (x^y)	<code>pow(2, 7)</code> returns 128 <code>pow(9, 0.5)</code> returns 3
<code>sin(x)</code>	trigonometric sine of x (x in radians)	<code>sin(0.0)</code> returns 0
<code>sqrt(x)</code>	square root of x	<code>sqrt(900.0)</code> returns 30.0 <code>sqrt(9.0)</code> returns 3.0
<code>tan(x)</code>	trigonometric tangent of x (x in radians)	<code>tan(0.0)</code> returns 0

Example1

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

int main() {

    double x;
    cout<<"enter a number to find its square root: ";
    cin>>x;
    cout<<"square root = "<< sqrt(x) ;

}
```

```
enter a number to find its square root: 25
square root = 5
```

Example 2

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

int main() {

    double A = 6.1, B = 2.3, C = -8.14;

    cout<<"fmod ("<<A<<" , "<<B<<" )="<<fmod (A,B) <<endl;

    cout<<"pow (ceil ("<<C<<" ) , 2)="<<pow (ceil (C) , 2) <<endl;

}
```

fmod(6.1,2.3) = 1.5 pow(ceil(-8.14),2) = 64
--

Random number generator

- `rand()`: returns a random **integer** between **0** and **32767**

```
int main() {  
    int x;  
    for(int i = 1; i <= 100; i++) {  
        x = rand();  
        cout<<x<<endl;  
    }  
}
```

```
41  
18467  
6334  
26500  
19169  
15724  
11478  
29358  
26962  
24464
```

Generate 12 random numbers between 0 and 9

```
int main() {  
    int x;  
    for(int i=1;i<=12;i++) {  
        x = rand()%10;  
        cout << x << endl;  
    }  
}
```



1
7
4
0
9
4
8
8
2
4
5
5

Generate 15 random numbers between 1 - 10

```
int main() {  
    int x;  
    for( int i=1 ; i <= 15 ; i++) {  
        x = rand()%10 + 1;  
        cout << x << endl;  
    }  
}
```



2
8
5
1
10
5
9
9
3
5
6
6
2
8
2

Generate 10 random numbers between 5 - 7

```
int main() {  
    int x;  
    for( int i=1 ; i<=10 ; i++) {  
        x = rand()%3 + 5;  
        cout<<x<<endl;  
    }  
}
```



7
7
6
6
7
6
5
5
6
7

Using srand function

```
int main() {  
  
    int x, seed;  
    cout << "Enter a seed:";  
    cin >> seed;  
    srand(seed);  
  
    for(int i=1 ; i<=5 ; i++){  
        x = rand();  
        cout << x << endl;  
    }  
}
```

Enter a seed:5

54

28693

12255

24449

27660

Enter a seed:11

74

27648

21136

4989

24011

Generate 1200 random numbers between 1-6,
then count the occurrence of each number

```
int main(){
    int x, c1=0, c2=0, c3=0, c4=0, c5=0, c6=0;
    for( int i = 1; i <= 1200 ; i++){
        x = rand()%6 + 1;
        if (x == 1) c1++;
        if (x == 2) c2++;
        if (x == 3) c3++;
        if (x == 4) c4++;
        if (x == 5) c5++;
        if (x == 6) c6++;
    }
    cout<<1<<" : "<<c1<<endl;
    cout<<2<<" : "<<c2<<endl;
    cout<<3<<" : "<<c3<<endl;
    cout<<4<<" : "<<c4<<endl;
    cout<<5<<" : "<<c5<<endl;
    cout<<6<<" : "<<c6<<endl;
}
```

1	:	190
2	:	188
3	:	214
4	:	207
5	:	200
6	:	201

Generate 1200 random numbers between 1 and 6,
then print the count of even and odd numbers

```
int main() {
    int x , even = 0, odd = 0;
    for( int i = 1; i <= 1200 ; i++) {
        x = rand() % 6 + 1;
        switch( x ) {
            case 1: case 3: case 5:
                odd++;
                break;
            case 2: case 4: case 6:
                even++;
        }
    }
    cout<<"Even count is "<<even<<endl;
    cout<<"Odd count is "<<odd<<endl;
}
```

Programmer defined functions

- Functions modularize a program
- function can be called multiple times
 - Software reusability
- Local variables
 - Known only in the function in which they are defined
 - All variables declared in function definitions are local variables
- Parameters
 - Local variables passed to function when called
 - Provide outside information

Programmer defined functions

- Each program consists of a function called **main**
- programmer can write there own customized functions
- Programmer defined functions have 2 important components:
 - Function definition
 - Describes how function does its task
 - Can appear before or after the function is called
 - Function prototype

Function Definition

```
return-value-type  function-name( parameter-list )  
{  
    declarations and statements  
}
```

- **Function-name:** any valid identifier
- **Return-value-type:**
 - The data type of the result returned from the function.
 - Return value type **void** indicated that the function does not return a value
- **Parameter-list**
 - comma-separated list of the arguments
 - contains the declarations of the parameters received by the function when it is called
 - If the function does not receive any values, **parameter-list** is **void** or simply left **empty**

Example

```
int square(int y){ return y * y; }
```

- **return** keyword
 - Format **return** *expression*;
 - Returns the value calculated by the function
 - Returns data, and control goes to function's caller
 - If no data to return, use **return**;
 - Function ends when reaches right brace
 - Control goes to caller
- Functions cannot be defined inside other functions

Function Prototype

- Must appear in the code before the function can be called
- The compiler use function prototypes to validate function call

- Format:

Return-type Function_Name (Parameter_List) ;

- Function prototype tells the compiler
 - The function's name
 - Type of data returned by the function (void if return nothing)
 - Number of parameters the function accepts to receive
 - Types of parameters
 - The order in which these parameters are expected

Function prototype

- Function prototype is not required if the function definition appears before the first use function's first use in the program
- Prototype must match function definition

- Function prototype

```
double max(double, double, double);
```

- Definition

```
double max(double x, double y, double z)
{
    ...
}
```


Forms of functions

- Functions that take inputs and returns output:
 - `double square(double)`
 - `int max(int a, int b, int c)`
- Functions that take inputs but don't return output
 - `void Printsum(int x, int y)`
- Functions that don't take any inputs but returns an output:
 - `int Read_number(void) ;`
- Functions that don't take and inputs and don't return any input
 - `void Print_hello(void) ;`

Function call

- Function is invoked by function call
- A function call specifies the function name and provides information (as arguments) that the called function needs
- Functions called by writing
 - `functionName (argument) ;`
 - or
 - `functionName (argument1, argument2, ...) ;`
- The argument can be a constant, variable or expression

Example program

```
int square( int );           //function prototype

int main()  {
    for ( int x = 1; x <= 7; x++ )
        cout << square(x) << "    "; //function call
    cout << endl;
}

int square(int y) {          //function definition
    return y * y;
}
```

1	4	9	16	25	36	49
---	---	---	----	----	----	----

Example Program

```
double maximum( double, double, double ); //function prototype
int main() {
    double n1;
    double n2;
    double n3;
    cout << "Enter three floating-point numbers:";
    cin >> n1 >> n2 >> n3;
    cout << "Maximum is: " << maximum( n1,n2,n3 ) << endl;
}

double maximum( double x, double y, double z ) {
    double max = x;
    if ( y > max )
        max = y;
    if ( z > max )
        max = z;
    return max;
}
```

```
Enter three floating-point numbers:7.1
2.9
5.5
Maximum is: 7.1
```

Function signature

- Function signature is also called simply signature
- It is the portion of a function prototype that includes
 - name of function
 - parameters (types of its argument)
- Example

```
double maximum( double, double, double );
```

Function signature

Argument Correction

- Argument values that do not correspond precisely to the parameter types are converted to proper type before the function is called
- `cout<<sqrt(4)`
 - the compiler converts `int` value `4` to the `double` value `4.0` before the value is passed to `sqrt`
- Changing from `double` to `int` can truncate data e.g. 3.5 to 3
- Some of the conversions may lead to incorrect results if proper promotion rules are not followed

Promotion rules

- How types can be converted to other types without losing data
- Applies to expressions containing values of two or more data types in which, type of each value is promoted to the highest type in the expression
- Also used when the type of an argument to a function does not match the parameter type specified in the function definition
- Converting values to lower type can result in incorrect values

Promotion hierarchy for built in data types

Data types
long double
double
float
unsigned long
long
unsigned
int
unsigned short
Short
unsigned char
char
bool

Type casting

- A value can be converted to a lower type only by explicitly assigning the value to a variable of lower type by using the cast operator
- Example: `static_cast<double>(total)` // total is of type integer
 - produces a `double` representing the integer value of the variable total (operand in parenthesis)
 - `double` is higher type and `int` is lower type
- Converting values to lower type can result in incorrect values

Header Files

- Library header file
 - Contain function prototypes for library functions
 - Definition of various data type and constants needed by library functions
 - Examples: `<cstdlib>` , `<cmath>` , `<iostream>`.
 - Load with `#include<filename>`: `#include<cmath>`
- Custom header files
 - Defined by the programmer
 - Save as `filename.h`
 - Included in a program using `#include` preprocessor directive
 - Example: `#include "square.h" //programmer defined header file`
- The programmer defined header file should end with `.h`

Write a function that takes the length and width of a rectangle and returns the area

```
double area(double, double);  
void main( ) {  
    double L,W,A;  
    cout<<"Enter the length:";  
    cin>> L;  
    cout<<"Enter the width:";  
    cin>> W;  
    A = area(L,W);  
    cout<<"The area of the rectangle is "<<A<<endl;  
}  
double area(double X,double Y) {  
    double Z;  
    Z = X*Y;  
    return Z;  
}
```

Write a function that takes three integers and returns the maximum one

```
int max(int, int, int);
int main() {
    int x, y, z;
    cout<<"Enter 3 numbers please :";
    cin>> x >> y >> z;
    cout<<"The Maximum is "<< max(x,y,z) <<endl;
}
int max(int a,int b,int c){
    int m = a;
    if (m < b) m=b;
    if (m < c) m=c;
    return m;
}
```

```
Enter 3 numbers please :5
9
6
The Maximum is 9
```

Write a function that takes an integer and returns **true** if it is a prime number and **false** otherwise

```
bool prime(int) ;
int main() {
    int x;
    cout << "Enter a number please :";
    cin >> x;
    if (prime(x))
        cout<< x <<" is a prime number\n";
    else
        cout << x <<" is not a prime number\n";
}
bool prime(int a) {
    bool p = true;
    for(int i=2 ;i < a; i++)
        if (a%i == 0) { p = false; break; }

    return p;
}
```

Write a program that uses a function to calculate and print the sum of two numbers.

```
void print_sum( int,int );
int main() {
    int x,y;
    cout<<"Enter two numbers please :";
    cin>>x>>y;
    print_sum(x,y);
    print_sum(y,x);
    print_sum(5,7);
}
void print_sum( int x , int y){
    int s;
    s = x + y;
    cout<<x<<"+"<<y<<" = "<<s<<endl;
}
```

Write a function that reads two numbers and another function to prints their sum

```
void read_number();
void print_sum(int,int);
int main(){
    read_number();
}

void read_number(){
    int A , B;
    cout<<"Enter two numbers please ";
    cin>>A>>B;
    print_sum(A , B);
}

void print_sum(int x,int y){
    int s;
    s = x + y;
    cout<<x<<"+"<<y<<" = "<<s<<endl;
}
```

Write a function that prints one of three messages randomly

```
void print_message();  
int main() {  
    for(int i = 1 ; i <= 10 ; i++)  
        print_message();  
}  
  
void print_message() {  
    int i = rand()%3 + 1;  
    if(i == 1)    cout<<"Hello\n";  
    if(i == 2)    cout<<"Hi\n";  
    if(i == 3)    cout<<"Welcome\n";  
}
```


Scope Rules

- Scope
 - Portion of program where identifier can be used
- Scopes for an identifier are
 - Function scope
 - File scope
 - Function prototype scope
 - Block scope
 - Class scope
 - namespace scope

Scope Rules

- Function scope
 - Can only be referenced inside function in which they appear, can not be referenced outside function body
- File scope
 - Defined outside a function, known in all functions
 - Global variables, function definitions and prototypes all have file scope
- Function-prototype scope
 - Names in function prototype are optional, only type required
 - Compiler ignores the name if used in parameter list of function-prototype
 - In a single prototype, name can be used once
 - Identifier used in function prototype can be reused elsewhere in the program

Scope Rules

- Block scope
 - Identifiers declared inside the block
 - Begins at declaration, ends at right brace } of the block
 - Can only be referenced in this range
 - Local variables, function parameters
 - In case of nested blocks and identifiers in inner and outer block have same name
 - Identifier in the outer block is hidden until the inner block terminates

Unary Scope Resolution Operator

- Unary scope resolution operator (`::`)
 - Access global variable if local variable of same name is in scope
 - Not needed if names are different
 - Use `::variable`
$$y = ::x + 3;$$
 - Can not be used to access a local variable of same name in an outer block
 - Good to avoid using same names for local and global variables

Example Program

```
int x = 10;
int f1(int);
int main(){
    cout << x << endl;
    int x = 15;
    cout << x << endl;
    cout << ::x << endl;
    if (true){ int x = 5; cout << x << endl; }
    cout << x << endl;
    cout << f1(x) << endl;
    cout << ::x << endl;
}

int f1(int a){
    cout << x << endl;
    x = x - a;
    int x = 13;
    cout << x << endl;
    return x + a;
}
```

10

15

10

5

15

10

13

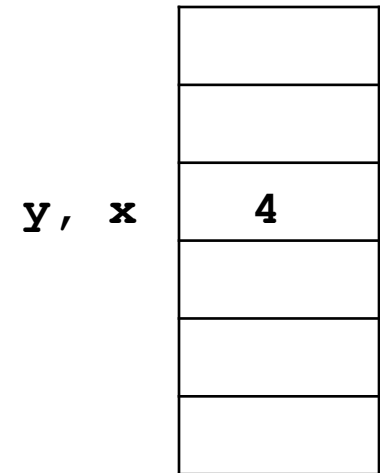
28

-5

Reference Variables (Alias)

- A reference variable is an alias (another name) for an already existing variable
- Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.
- Example

```
int x = 4;  
int &y = x;
```



- **y** is an alias for variable **x**
- **x** and **y** both refer to same variable (same memory location)

Example

```
int main() {  
  
    int x;  
    int &y = x; //y is an alias for x  
  
    x = 5;  
    cout<<"x = "<< x <<"\t"<<"y = "<< y << endl;  
  
    y = 7;  
    cout<<"x = "<< x <<"\t"<<"y = "<< y << endl;  
}
```

x = 5	y = 5
x = 7	y = 7

Call By Value and Call By Reference

- When passing a parameter by value, the function receives a ***copy*** of the variable, so it can't modify the variable

```
void function(int var) ;
```

- When passing a parameter by reference, the function receives a ***reference*** to the variable (**not a copy of it**), so it can modify the variable

```
void function(int &var) ;
```


Example

```
void square1(int) ;
void square2(int &) ;
int main() {
    int x,y;
    cout<<"Enter a number:";
    cin>>x;
    square1(x) ;
    cout<< "calling x by value, x = " << x <<endl;
    square2(x) ;
    cout<< "calling x by ref, x = " << x <<endl;
}
void square1(int a) { // call by value, int a = x
    a = a * a; }

void square2(int &a){//call by reference, int &a = x
    a = a * a; }
```

```
Enter a number: 10
calling x by value, x = 10
calling x by ref, x = 100
```

Example

```
void read(int &, int);  
int main() {  
    int x = 0, y = 0;  
    read(x, y);  
    cout << "x = " << x << endl;  
    cout << "y = " << y << endl;  
}  
void read(int &a, int b) {  
    cout << "Enter two numbers:\n";  
    cin >> a >> b;  
}
```

Enter two numbers:

6

4

x = 6

y = 0

Static Variables

- Local variables in function declared with keyword **static**
- Keeps value between function calls
- Only known in own function
- Static local variables retain their values when function is exited

Example

```
void f( );  
int main() {  
    f();  
    f();  
    f();  
}  
void f() {  
    static int a = 1;  
    int b = 1;  
    cout <<"a = "<< a++ <<"\tb = "<< b++ <<endl;  
}
```

a = 1	b = 1
a = 2	b = 1
a = 3	b = 1

Default arguments

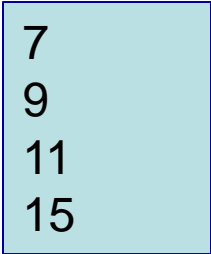
- default argument provide a value to be used instead of omitted parameters in function call
- Default value is automatically used by the compiler and passed in to the function
- If not enough parameters, rightmost go to their defaults
- Default argument should be specified in the first occurrence of the function name, typically prototype

Example

```
int f(int x=1 , int y=2 , int z=4 );
```

```
int main() {  
    cout << f() <<endl;  
    cout << f(3) <<endl;  
    cout << f(3,4) <<endl;  
    cout << f(6,8,1) <<endl;  
}
```

```
int f(int x, int y , int z) {  
    return x + y + z;  
}
```



7
9
11
15

Recursive functions

- *A recursive function* is a function that makes a call to itself
- May result in a infinite recursion
- Typical problems solved using recursion
 - Factorial
 - Fibonacci series
 - Sum between X, Y

Recursive Factorial

```
int fact(int) ;

int main() {
    int x;
    cout << "Enter a number:";
    cin >> x;
    cout << x << "! = " << fact(x) << endl;
}

int fact(int n) {
    if (n == 1)
        return 1;
    else
        return n*fact(n-1) ;
}
```

Enter a number:4
4! = 24

Fibonacci Series

- 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , ...
- $fib(0) = 0$
- $fib(1) = 1$
- $fib(n) = fib(n - 1) + fib(n - 2)$

Fibonacci Series

```
int fib(int) ;
int main() {
    int x;
    cout << "Enter a number:";
    cin >> x;
    cout << "Fibonacci ("<<x<<") = "<< fib(x) << endl;
}

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

Enter a number:7
Fibonacci(7) = 13

Sum of numbers from 1 to n

```
int sum(int) ;
void main() {
    int x;
    cout<<"Enter end number: ";
    cin>>x;
    cout<<"Sum of the series = "<<sum(x)<<endl;
}

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

Enter end number: 3
Sum of the series = 6

Sum of numbers from X to Y

```
int sum(int , int) ;
int main() {
    int x, y;
    cout<<"Enter first number:";
    cin>> x;
    cout<<"Enter second number:";
    cin>> y;
    cout<<"Sum of the series = "<<sum(x,y)<<endl;
}
```

```
int sum (int a ,int b) {
    if (a == b)
        return b;
    else
        return b + sum(a , b-1);
}
```

```
Enter first number:5
Enter second number:7
Sum of the series = 18
```

Function Overloading

- C++ enables several functions of the same name to be defined
 - as long as they have different signatures
- The C++ compiler selects the proper function to call by examining the number, types and order of the arguments in the call
- Overloaded functions are normally used to perform similar operations that involve different program logic on different data types

Example

```
int square(int x) {  
    cout << "square of integer " << x << " is ";  
    return x * x;  
}  
  
double square(double y) {  
    cout << "square of double " << y << " is ";  
    return y * y;  
}  
  
int main() {  
    cout << square(5) << endl;  
    cout << square(5.0) << endl;  
}
```

square of integer 5 is 25
square of double 5 is 25

Inline Functions

- Keyword **inline** before function return type
 - Asks the compiler to copy code into program instead of making function call
 - Reduce function-call overhead
 - Compiler can ignore **inline**
 - Good for small, often-used functions
- Example

```
inline double cube(double s) { return s*s*s; }
```

Example

```
inline double cube(double s) { return s* s* s }  
int main() {  
  
    double side;  
    cout<<"Enter the side length of your cube: ";  
    cin >> side;  
    cout << "Volume of cube = "<<cube(side)<<endl;  
}
```


04 - Arrays

Array

- Set of adjacent memory locations of the same data type.
- All of them have one name, which is the array name.
- each location has subscript number starts at zero.
- One Dimensional array

```
Data_Type array_name[size]
```

```
int a[10]
```

- Two Dimensional array

```
Data_Type array_name[Rows][Columns]
```

```
int a[3][4]
```

One Dimensional array Example

```
int List[5];  
List[0] = 100;  
    List[1] = 30;  
    List[2] = 10;  
    List[3] = -20;  
    List[4] = 5;  
  
for (i=0; i < 5; i++)  
    cout<<List[i]<<"\t";
```

100	30	10	-20	5
-----	----	----	-----	---

List[0]	100
List[1]	30
List[2]	10
List[3]	-20
List[4]	5

Definitions and initial values

```
int X[4]={4,2,3,1};
```

```
int A[10]={0};
```

```
int B[100]={1};
```

```
int C[7]={1,2,6};
```

```
int d[5];
```

```
int E[ ]={4,6,2};
```

Not accepted Statements

```
int A[4]={4,2,4,5,2};
```

```
int E[ ];
```

Read and print an Array

```
int main() {  
    int A[3] = {10,20,15};  
    cout<<"Print Array:";  
    for (int i=0 ; i<3 ; i++)  
        cout << A[i] << '\\t';  
    cout<<endl;  
  
    for (int i=0 ; i<3 ; i++) {  
        cout << "Enter A["<<i<<"]:";  
        cin >> A[i];  
    }  
    cout<<"Print Array:";  
    for (int i=0 ; i<3 ; i++)  
        cout << A[i] << '\\t';  
}
```

Print Array:10	20	15
Enter A[0]:3		
Enter A[1]:7		
Enter A[2]:1		
Print Array:3	7	1

Write a program that inputs an array elements then prints the count of elements greater than 10

```
const int S = 10;
int main() {
    int A[S];
    for(int i=0 ; i<S ; i++) {
        cout << "Enter A[" << i << "]:";
        cin >> A[i];
    }
    int count = 0;
    for (int i = 0; i < S; i++) {
        if (A[i] > 10)
            count++;
    }
    cout<< "No of element > 10 = "<<count <<endl;
}
```

Multiplying elements in even positions

```
const int S = 5;
int main() {
    int A[S];
    for(int i=0; i < S; i++) {
        cout<<"Enter A["<<i<<"]:";
        cin>>A[i];
    }

    long even = 1;
    for (int i = 0; i < S; i++)
        if(i % 2 == 0)
            even *= A[i];

    cout<<"Multiplying elements in Even positions = "<<even;
}
```

```
Enter A[0]:2
Enter A[1]:6
Enter A[2]:1
Enter A[3]:11
Enter A[4]:3
Multiplying elements in Even positions = 6
```

Find the Maximum element

```
const int Size = 5;
int main() {

    int V[Size];
    cout<<"Enter 5 numbers to find the maximum\n";

    for(int i = 0; i < Size; i++)
        cin >> V[i];

    int Max = V[0];
    for(int i = 1; i < Size; i++)
        if (Max < V[i])
            Max = V[i];
    cout<<"Max = "<< Max <<endl;
}
```

```
Enter 5 numbers to find the maximum
14
11
23
7
19
Max = 23
```


Find the Maximum element

```
const int Size = 5;
int main() {
    int V[Size];
    cout<<"Enter 5 numbers to find the maximum\n";
    for(int i = 0; i < Size; i++)
        cin >> V[i];

    int Max = V[0];
    int pos = 0;
    for(int i = 1; i < Size; i++)
        if (Max < V[i]) {
            Max = V[i];
            pos = i;
        }
    cout<<"Max= " <<Max<<" at position " <<pos<<endl;
}
```

Enter 5 numbers to find the maximum

14

11

23

7

19

Max = 23 at position 2

Array Search using Linear Search

```
int main() {  
    int V[5]={7,12,5,31,4}, Element, i;  
  
    cout << "Enter the element to search for:";  
    cin >> Element;  
  
    bool Found = false;  
    for(i = 0; i < 5; i++)  
        if(Element == V[i]){  
            Found = true;  
            break;  
        }  
    if (Found)  
        cout<<Element<<" Found at position "<<i<<endl;  
    else  
        cout<<Element<<" is not in the array \n";  
}
```

Enter the element to search for:18
18 is not in the array

Enter the element to search for:5
5 Found at position 2

Swap Function

```
void swap(int &,int &);  
int main() {  
    int a = 2, b = 9;  
    cout<<"a = "<<a<<' \t'<<"b = "<<b<<endl;  
    swap(a,b);  
    cout<<"\nAfter Swap\n";  
    cout<<"a = "<<a<<' \t'<<"b = "<<b<<endl;  
}  
void swap(int &x , int &y) {  
    int temp = x;  
    x = y;  
    y = temp;  
}
```

a = 2 b = 9

After Swap

a = 9 b = 2

Array Sort with Bubble Sort

```
void swap(int &, int &);  
const int Size = 5;  
int main() {  
    int V[Size] = {5,9,7,4,3};  
  
    for(int i = 1 ; i < Size; i++)  
        for(int j = 0 ; j < Size-i; j++)  
            if(V[j] > V[j+1])  
                swap(V[j], V[j+1]);  
  
    cout<<"Array after Sort:\n";  
    for(int i = 0; i < Size; i++)    cout<< V[i]<<'\\t';  
}  
  
void swap(int &x , int &y) {  
    int temp = x;  
    x = y;  
    y = temp;  
}
```

Array after Sort:

3	4	5	7	9
---	---	---	---	---

5	9	7	4	3
---	---	---	---	---

- $i = 1$:

- $j = 0$

5	9	7	4	3
---	---	---	---	---

- $j = 1$

5	7	9	4	3
---	---	---	---	---

- $j = 2$

5	7	4	9	3
---	---	---	---	---

- $j = 3$

5	7	4	3	9
---	---	---	---	---

- $i = 2$:

- $j = 0$

5	7	4	3	9
---	---	---	---	---

- $j = 1$

5	4	7	3	9
---	---	---	---	---

- $j = 2$

5	4	3	7	9
---	---	---	---	---

- $i = 3$:

- $j = 0$

4	5	3	7	9
---	---	---	---	---

- $j = 1$

4	3	5	7	9
---	---	---	---	---

- $i = 4$:

- $j = 0$

3	4	5	7	9
---	---	---	---	---

```
if (V[j] > V[j+1])
    swap( V[j], V[j+1] );
```

5	4	3	2	1
---	---	---	---	---

• $i = 1$:

– $j = 0$	4	5	3	2	1
– $j = 1$	4	3	5	2	1
– $j = 2$	4	3	2	5	1
– $j = 3$	4	3	2	1	5

• $i = 2$:

– $j = 0$	3	4	2	1	5
– $j = 1$	3	2	4	1	5
– $j = 2$	3	2	1	4	5

• $i = 3$:

– $j = 0$	2	3	1	4	5
– $j = 1$	2	1	3	4	5

• $i = 4$:

– $j = 0$	1	2	3	4	5
-----------	---	---	---	---	---

```
if (V[j] > V[j+1])
    swap( V[j], V[j+1] );
```

Passing Arrays to functions

- When passing an array to a function, it is passed by reference.
- No need to declare add the `&` operator

Sum of Array elements

```
int Sum(int [],int) ;
int main( ) {

    int A[5]={2,3,4,5,6} ;
    int B[4]={5,3,1,7} ;
    cout<<"The sum of A is "<<Sum(A,5)<<endl ;
    cout<<"The sum of B is "<<Sum(B,4)<<endl ;
}
```



```
int Sum(int x[],int size){
    int S = 0 ;
    for(int i = 0 ; i < size ; i++)
        S = S + x[i] ;
    return S ;
}
```

The sum of A is 20
The sum of B is 16

Read and print Arrays

```
void read(int [], int) ;
void print(int[], int) ;
int main() {
    int A[5] , B[4] ;
    read(A, 5) ;
    read(B, 4) ;
    print(A, 5) ;
    print(B, 4) ;
}

void read(int x[], int size) {
    for(int i = 0 ; i < size ; i++) {
        cout<<"Enter a number please:";
        cin >> x[i];
    }
}

void print(int x[], int size) {
    for(int i = 0 ; i < size ; i++)
        cout << x[i]<<" ";
}
```

Sum of two Arrays

```
void sum (int [], int [], int[], int) ;
void print(int [], int) ;
void main() {
    int a[4]={4,3,2,1};
    int b[4]={2,2,4,4};
    int c[4];
    sum(a,b,c,4) ;
    print(c,4) ;
}

void sum(int x[], int y[], int z[], int size){
    for (int i = 0 ; i < size ; i++)
        z[i] = x[i]+y[i];
}

void print(int x[] , int size){
    for(int i = 0 ; i < size ; i++)
        cout << x[i]<<" ";
}
```

Two Dimensional array

- Declare a 3 by 4 array

```
int Matrix[3][4];
```

- Assign values to array Elements

```
Matrix[0][1]= 30;
```

```
Matrix[1][0]= 100;
```

```
Matrix[1][2]= 10;
```

```
Matrix[2][3]= -20;
```

```
Matrix[2][0]= Matrix[1][2];
```

	0	1	2	3
0		100		
1	30		10	
2	10			-20

- Print Elements

```
for( int i=0 ; i < 3 ; i++){  
    for( int j = 0 ; j < 4 ; j++)  
        cout << Matrix[i][j] << "\t";  
    cout << endl;  
}
```

2D Array Initialization

```
int A[2][3] = {1,2,3,10,20};
```

	0	1	2
0	1	2	3
1	10	20	0

```
int M[3][4] = {{4,30},{100,5,10},{1,7,2,-20}};
```

	0	1	2	3
0	4	30	0	0
1	100	5	10	0
2	1	7	2	-20

Example 1

```
int main() {  
    int A[2][3]={ {1,2}, {4,5,6} }, B[2][3]={1,2,4,5,6};  
    for(int i=0; i<2; i++){  
        for(int j=0; j<3; j++){  
            cout<<A[i][j]<<'\\t';  
            cout<<endl;  
        }  
        cout<<"\\n\\n";  
        for(int i=0; i<2; i++){  
            for(int j=0; j<3; j++){  
                cout<<B[i][j]<<'\\t';  
                cout<<endl;  
            }  
        }  
    }
```

1	2	0
4	5	6
1	2	4
5	6	0

Example 2

```
const int R = 2, C = 3;
```

```
int main() {
```

```
    int A[R][C];
```

```
    for (int i=0; i < R ; i++)
```

```
        for (int j=0; j < C ; j++) {
```

```
            cout<<"Enter A["<<i<<"] ["<<j<<"] : ";
```

```
            cin>>A[i][j];
```

```
        }
```

```
    for (int i=0; i < R; i++) {
```

```
        for(int j=0; j < C ; j++)
```

```
            cout<<A[i][j]<<'\\t';
```

```
        cout<<endl;
```

```
    }
```

```
}
```

Enter A[0][0]:1

Enter A[0][1]:6

Enter A[0][2]:7

Enter A[1][0]:3

Enter A[1][1]:5

Enter A[1][2]:2

1	6	7
---	---	---

3	5	2
---	---	---

Sum of 2D Array elements

```
int sum(int [][][2], int);  
int main(){  
    int x[3][2]={6,5,4,3,2,1};  
    int y[4][2]={5,4,3,2,3,4,1};  
    cout<<"The sum of array x is " <<sum(x,3)<<endl;  
    cout<<"The sum of array y is " <<sum(y,4)<<endl;  
}  
int sum(int a[][2],int r){  
    int s = 0;  
    for(int i = 0 ;i < r ; i++)  
        for(int j = 0 ; j < 2 ; j++)  
            s += a[i][j];  
    return s;  
}
```

The sum of array x is 21
The sum of array y is 22

Diagonal Sum

```
int Dsum(int [][][3]);

int main() {

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

    cout<<"Diagonal Sum = "<<Dsum(A)<<endl;
}

int Dsum(int a[][3]) {
    int S=0;
    for (int i=0 ; i<s ; i++)
        S += a[i][i];

    return S;
}
```

Diagonal Sum = 15

Inverse Diagonal Summation

```
const int R = 3, C = 3;
void main() {

    int A[R][C]= {{1,2,3}, {6,11,7}, {7,3,9}};
    int DSum = 0;

    for (int i=0 ; i < R ; i++)
        for (int j=0; j < C ; j++)
            if ( i+j == 2)
                DSum += A[i][j];

    cout<<"Inverse Diagonal Sum = "<<DSum<<endl;
}
```

Inverse Diagonal Sum = 21

Lower Triangular Multiplication

```
const int R = 3, C = 3;
```

```
int main() {
```

```
    int A[R][C] = { {1,2,3},  
                    {4,5,6},  
                    {7,8,9} };
```

```
    long m = 1;
```

```
    for (int i=0; i < R; i++)
```

```
        for (int j=0; j<i; j++)
```

```
            m *= A[i][j];
```

```
    cout<<"Lower Triangular Multiplication =  
    "<<m<<endl;
```

```
}
```

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

Lower Triangular Multiplication = 224

Lower Triangular Multiplication

```
const int R = 4, C = 4;
int main() {
    int A[R][C] = {
        {6, 2, 3, 4},
        {4, 2, 3, 4},
        {1, 2, 3, 4},
        {1, 2, 3, 4}};
```

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

```
long m = 1;
for (int i=0; i < R; i++)
    for (int j=0; j < C; j++)
        if (i > j)
            m *= A[i][j];
cout<<"Lower Triangular Mul. = "<<m<<endl;
}
```

Lower Triangular Mul. = 48

Copy one array to another

```
const int R = 4, C= 3;
```

```
int main() {
```

```
    int A[R][C] = {{1,1,1}, {2,2,2}, {3,3,3}, {4,4,4}};
```

```
    int B[R][C];
```

```
    for(int i = 0 ; i < R ; i++)
```

```
        for(int j = 0 ; j < C ; j++)
```

```
            B[i][j] = A[i][j];
```

```
    for(int i=0; i < R; i++){
```

```
        for(int j = 0 ; j < C ; j++)
```

```
            cout<< B[i][j] <<'\\t';
```

```
        cout<<endl;
```

```
    }
```

```
}
```

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

Store the following symbols in an array

```
const int R = 4, C = 4;
```

```
int main() {  
    char Symbol[R][C];  
    for (int i=0 ; i < R ; i++)  
        for (int j=0 ; j < C ; j++)  
            if (i == j)  
                Symbol[i][j] = '*';  
            else  
                if (i > j)  
                    Symbol[i][j] = '#';  
                else  
                    Symbol[i][j] = '$';  
}
```

*	\$	\$	\$
#	*	\$	\$
#	#	*	\$
#	#	#	*

Matrix Summation

```
const int R = 3, C = 4;
int main() {
    int A[R][C] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
    int B[R][C] = {{4,5,6,7}, {3,6,7,8}, {9,1,4,2}};
    int C[R][C];

    for(int i = 0 ; i < R ; i++)
        for(int j = 0; j < C ; j++)
            C[i][j] = A[i][j] + B[i][j];

    for(int i = 0 ; i < R ; i++){
        for(int j = 0 ; j < C ; j++)
            cout<<C[i][j]<<'\\t';
        cout<<endl;
    }
}
```

5	7	9	11
8	12	14	16
18	11	15	14

Compute the average for 2 marks for 3 students

```
int main() {  
    int marks[3][2];  
    for(int i = 0 ; i < 3 ; i++){  
        for(int j = 0 ; j < 2 ; j++){  
            cout <<"Enter mark "<<j+1<<" for student "<<i+1<<" : ";  
            cin>>marks[i][j];  
        }  
        cout<<endl;  
    }  
  
    float avg[3];  
    for(int i = 0 ; i < 3; i++){  
        float sum = 0;  
        for(int j = 0 ; j < 2 ; j++){  
            sum = sum + marks[i][j];  
        }  
        avg[i] = sum/2;  
    }  
  
    for(int i = 0 ; i < 3 ; i++)  
        cout <<"Average of Student "<<i+1<<" = "<<avg[i]<<endl;  
}
```

Enter mark 1 for student 1 : 5
Enter mark 2 for student 1 : 7

Enter mark 1 for student 2 : 3
Enter mark 2 for student 2 : 4

Enter mark 1 for student 3 : 6
Enter mark 2 for student 3 : 11

Average of Student 1 = 6
Average of Student 2 = 3.5
Average of Student 3 = 8.5

Transpose

```
int main() {
    int A[3][4]={ {1,2,3,4} , {5,6,7,8} , {9,10,11,12} };
    cout<<"Original Array:\n";
    for(int i = 0 ; i < 3 ; i++){
        for(int j = 0 ; j < 4 ; j++){
            cout<<A[i][j]<<"\t";
            cout<<endl;
        }
    }
    int B[4][3];
    for(int i = 0 ; i < 3 ; i++){
        for(int j = 0 ; j < 4 ; j++){
            B[j][i]=A[i][j];
        }
    }
    cout<<"\nTrnspose:\n";
    for(int i = 0 ; i < 4 ; i++){
        for(int j = 0 ; j < 3 ; j++){
            cout<<B[i][j]<<"\t";
            cout<<endl;
        }
    }
}
```

Original Array:

1	2	3	4
5	6	7	8
9	10	11	12

Transpose:

1	5	9
2	6	10
3	7	11
4	8	12

5 - Pointers

Pointer Variables

- Variables contain a specific value (direct reference)
- Pointers variables contain the address of a variable that has a specific value (indirect reference)

```
int *ptr
```

- * indicates that variable `ptr` is a pointer
- the pointer `ptr` point to an `int` variable
- Multiple pointers require multiple asterisks:

```
int *ptr1, *ptr2;
```

- Can declare pointers to any data type
- Pointers can be initialized to 0, NULL, or an address
 - 0 or NULL points to nothing

Pointer Operators

```
int y = 5;
int *yPtr;
yPtr = &y; // yPtr gets address of y
```

- **&** (address operator)
 - returns the address of its operand
 - **yPtr** points to **y**: address of **y** is value of **yPtr**
`cout << &y; //prints 600`
- ***** (indirection/dereferencing operator)
 - returns the value of what its operand points to
 - ***yPtr** returns the value of **y** (because **yPtr** points to **y**).
 - ***** can be used to assign a value to a location in memory
`*yPtr = 7; // changes y to 7`
- ***** and **&** are inverses they cancel each other out

Address	value	Name
600	5	y
500	600	yPtr

Example 1

```
int main() {  
    int a = 7, *aPtr = &a;  
    cout << "address of a is " << &a << endl;  
    cout << "value of aPtr is " << aPtr << endl;  
  
    cout << "value of a is " << a << endl;  
    cout << "value of *aPtr is " << *aPtr << endl;  
  
    cout << "* and & are inverses of each other\n";  
    cout << "&*aPtr = " << &*aPtr << endl;  
    cout << "&*aPtr = " << &*aPtr << endl;  
}
```

```
address of a is 0xedfe0c  
value of aPtr is 0xedfe0c  
value of a is 7  
value of *aPtr is 7  
* and & are inverses of each other  
&*aPtr = 0xedfe0c  
&*aPtr = 0xedfe0c
```

Example 2

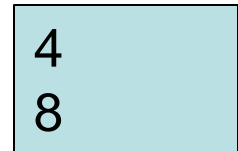
```
int main( ) {  
    int a = 10, b = 20, *ptr_a, *ptr_b;  
    ptr_a = &a;  
    ptr_b = &b;  
    cout<<*ptr_a<<"\t"<<*ptr_b<<endl;  
    a += 10;  
    b += 20;  
    cout<<*ptr_a<<"\t"<<*ptr_b<<endl;  
    ++*ptr_a;  
    ++*ptr_b;  
    cout<<a<<"\t"<<b<<endl;  
    ptr_a = ptr_b;  
    cout<<*ptr_a<<"\t"<<*ptr_b<<endl;  
}
```

10	20
20	40
21	41
41	41

Pass by Reference with Pointers

- There are three ways in C++ to pass arguments to a functions:
 - Pass by value
 - Pass by reference with reference arguments
 - Pass by reference with pointer arguments.
- Pointers can be used to accomplish pass by reference, for example:

```
void f(int *n) {    *n = 2 * *n;    }  
  
int main() {  
    int x = 2 , y = 4 , *p = &y;  
    f(&x) ;  
    f(p) ;  
    cout<<x<<endl<<y;  
}
```



- A pointer to an `int` or an address of an `int` variable can be passed to function `f`

Pass by reference with pointers

```
void increment(int *);  
int main() {  
    int x = 2 , *p = &x;  
    cout<<"original value of x = "<<x<<endl;  
    cube(&x);  
    cout<<"new value of x = "<< x <<endl;  
    cube(p);  
    cout<<"new value of x = "<< x <<endl;  
}  
void increment(int *n) {  
    *n = *n + 1;  
}
```

original value of x = 2
new value of x = 3
new value of x = 4

Pointer Expressions and Pointer Arithmetic

- Pointer arithmetic
 - Increment/decrement pointer (`++` or `--`)
 - Add/subtract an integer to/from a pointer(`+` or `+=` , `-` or `-=`)
- Pointer arithmetic is meaningless unless performed on an array

Pointer Expressions and Pointer Arithmetic

- Subtracting pointers

- Returns the number of elements between two addresses

```
vPtr2 = &v[2];
```

```
vPtr = &v[0];
```

```
vPtr2 - vPtr will return 2
```

- Pointer comparison

- Test which pointer points to the higher numbered array element
- Test if a pointer points to 0 (**NULL**)

```
if (vPtr == 0)
```

```
    statement
```

Pointer Expressions and Pointer Arithmetic

- Pointers assignment
 - If not the same type, a cast operator must be used
 - Exception: pointer to `void` (type `void *`)
 - Generic pointer, represents any type
 - No casting needed to convert a pointer to `void` pointer
 - `void` pointers cannot be dereferenced

Pointers and Arrays

- Array name is a constant pointer, it always points to the beginning of the array.
- Example: `int b[5] , *bPtr = b; //or bPtr = &b[0]`
 - `bPtr` points to the first element of array `b` (`bPtr` equals address of first element of `b`)
- Accessing array elements with pointers
 - Element `b[n]` can be accessed by `*(bPtr+n)`
 - Called pointer/offset notation
 - Array itself can use pointer arithmetic.
 - `b[3]` same as `*(b+3)`
 - Pointers can be subscripted (pointer/subscript notation)
 - `bPtr[3]` same as `b[3]`

Arrays and Pointers

```
int main( ) {
    int a[5] = {2,5,7,4,3}, *p = a;
    cout<<"printing array in different ways:\n";
    for(int i=0;i<5;i++)  cout<<*(a+i)<<"\t";
    cout<<endl;
    for(int i=0;i<5;i++)  cout<<*(p+i)<<"\t";
    cout<<endl;
    for(int i=0;i<5;i++)  cout<<p[i]<<"\t";
    cout<<endl;
    for(int i=0;i<5;i++)  cout<<*p++<<"\t";
}
```

printing array elements in different ways:

2	5	7	4	3
2	5	7	4	3
2	5	7	4	3
2	5	7	4	3

Array name is a static pointer

```
float V[3]={3.2 , 2.1 , 6.7};

float *ptr = V; //ptr=&V[0];

cout<<"Value of ptr: "<< ptr <<endl;
cout<<"Value of V: "<< V <<endl;
cout<<"Address of V[0]: "<< &V[0] <<endl;

for (int i=0 ; i<3 ; i++){
    cout<< ptr <<" : "<< *ptr <<endl;
    ptr++;
}
```

```
Value of ptr: 0xedfdf0
Value of V: 0xedfdf0
Address of V[0]: 0xedfdf0
0xedfdf0 : 3.2
0xedfdf4 : 2.2
0xedfdf8 : 6.2
```

Address	Value	Name
0xedfdd8	0xedfdf0	ptr
	.	
	.	
	.	
0xedfdf0	3.2	V[0]
0xedfdf4	2.1	V[1]
0xedfdf8	6.7	V[2]

Check for identical arrays using pointers

```
int main( ){
    int a[5]={1, 3, 5, 7, 9};
    int b[5]={1, 3, 5, 8, 9};
    bool flag = true;
    for (int i = 0; i < 5; i++)
        if (*(a+i) != *(b+i)){ //if(a[i] != b[i])
            flag = false;
            break;
        }
    if (flag)
        cout<<"arrays are identical";
    else
        cout<<"arrays are not identical";
}
```

arrays are not identical

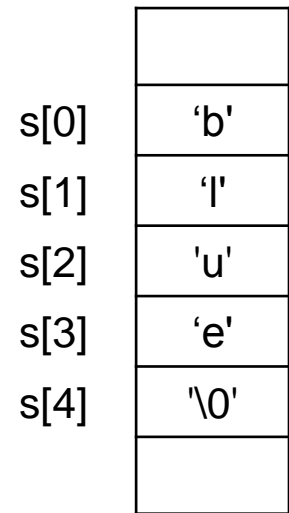
6 - Strings

Fundamentals of Characters and Strings

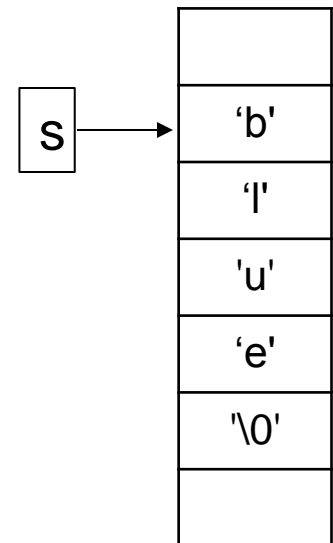
- String is a series of characters treated as one unit. String may include letters, digits, special characters
- A string literal (string constant) is enclosed in double quotes, for example: `"I like C++"`
- String always end with the null character `'\0'`

Fundamentals of Characters and Strings

- `char s[] = { 'b', 'l', 'u', 'e' };` or `char s[] = "blue";`
 - Creates a 5 element character array (named s)
 - last element is the null character `'\0'`



- `char *s = "blue";`
 - Creates a pointer named (s) that points to the first element of **constant string** "blue", somewhere in memory
 - **s** is a pointer to a constant character
 - last element is the null character `'\0'`



Character array

```
int main() {  
    char str[] = "Ahmad Ali";  
    cout<< str << endl;  
  
    for(int i=0; str[i] != '\0'; i++)  
        cout << str[i];  
  
}
```

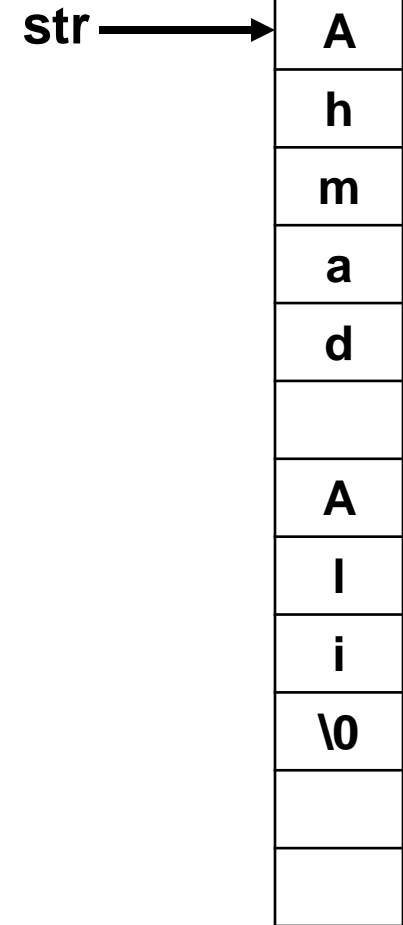
Ahmad Ali
Ahmad Ali

str[0]	'A'
str[1]	'h'
str[2]	'm'
str[3]	'a'
str[4]	'd'
str[5]	' '
str[6]	'A'
str[7]	'l'
str[8]	'i'
str[9]	'\0'

Pointer to a character

```
int main() {  
    char *str = "Ahmad Ali";  
    cout<< str << endl;  
  
    for(int i=0; *str != '\\0'; i++)  
        cout << *(str++);  
}
```

Ahmad Ali
Ahmad Ali



Reading strings

- `cin >> char_array;`
 - Assign input to the character array
 - Reads characters until whitespace is entered
 - String could exceed array size

```
char word[20];  
cin >> word;
```
- `cin.getline(char_array, size, delimiter_char);`
 - Reads characters to the specified array until either:
 - One less than the size is reached
 - The delimiter character is input

```
char sentence[80];  
cin.getline( sentence, 80, '\n' )
```

Reading strings using `cin>>`

```
int main() {  
    char a[25], *s = a;  
    cout << "Enter a name : ";  
    cin>> s; //or cin>> a; read until space or enter  
    cout << a << endl;  
    cout<< s <<endl;  
    for(int i=0 ; a[i] != '\0' ; i++)  
        cout << a[i] << endl;  
}
```

Enter a name : ali
ali
ali
a
l
i

Enter a name : sam ahmad
sam
sam
s
a
m

Read strings using `getline()`

```
int main() {

    char a[20], *s = a;

    cout<<"Please enter your name: ";

    cin.getline(s,20,'\n');//or use a instead of s

    cout << s << endl;

    for(int i=0; a[i] != '\0'; i++)
        cout << a[i];

}
```

A	a[0]
h	a[1]
m	a[2]
a	a[3]
d	a[4]
	a[5]
A	a[6]
l	a[7]
i	a[8]
\0	a[9]

```
Please enter your name: Ahmad Ali
Ahmad Ali
Ahmad Ali
```

Array of `char` and Array of `int`

```
int main() {  
  
    char a[10] = "Ahmad"; //char a[10]={'A','h','m','a','d'};  
  
    cout << a << endl;    // prints Ahmed  
  
    int b[10] = {1,2,3,4,5};  
  
    cout << b << endl;  
    //prints the address of the first element in the array  
}
```

Function to return the length of a string

```
int length(char []);  
  
int main() {  
    char s1[] = "hello world";  
    char *s2 = "welcome to C++";  
    cout<<"the length of s1 = "<<length(s1);  
    cout<<endl;  
    cout<<"the length of s2 = "<<length(s2);  
}  
  
int length(char name[]) {  
    int i;  
    for( i=0 ; name[i] != 0 ; i++);  
    // do nothing, just make i= the last element in the array  
    return i;  
}
```

the length of s1 = 11
the length of s2 = 14

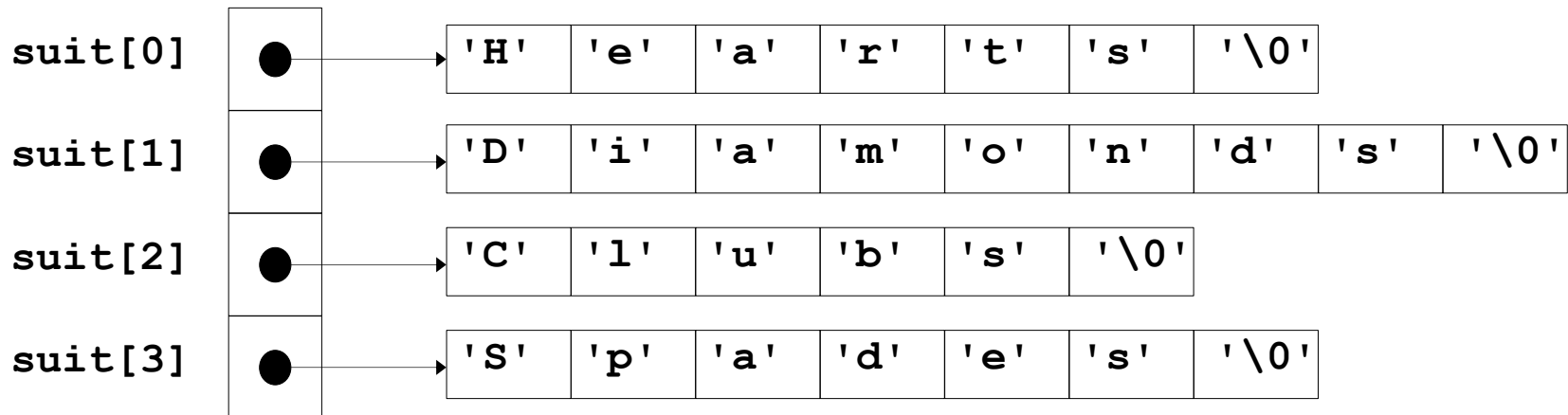
Program to print a string in reverse order

```
int main() {  
    int i , j;  
    char name[25];  
    cout<<"Enter your first name: ";  
    cin>>name;  
  
    for(int i=0 ; name[i] != '\0' ; i++);  
    //do nothing, just make i = the last element in the array  
    cout<<"First name in reverse: ";  
    for(int j = i-1 ; j >= 0 ; j--)  
        cout<< name[j];  
}
```

Enter your first name: ahmad
First name in reverse: damha

Arrays of Pointers

- Arrays can contain pointers
- Commonly used to store an array of strings
`char *suit[4]={"Hearts", "Diamonds", "Clubs", "Spades"};`
- Each element of suit array is a pointer to a `char` constant
- Strings are not stored in array, only the pointers are stored in array
- `suit` array has a fixed size, but strings can be of any size



String Manipulation Functions

- The string-handling library `<cstring>`, provides many useful functions for manipulating string data:
 - Copying strings
 - Comparing strings
 - Concatenating strings
 - Searching strings for characters and other strings
 - Tokenizing strings: (separating strings into logical pieces such as the separate words in a sentence)
 - Determining the length of strings
- ASCII character code
 - Strings are compared using their character codes
- Tokenizing
 - Breaking strings into tokens, separated by user-specified characters
 - Tokens are usually logical units, such as words (separated by spaces)
 - **"This is my string"** has 4 word tokens (separated by spaces)

Copying Strings

- `char *strcpy(char *s1, const char *s2)`
 - Copies the string **s2** into the character array **s1**. The value of **s1** is returned
- `char *strncpy(char *s1, const char *s2, size_t n)`
 - Copies at most **n** characters of the string **s2** into the character array **s1**. The value of **s1** is returned
 - **Size_t** is an unsigned integral data type

Example (strcpy, strncpy)

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

int main( ) {
    char x[ ] = "Happy Birthday";
    char y[25];
    char z[] = "Today is sunday";

    cout << strcpy(y,x) << endl;
    cout << strncpy(z,x,10) ;

}
```

```
Happy Birthday
Happy Birtunday
```

Comparing Strings

- `int strcmp(const char *s1, const char *s2)`
 - Compares string **s1** with string **s2**, and returns a value of:
 - 0: if **s1** is equal to **s2**
 - -1: if **s1** is less than **s2**
 - 1: if **s1** is greater than **s2**
- `int strncmp(const char *s1, const char *s2, size_t n)`
 - Compares up to **n** characters of string **s1** with string **s2**, and returns:
 - 0: if **s1** is equal to **s2**
 - -1: if **s1** is less than **s2**
 - 1: if **s1** is greater than **s2**

Example (strcmp, strncmp)

```
int main( ) {  
    char s1[]="Happy Year", s2[]="Happy Year", s3[]="Happy";  
  
    cout << "s1 vs s2 : " << strcmp(s1,s2) << endl;  
    cout << "s1 vs s3 : " << strcmp(s1,s3) << endl;  
    cout << "s3 vs s1 : " << strcmp(s3,s1) << endl;  
    cout << endl;  
    cout << "5 chars of s1 vs s3 : " << strncmp(s1,s3,5) << endl;  
    cout << "7 chars of s1 vs s3 : " << strncmp(s1,s3,7) << endl;  
    cout << "7 chars of s3 vs s1 : " << strncmp(s3,s1,7) << endl;  
}
```

```
s1 vs s2 : 0  
s1 vs s3 : 1  
s3 vs s1 : -1  
  
5 chars of s1 vs s3 : 0  
7 chars of s1 vs s3 : 1  
7 chars of s3 vs s1 : -1
```

Concatenating Strings

- `char *strcat(char *s1, const char *s2)`
 - Appends string **s2** to string **s1**
 - The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.
- `char *strncat(char *s1, const char *s2, size_t n)`
 - Appends at most **n** characters of string **s2** to string **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.

Example (strcat, strncat)

```
#include <iostream>
#include <cstring>
using namespace std;
int main( ) {
    char s1[20]="Happy New", s2[ ]="Year", s3[40] = "";

    cout << strcat(s1,s2) <<endl;
    cout << strncat(s3, s1, 5)<<endl;
    cout << strcat(s3, s1)<<endl;
}
```

```
Happy NewYear
Happy
HappyHappy NewYear
```

Tokenizing Strings

- `char *strtok(char *s1, const char *s2)`
 - String `s1` is broken up based on the characters contained in string `s2`
 - `char s1[] = "this:is:a:string", s1` has 4 tokens based on character `:` which are: `"this"` , `"is"` , `"a"` , `"string"`.
 - `strtok` returns only one token at a time
 - `strtok(s1, ":")` returns `"this"`
 - A sequence of calls to `strtok` must be made to break string `s1` into tokens
 - A pointer to the current token is returned by each call.

Tokenizing Strings

- The first call must contains **s1** as the first argument, and subsequent calls to continue tokenizing the same string must contain **NULL** as the first argument
- When called, **strtok** checks whether the first argument is:
 - **NULL**: return the next token in the previous string that was passed to **strtok**
 - not **NULL**: return the first token in the current string passed to **strtok**
- If there are no more tokens when the function is called, **NULL** is returned

Example (strtok)

```
int main( ) {  
    char s[]="This is a String",*p;  
    int i = 0;  
    p = strtok(s," "); //p points to the first token in s  
  
    while(p != NULL) {  
  
        cout<<"Token # "<<++i<<" : "<<p<<endl; //print current token  
  
        p = strtok(NULL ," "); //p points to next token in s  
    }  
    cout<<"string s has "<<i<<" tokens."  
}
```

```
Token # 1 : This  
Token # 2 : is  
Token # 3 : a  
Token # 4 : String  
string s has 4 tokens.
```

Example 2 (strtok)

```
int main( ) {  
    char s[]="Wafaa.Salem.Ahmad.Jameel", *p[50];  
    int i = 0;  
  
    p[0] = strtok(s, "."); //p[0] points to the first token in s  
  
    while(p[i++] != NULL)  
        p[i] = strtok(NULL, "."); //p[i] points to next token in s  
  
    for(i = 0 ; p[i] != NULL ; i++) //print tokens  
        cout<<p[i]<<endl;  
}
```

Wafaa Salem Ahmad Jameel

Function that return how many times a word is found in a sentence

```
int search(char *sentence, char *word) {  
    char *p;  
    int c = 0;  
    p = strtok(sentence, " ");  
    while (p != NULL) {  
        if (strcmp(p, word) == 0)  
            c++;  
        p = strtok(NULL, " ");  
    }  
    return c;  
}
```

String Size

- `size_t strlen(const char *s)`
 - Determines the length of string `s`. The number of characters preceding the terminating null character is returned
- `size_t`: this type is defined in the header file `<cstring>` to be an unsigned integral type such as `unsigned int` or `unsigned long`.

Example (strlen)

```
int main( ) {  
  
    char s1 []="Happy Year", s2 []="Happy";  
  
    cout <<"s1 length : "<< strlen(s1)<<endl;  
  
    cout <<"s2 length : "<< strlen(s2)<<endl;  
}
```

```
s1 length : 10  
s2 length : 5
```