C++ programming from Zero to Hero

Presented by Youssef M. Khalil

COURSE OUTLINES

- Defining variables
- comments
- Input / output
- If statement
- Switch..case
- While loop
- Do while loop
- For loop
- Goto
- Functions
- Arrays
- pointers

COURSE INSTRUCTIONS

- create an account on HackerRank
- Watch the videos all attend all the lectures and take notes
- Solve the Homework and the examples
- try to google or YouTube and the concepts by yourself again to get on the concepts deeply

WHY TO LEARN C++

C++ is a MUST for students and working professionals to become a great Software Engineer. I will list down some of the key advantages of learning C++:

- C++ is very close to hardware, so you get a chance to work at a low level which gives you lot of control in terms of memory management, better performance and finally a robust software development.
- **C++ programming** gives you a clear understanding about Object Oriented Programming. You will understand low level implementation of polymorphism when you will implement virtual tables and virtual table pointers, or dynamic type identification.
- C++ is one of the every green programming languages and loved by millions of software developers. If you are a great C++ programmer then you will never sit without work and more importantly you will get highly paid for your work.
- C++ is the most widely used programming languages in application and system programming. So you can choose your area of interest of software development.
- C++ really teaches you the difference between compiler, linker and loader, different data types, storage classes, variable types their scopes etc.

APPLICATIONS OF C++ PROGRAMMING

- **Application Software Development** - C++ programming has been used in developing almost all the major Operating Systems like Windows, Mac OSX and Linux. Apart from the operating systems, the core part of many browsers like Mozilla Firefox and Chrome have been written using C++. C++ also has been used in developing the most popular database system called MySQL.
- **Programming Languages Development** - C++ has been used extensively in developing new programming languages like C#, Java, JavaScript, Perl, UNIX's C Shell, PHP and Python, and Verilog etc.
- **Computation Programming** - C++ is the best friends of scientists because of fast speed and computational efficiencies.
- **Games Development** - C++ is extremely fast which allows programmers to do procedural programming for CPU intensive functions and provides greater control over hardware, because of which it has been widely used in development of gaming engines.
- **Embedded System** - C++ is being heavily used in developing Medical and Engineering Applications like softwares for MRI machines, high-end CAD/CAM systems etc.

FOR COMPILER AND IDE INSTALLATION

https://www.youtube.com/watch?v=VIkmXi9lRiU
https://www.youtube.com/watch?v=ZixCyiMVFqc
https://www.youtube.com/watch?v=MNJ7aYHCMCo

THE STRUCTURE OF C++ PROGRAM

```cpp
// my first program in C++
#include <iostream>

int main()
{
  std::cout << "Hello World!";
}
```

```
Hello World!
```

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

// main() is where program execution begins.
int main() {
   cout << "Hello World"; // prints Hello World
   return 0;
}
```

C++ IDENTIFIERS

A C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).
C++ does not allow punctuation characters such as @, $, and % within identifiers. C++ is a case-sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in C++.

Here are some examples of acceptable identifiers −

```
mohd       zara    abc    move_name   a_123
myname50   _temp   j      a23b9       retVal
```

KEYWORDS IN C++

alignas	default	if	reinterpret_cast	try
alignof	delete	inline	return	typedef
asm	do	int	short	typeid
auto	double	log	signed	typename
bool	dynamic_cast	long	sizeof	union
break	else	mutable	static	unsigned
case	enum	namespace	static_assert	using
catch	explicit	new	static_cast	virtual
char	export	noexcept	struct	void
class	extern	nullptr	switch	volatile
const	false	operator	template	wchar_t
const_cast	float	private	this	while
constexpr	for	protected	thread_local	
continue	friend	public	throw	
decltype	goto	register	true	

COMMENTS IN C++

Program comments are explanatory statements that you can include in the C++ code. These comments help anyone reading the source code. All programming languages allow for some form of comments.
C++ supports single-line and multi-line comments. All characters available inside any comment are ignored by C++ compiler.
C++ comments start with /* and end with */. For example −

```
/* This is a comment */

/* C++ comments can also
   * span multiple lines
*/
```

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

main() {
   cout << "Hello World"; // prints Hello World

   return 0;
}
```

PRIMITIVE BUILT-IN TYPES IN C++

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	8bytes	-2,147,483,648 to 2,147,483,647
signed long int	8bytes	same as long int
unsigned long int	8bytes	0 to 4,294,967,295
long long int	8bytes	-(2^63) to (2^63)-1
unsigned long long int	8bytes	0 to 18,446,744,073,709,551,615
float	4bytes	
double	8bytes	
long double	12bytes	
wchar_t	2 or 4 bytes	1 wide character

SIZE OF PRIMITIVE BUILT-IN TYPES IN C++

Input

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

int main() {
   cout << "Size of char : " << sizeof(char) << endl;
   cout << "Size of int : " << sizeof(int) << endl;
   cout << "Size of short int : " << sizeof(short int) << endl;
   cout << "Size of long int : " << sizeof(long int) << endl;
   cout << "Size of float : " << sizeof(float) << endl;
   cout << "Size of double : " << sizeof(double) << endl;
   cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;

   return 0;
}
```

output

```
Size of char : 1
Size of int : 4
Size of short int : 2
Size of long int : 4
Size of float : 4
Size of double : 8
Size of wchar_t : 4
```

SIZE OF PRIMITIVE BUILT-IN TYPES IN C++

Input

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

int main() {
   cout << "Size of char : " << sizeof(char) << endl;
   cout << "Size of int : " << sizeof(int) << endl;
   cout << "Size of short int : " << sizeof(short int) << endl;
   cout << "Size of long int : " << sizeof(long int) << endl;
   cout << "Size of float : " << sizeof(float) << endl;
   cout << "Size of double : " << sizeof(double) << endl;
   cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;

   return 0;
}
```

output

```
Size of char : 1
Size of int : 4
Size of short int : 2
Size of long int : 4
Size of float : 4
Size of double : 8
Size of wchar_t : 4
```

DEFINING A VARIABLE

Assignment Statements

In an assignment statement, first the expression on the right-hand side of the equal sign is evaluated, and then the variable on the left-hand side of the equal sign is set equal to this value.

SYNTAX

```
Variable = Expression;
```

EXAMPLES

```
distance = rate * time;
count = count + 2;
```

DEFINING A VARIABLE

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

// Variable declaration:
extern int a, b;
extern int c;
extern float f;

int main () {
   // Variable definition:
   int a, b;
   int c;
   float f;

   // actual initialization
   a = 10;
   b = 20;
   c = a + b;

   cout << c << endl ;

   f = 70.0/3.0;
   cout << f << endl ;

   return 0;
}
```

DEFINING A VARIABLE

```
// initialization of variables

#include <iostream>
using namespace std;

int main ()
{
  int a=5;               // initial value: 5
  int b(3);              // initial value: 3
  int c{2};              // initial value: 2
  int result;            // initial value undetermined

  a = a + b;
  result = a - c;
  cout << result;

  return 0;
}
```

```
6
```

DEFINING A VARIABLE

PROGRAMMING TIP Use Meaningful Names

Variable names and other names in a program should at least hint at the meaning or use of the thing they are naming. It is much easier to understand a program if the variables have meaningful names. Contrast the following:

```
x = y * z;
```

with the more suggestive:

```
distance = speed * time;
```

The two statements accomplish the same thing, but the second is easier to understand.

DEFINING A VARIABLE

SELF-TEST EXERCISES

1. Give the declaration for two variables called `feet` and `inches`. Both variables are of type *`int`* and both are to be initialized to zero in the declaration. Use both initialization alternatives.

2. Give the declaration for two variables called `count` and `distance`. `count` is of type *`int`* and is initialized to zero. `distance` is of type *`double`* and is initialized to `1.5`.

3. Give a C++ statement that will change the value of the variable `sum` to the sum of the values in the variables `n1` and `n2`. The variables are all of type *`int`*.

TYPE DEDUCTION: AUTO AND DECLTYPE

When a new variable is initialized, the compiler can figure out what the type of the variable is Automatically by the initializer. For this, it suffices to use auto as the type specifier for the variable

```
#include <iostream>

using namespace std;

int main()
{
    auto num =10.5;
    cout<<num;

    return 0;
}
```

```
10.5


...Program finished with exit code 0
Press ENTER to exit console.
```

STRINGS

```
1  // my first string
2  #include <iostream>
3  #include <string>
4  using namespace std;
5
6  int main ()
7  {
8    string mystring;
9    mystring = "This is a string";
10   cout << mystring;
11   return 0;
12 }
```

BASIC INPUT/OUTPUT

cin Statements

A cin statement sets variables equal to values typed in at the keyboard.

SYNTAX

```
cin >> Variable_1 >> Variable_2 >> ... ;
```

EXAMPLE

```
cin >> number >> size;
cin >> time_to_go
    >> points_needed;
```

BASIC INPUT/OUTPUT

```
1  // i/o example
2
3  #include <iostream>
4  using namespace std;
5
6  int main ()
7  {
8    int i;
9    cout << "Please enter an integer value: ";
10   cin >> i;
11   cout << "The value you entered is " << i;
12   cout << " and its double is " << i*2 << ".\n";
13   return 0;
14 }
```

SCAPE CHARACTERS

Escape code	Description
\n	newline
\r	carriage return
\t	tab
\v	vertical tab
\b	backspace
\f	form feed (page feed)
\a	alert (beep)
\'	single quote (')
\"	double quote (")
\?	question mark (?)
\\	backslash (\)

CONSTANT

```
1  #include <iostream>
2  using namespace std;
3
4  const double pi = 3.14159;
5  const char newline = '\n';
6
7  int main ()
8  {
9    double r=5.0;                  // radius
10   double circle;
11
12   circle = 2 * pi * r;
13   cout << circle;
14   cout << newline;
15 }
```

OPERATORS

operator	description
+	addition
-	subtraction
*	multiplication
/	division
%	modulo

&& OPERATOR (and)		
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

|| OPERATOR (or)		
a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

operator	description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

expression	equivalent to...
`y += x;`	`y = y + x;`
`x -= 5;`	`x = x - 5;`
`x /= y;`	`x = x / y;`
`price *= units + 1;`	`price = price * (units+1);`

Example 1	Example 2

`x = 3;`
`y = ++x;`
`// x contains 4, y contains 4`
	`x = 3;`
	`y = x++;`
	`// x contains 4, y contains 3`

PRECEDENCE OF OPERATORS

Level	Precedence group	Operator	Description	Grouping
1	Scope	`::`	scope qualifier	Left-to-right
2	Postfix (unary)	`++ --`	postfix increment / decrement	Left-to-right
		`()`	functional forms	
		`[]`	subscript	
		`. ->`	member access	
3	Prefix (unary)	`++ --`	prefix increment / decrement	Right-to-left
		`~ !`	bitwise NOT / logical NOT	
		`+ -`	unary prefix	
		`& *`	reference / dereference	
		`new delete`	allocation / deallocation	
		`sizeof`	parameter pack	
		`(type)`	C-style type-casting	
4	Pointer-to-member	`.* ->*`	access pointer	Left-to-right
5	Arithmetic: scaling	`* / %`	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	`+ -`	addition, subtraction	Left-to-right
7	Bitwise shift	`<< >>`	shift left, shift right	Left-to-right
8	Relational	`< > <= >=`	comparison operators	Left-to-right
9	Equality	`== !=`	equality / inequality	Left-to-right
10	And	`&`	bitwise AND	Left-to-right
11	Exclusive or	`^`	bitwise XOR	Left-to-right
12	Inclusive or	`|`	bitwise OR	Left-to-right
13	Conjunction	`&&`	logical AND	Left-to-right
14	Disjunction	`||`	logical OR	Left-to-right
15	Assignment-level expressions	`= *= /= %= += -= >>= <<= &= ^= |=`	assignment / compound assignment	Right-to-left
		`?:`	conditional operator	
16	Sequencing	`,`	comma separator	Left-to-right

Highest precedence (done first)

Lowest precedence (done last)

IF STATEMENT

A Single Statement for Each Alternative:

```
1     if (Boolean_Expression)
2         Yes_Statement
3     else
4         No_Statement
```

A Sequence of Statements for Each Alternative:

```
5     if (Boolean_Expression)
6     {
7         Yes_Statement_1
8         Yes_Statement_2
9           ...
10        Yes_Statement_Last
10    }
12    else
13    {
14        No_Statement_1
15        No_Statement_2
16          ...
17        No_Statement_Last
18    }
```

IF STATEMENT

```
1     #include <iostream>
2     using namespace std;
3     int main( )
4     {
5         int hours;
6         double gross_pay, rate;
7         cout << "Enter the hourly rate of pay: $";
8         cin >> rate;
9         cout << "Enter the number of hours worked,\n"
10             << "rounded to a whole number of hours: ";
11        cin >> hours;
12        if (hours > 40)
13           gross_pay = rate * 40 + 1.5 * rate * (hours - 40);
14        else
15           gross_pay = rate * hours;

16        cout.setf(ios::fixed);
17        cout.setf(ios::showpoint);
18        cout.precision(2);
19        cout << "Hours = " << hours << endl;
20        cout << "Hourly pay rate = $" << rate << endl;
21        cout << "Gross pay = $" << gross_pay << endl;
22        return 0;
23    }
```

SHORT HAND IF...ELSE (TERNARY OPERATOR)

Syntax

```
variable = (condition) ? expressionTrue : expressionFalse;
```

Example

```
int time = 20;
string result = (time < 18) ? "Good day." : "Good evening.";
cout << result;
```

MULTIWAY BRANCHES

```
1   if (count > 0)
2      if (score > 5)
3         cout << "count > 0 and score > 5\n";
4      else
5         cout << "count > 0 and score <= 5\n";
```

MULTIWAY IF-ELSE STATEMENT

Multiway *if-else* Statement

SYNTAX

```
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
        .
        .
        .
else if (Boolean_Expression_n)
    Statement_n
else
    Statement_For_All_Other_Possibilities
```

MULTIWAY IF-ELSE STATEMENT

EXAMPLE>

```
if ((temperature <-10) && (day == SUNDAY))
    cout << "Stay home.";
else if (temperature <-10) //and day != SUNDAY
    cout << "Stay home, but call work.";
else if (temperature <= 0) //and temperature >= -10
    cout << "Dress warm.";
else //temperature > 0
    cout << "Work hard and play hard.";
```

The Boolean expressions are checked in order until the first *true* Boolean expression is encountered, and then the corresponding statement is executed. If none of the Boolean expressions is *true*, then the *Statement_For_All_Other_Possibilities* is executed.

USING = IN PLACE OF ==

```
if (x = 12)
    Do_Something
else
    Do_Something_Else
```

Suppose you wanted to test to see if the value of x is equal to 12 so that you really meant to use == rather than =. You might think the compiler will catch your mistake. The expression

```
x = 12
```

is not something that is satisfied or not. It is an assignment statement, so surely the compiler will give an error message. Unfortunately, that is not the case. In C++ the expression `x = 12` is an expression that returns (or has) a value, just like `x + 12` or `2 + 3`. An assignment expression's value is the value transferred to the variable on the left. For example, the value of `x = 12` is 12. We saw in our discussion of Boolean value compatibility that *int* values may be converted to *true* or *false*. Since 12 is not zero, it is converted to *true*. If you use `x = 12` as the Boolean expression in an *if* statement, the Boolean expression is always *true*, so the first branch (*Do_Something*) is always executed.

This error is very hard to find because it *looks correct*! The compiler can find the error without any special instructions if you put the 12 on the left side of the comparison, as in

COMPARISON OPERATORS

Math Symbol	English	C++ Notation	C++ Sample	Math Equivalent
=	equal to	==	x + 7 == 2 * y	x + 7 = 2y
≠	not equal to	!=	ans != 'n'	ans ≠ 'n'
<	less than	<	count < m + 3	count < m + 3
≤	less than or equal to	<=	time <= limit	time ≤ limit
>	greater than	>	time > limit	time > limit
≥	greater than or equal to	>=	age >= 21	age ≥ 21

COMPARISON OPERATORS

The "and" Operator &&

You can form a more elaborate Boolean expression by combining two simple tests using the "and" operator &&.

SYNTAX (FOR A BOOLEAN EXPRESSION USING &&)

```
(Comparison_1) && (Comparison_2)
```

EXAMPLE (WITHIN AN *if-else* STATEMENT)

```
if  ( (score > 0) && (score < 10) )
    cout << "score is between 0 and 10\n";
else
    cout << "score is not between 0 and 10.\n";
```

If the value of score is greater than 0 and the value of score is also less than 10, then the first cout statement will be executed; otherwise, the second cout statement will be executed.

COMPARISON OPERATORS

The "or" Operator ||

You can form a more elaborate Boolean expression by combining two simple tests using the "or" operator ||.

SYNTAX (FOR A BOOLEAN EXPRESSION USING ||)

```
(Comparison_1) || (Comparison_2)
```

EXAMPLE (WITHIN AN *if-else* STATEMENT)

```
if ( (x == 1) || (x == y) )
    cout << "x is 1 or x equals y.\n";
else
    cout << "x is neither 1 nor equal to y.\n";
```

If the value of x is equal to 1 or the value of x is equal to the value of y (or both), then the first cout statement will be executed; otherwise, the second cout statement will be executed.

COMPARISON OPERATORS

SELF-TEST EXERCISES

1. Determine the value, *true* or *false*, of each of the following Boolean expressions, assuming that the value of the variable `count` is 0 and the value of the variable `limit` is 10. Give your answer as one of the values *true* or *false*.

a. `(count == 0) && (limit < 20)`
b. `count == 0 && limit < 20`
c. `(limit > 20) || (count < 5)`
d. `!(count == 12)`
e. `(count == 1) && (x < y)`
f. `(count < 10) || (x < y)`
g. `!( ((count < 10) || (x < y)) && (count >= 0) )`
h. `((limit/count) > 7) || (limit < 20)`
i. `(limit < 20) || ((limit/count) > 7)`
j. `((limit/count) > 7) && (limit < 0)`
k. `(limit < 0) && ((limit/count) > 7)`
l. `(5 && 7) + (!6)`

HOMEWORK

- https://www.hackerrank.com/challenges/c-tutorial-basic-data-types/problem?isFullScreen=true
- https://www.hackerrank.com/challenges/cpp-hello-world/problem?isFullScreen=true
- https://www.hackerrank.com/challenges/cpp-input-and-output/problem?isFullScreen=true
- https://www.codezclub.com/cpp-program-addition-two-numbers/
- https://www.codezclub.com/cpp-calculate-sum-average-three-numbers/
- https://www.codezclub.com/cpp-find-size-int-float-double-char/
- https://www.codezclub.com/cpp-check-number-odd-even/
- https://www.codezclub.com/cpp-convert-days-years-weeks-days/
- https://www.codezclub.com/cpp-program-find-largest-number/

C++ ENUMERATION

An **enumeration type** is a type whose values are defined by a list of constants of type *int*. An enumeration type is very much like a list of declared constants.

When defining an enumeration type, you can use any *int* values and can have any number of constants defined in an enumeration type. For example, the following enumeration type defines a constant for the length of each month:

```
enum MonthLength { JAN_LENGTH = 31, FEB_LENGTH = 28,
MAR_LENGTH = 31, APR_LENGTH = 30, MAY_LENGTH = 31,
JUN_LENGTH = 30, JUL_LENGTH = 31, AUG_LENGTH = 31,
SEP_LENGTH = 30, OCT_LENGTH = 31, NOV_LENGTH = 30,
DEC_LENGTH = 31 };
```

As this example shows, two or more named constants in an enumeration type can receive the same *int* value.

If you do not specify any numeric values, the identifiers in an enumeration-type definition are assigned consecutive values beginning with 0. For example, the type definition

```
enum Direction { NORTH = 0, SOUTH = 1, EAST = 2, WEST = 3 };
```

is equivalent to

```
enum Direction { NORTH, SOUTH, EAST, WEST };
```

The form that does not explicitly list the *int* values is normally used when you just want a list of names and do not care about what values they have.

If you initialize only some enumeration constant to some values, say

```
enum MyEnum { ONE = 17, TWO, THREE, FOUR = -3, FIVE };
```

C++ ENUMERATION

Example 1: Enumeration Type

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

enum week { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

int main()
{
    week today;
    today = Wednesday;
    cout << "Day " << today+1;
    return 0;
}
```

Output

```
Day 4
```

C++ ENUMERATION

Example2: Changing Default Value of Enums

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

enum seasons { spring = 34, summer = 4, autumn = 9, winter = 32};

int main() {

    seasons s;

    s = summer;
    cout << "Summer = " << s << endl;

    return 0;
}
```

Output

```
Summer = 4
```

USING = IN PLACE OF ==

```
if (x = 12)
    Do_Something
else
    Do_Something_Else
```

Suppose you wanted to test to see if the value of x is equal to 12 so that you really meant to use == rather than =. You might think the compiler will catch your mistake. The expression

```
x = 12
```

is not something that is satisfied or not. It is an assignment statement, so surely the compiler will give an error message. Unfortunately, that is not the case. In C++ the expression `x = 12` is an expression that returns (or has) a value, just like `x + 12` or `2 + 3`. An assignment expression's value is the value transferred to the variable on the left. For example, the value of `x = 12` is 12. We saw in our discussion of Boolean value compatibility that *int* values may be converted to *true* or *false*. Since 12 is not zero, it is converted to *true*. If you use `x = 12` as the Boolean expression in an *if* statement, the Boolean expression is always *true*, so the first branch (*Do_Something*) is always executed.

This error is very hard to find because it *looks correct*! The compiler can find the error without any special instructions if you put the 12 on the left side of the comparison, as in

IF STATEMENT

SELF-TEST EXERCISES

21. Write an *if-else* statement that outputs the word `High` if the value of the variable `score` is greater than 100 and `Low` if the value of `score` is at most 100. The variable `score` is of type *int*.

22. Suppose `savings` and `expenses` are variables of type *double* that have been given values. Write an *if-else* statement that outputs the word `Solvent`, decreases the value of `savings` by the value of `expenses`, and sets the value of `expenses` to 0, provided that `savings` is at least as large as `expenses`. If, however, `savings` is less than `expenses`, the *if-else* statement simply outputs the word `Bankrupt` and does not change the value of any variables.

23. Write an *if-else* statement that outputs the word `Passed` provided the value of the variable `exam` is greater than or equal to 60 and the value of the variable `programs_done` is greater than or equal to 10. Otherwise, the *if-else* statement outputs the word `Failed`. The variables `exam` and `programs_done` are both of type *int*.

C++ SWITCH STATEMENTS

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

C++ SWITCH STATEMENTS

Example

```
int day = 4;
switch (day) {
  case 1:
    cout << "Monday";
    break;
  case 2:
    cout << "Tuesday";
    break;
  case 3:
    cout << "Wednesday";
    break;
  case 4:
    cout << "Thursday";
    break;
  case 5:
    cout << "Friday";
    break;
  case 6:
    cout << "Saturday";
    break;
  case 7:
    cout << "Sunday";
    break;
}
// Outputs "Thursday" (day 4)
```

C++ SWITCH STATEMENTS

```
1   //Program to illustrate the switch statement.
2   #include <iostream>
3   using namespace std;
4   int main( )
5   {
6       char grade;
7       cout << "Enter your midterm grade and press Return: ";
8       cin >> grade;
9       switch (grade)
10      {
11          case 'A':
12              cout << "Excellent. "
13                   << "You need not take the final.\n";
14              break;
15          case 'B':
16              cout << "Very good. ";
17              grade = 'A';
18              cout << "Your midterm grade is now "
19                   << grade << endl;
20              break;
21          case 'C':
22              cout << "Passing.\n";
23              break;
24          case 'D':
25          case 'F':
26              cout << "Not good. "
27                   << "Go study.\n";
28              break;
29          default:
30              cout << "That is not a possible grade.\n";
31      }
32      cout << "End of program.\n";
33      return 0;
34  }
```

C++ SWITCH STATEMENTS

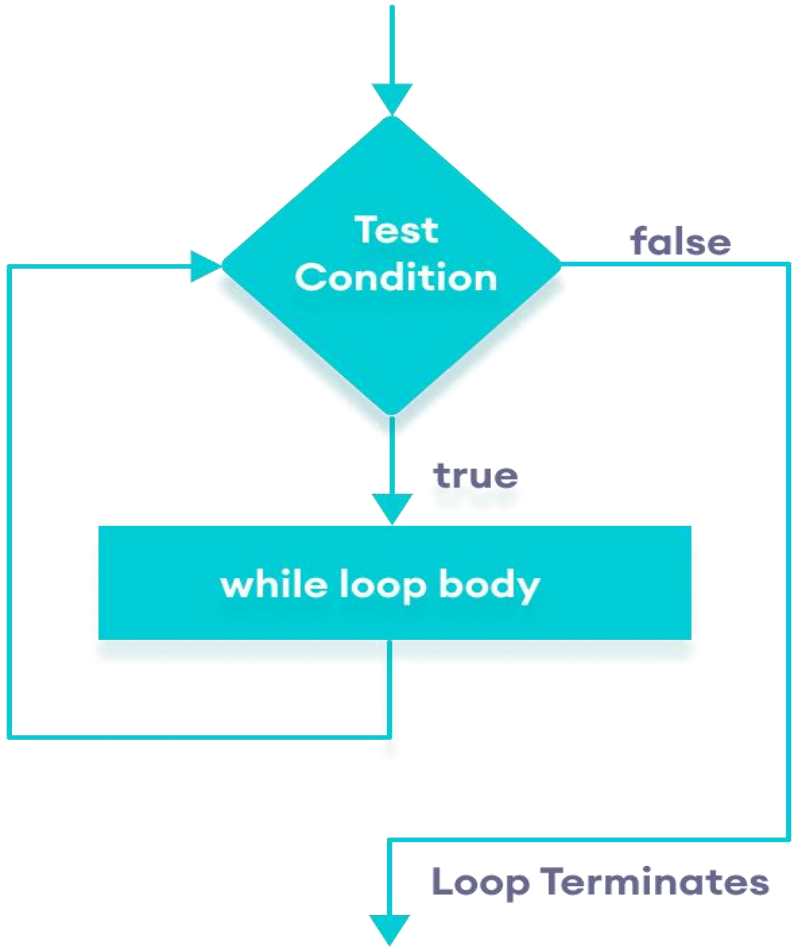
PITFALL Forgetting a *break* in a *switch* Statement

If you forget a *break* in a *switch* statement, the compiler will not issue an error message. You will have written a syntactically correct *switch* statement, but it will not do what you intended it to do. Consider the *switch* statement in the box entitled "*switch* Statement." If a *break* statement were omitted, as indicated by the arrow, then when the variable `vehicle_class` has the value `1`, the *case* labeled

```
case 1:
```

would be executed as desired, but then the computer would go on to also execute the next *case*. This would produce a puzzling output that says the vehicle is a passenger car and then later says it is a bus; moreover, the final value of `toll` would be `1.50`, not `0.50` as it should be. When the computer starts to execute a *case*, it does not stop until it encounters either a *break* or the end of the *switch* statement. ■

WHILE STATEMENT



WHILE STATEMENT

A Loop Body with Several Statements:

```
1    while (Boolean_Expression )
2    {
3        Statement_1
4        Statement_2
5          ...
6        Statement_Last
7    }
```

Do NOT put a semicolon here.

body

A Loop Body with a Single Statement:

```
8    while (Boolean_Expression )
9        Statement
```

body

WHILE STATEMENT

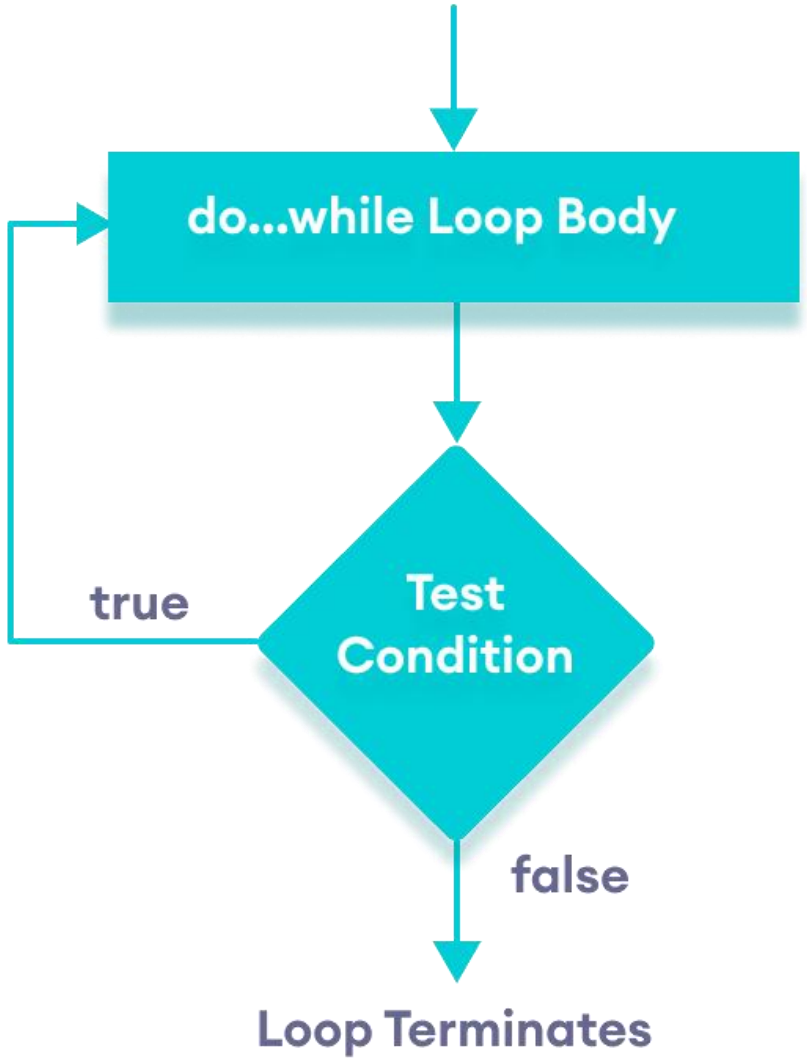
Example 1: Display Numbers from 1 to 5

```cpp
// C++ Program to print numbers from 1 to 5

#include <iostream>

using namespace std;

int main() {
    int i = 1;

    // while loop from 1 to 5
    while (i <= 5) {
        cout << i << " ";
        ++i;
    }

    return 0;
}
```

Output

```
1 2 3 4 5
```

DO...WHILE STATEMENT



DO...WHILE STATEMENT

A Loop Body with Several Statements:

```
1    do
2    {
3        Statement_1
4        Statement_2
5          ...
6        Statement_Last
7    } while (Boolean_Expression);
```

body

Do not forget the final semicolon.

A Loop Body with a Single Statement:

```
8    do
9        Statement
10   while (Boolean_Expression);
```

body

DO...WHILE STATEMENT

Example 3: Display Numbers from 1 to 5

```cpp
// C++ Program to print numbers from 1 to 5

#include <iostream>

using namespace std;

int main() {
    int i = 1;

    // do...while loop from 1 to 5
    do {
        cout << i << " ";
        ++i;
    }
    while (i <= 5);

    return 0;
}
```

Output

```
1 2 3 4 5
```

DO...WHILE STATEMENT

SELF-TEST EXERCISES

28. What is the output produced by the following (when embedded in a correct program with x declared to be of type *int*)?

```
x = 10;
while (x > 0)
{
    cout << x << endl;
    x = x - 3;
}
```

29. What output would be produced in the previous exercise if the > sign were replaced with < ?

30. What is the output produced by the following (when embedded in a correct program with x declared to be of type *int*)?

```
x = 10;
do
{
    cout << x << endl;
    x = x - 3;
} while (x > 0);
```

31. What is the output produced by the following (when embedded in a correct program with x declared to be of type *int*)?

```
x = -42;
do
{
    cout << x << endl;
    x = x - 3;
} while (x > 0);
```

DO...WHILE STATEMENT

SELF-TEST EXERCISES

28. What is the output produced by the following (when embedded in a correct program with x declared to be of type *int*)?

```
x = 10;
while (x > 0)
{
    cout << x << endl;
    x = x - 3;
}
```

29. What output would be produced in the previous exercise if the > sign were replaced with < ?

30. What is the output produced by the following (when embedded in a correct program with x declared to be of type *int*)?

```
x = 10;
do
{
    cout << x << endl;
    x = x - 3;
} while (x > 0);
```

31. What is the output produced by the following (when embedded in a correct program with x declared to be of type *int*)?

```
x = -42;
do
{
    cout << x << endl;
    x = x - 3;
} while (x > 0);
```

C++ FOR LOOP

DISPLAY 3.11 The *for* Statement *(part 1 of 2)*

for ***Statement***

SYNTAX

```
1    for (Initialization_Action; Boolean_Expression; Update_Action)
2         Body_Statement
```

EXAMPLE

```
1    for (number = 100; number >= 0; number--)
2         cout << number
3              << " bottles of beer on the shelf.\n";
```

Equivalent while Loop

EQUIVALENT SYNTAX

```
1    Initialization_Action;
2    while (Boolean_Expression)
3    {
4         Body_Statement
5         Update_Action;
6    }
```

EQUIVALENT EXAMPLE

```
1    number = 100;
2    while (number >= 0)
```

C++ FOR LOOP

```
1    //Illustrates a for loop.
2    #include <iostream>
3    using namespace std;
4
5    int main( )
6    {
7        int sum = 0;
8
9        for (int n = 1; n <= 10; n++) //Note that the variable n is a local
10           sum = sum + n;            //variable of the body of the for loop!
11
12       cout << "The sum of the numbers 1 to 10 is "
13            << sum << endl;
14       return 0;
15   }
```

Initializing action

Repeat the loop as long as this is true.

Done after each loop body iteration

Output

```
The sum of the numbers 1 to 10 is 55
```

C++ FOR LOOP

SYNTAX

```
for (Initialization_Action; Boolean_Expression; Update_Action)
{
    Statement_1
    Statement_2
            .
            .
            .
    Statement_Last
}
```

Body

EXAMPLE

```
for (int number = 100; number >= 0; number--)
{
    cout << number
         << " bottles of beer on the shelf.\n";
    if (number > 0)
        cout << "Take one down and pass it around.\n";
}
```

C++ FOR LOOP

PITFALL Extra Semicolon in a *for* Statement

Do not place a semicolon after the closing parentheses at the beginning of a *for* loop. To see what can happen, consider the following *for* loop:

```
for (int count = 1; count <= 10; count++);
    cout << "Hello\n";
```

Problem semicolon

If you did not notice the extra semicolon, you might expect this *for* loop to write `Hello` to the screen ten times. If you do notice the semicolon, you might expect the compiler to issue an error message. Neither of those things happens. If you embed this *for* loop in a complete program, the compiler will not complain. If you run the program, only one `Hello` will be output instead of ten `Hello`s. What is happening? To answer that question, we need a little background.

C++ FOR LOOP

PITFALL Extra Semicolon in a *for* Statement

Do not place a semicolon after the closing parentheses at the beginning of a *for* loop. To see what can happen, consider the following *for* loop:

```
for (int count = 1; count <= 10; count++);
    cout << "Hello\n";
```

Problem semicolon

If you did not notice the extra semicolon, you might expect this *for* loop to write `Hello` to the screen ten times. If you do notice the semicolon, you might expect the compiler to issue an error message. Neither of those things happens. If you embed this *for* loop in a complete program, the compiler will not complain. If you run the program, only one `Hello` will be output instead of ten `Hello`s. What is happening? To answer that question, we need a little background.

C++ FOR LOOP

Example 3: Find the sum of first n Natural Numbers

```cpp
// C++ program to find the sum of first n natural numbers
// positive integers such as 1,2,3,...n are known as natural numbers

#include <iostream>

using namespace std;

int main() {
    int num, sum;
    sum = 0;

    cout << "Enter a positive integer: ";
    cin >> num;

    for (int i = 1; i <= num; ++i) {
        sum += i;
    }

    cout << "Sum = " << sum << endl;

    return 0;
}
```

Output

```
Enter a positive integer: 10
Sum = 55
```

C++ FOR LOOP

SELF-TEST EXERCISES

25. What is the output of the following (when embedded in a complete program)?

```
for (int count = 1; count < 5; count++)
    cout << (2 * count) << " ";
```

26. What is the output of the following (when embedded in a complete program)?

```
for (int n = 10; n > 0; n = n - 2)
{
    cout << "Hello ";
    cout << n << endl;
}
```

27. What is the output of the following (when embedded in a complete program)?

```
for (double sample = 2; sample > 0; sample = sample - 0.5)
    cout << sample << " ";
```

LOOPS

For loop	While loop	Do while loop

for(initialization; condition; updating){
//statements;
}
	while(condition) {
	//statement(s);
	}
		do
		{
		//statements;
		}
		while(condition);

The control will never enter in a loop if the condition is not true for the first time.	The control will never enter in a loop if the condition is not true for the first time.	The control will enter a loop even if the condition is not true for the first time
No semicolon after the condition in the syntax.	No semicolon after the condition in the syntax.	There is semicolon after the condition in the syntax.
Initialization and updating is the part of the syntax.	Initialization and updating is not the part of the syntax.	Initialization and updating is not the part of the syntax

C++ BREAK AND CONTINUE

C++ Break

You have already seen the `break` statement used in an earlier chapter of this tutorial. It was used to "jump out" of a `switch` statement.

The `break` statement can also be used to jump out of a **loop**.

This example jumps out of the loop when `i` is equal to 4:

```
while (++count <= 10)
{
    cin >> number;

    if (number >= 0)
    {
        cout << "ERROR: positive number"
             << " or zero was entered as the\n"
             << count << "th number! Input ends "
             << "with the " << count << "th number.\n"
             << count << "th number was not added in.\n";
        break;
    }

    sum = sum + number;
}

cout << sum << " is the sum of the first "
     << (count - 1) << " numbers.\n";
```

C++ BREAK AND CONTINUE

C++ Continue

The `continue` statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

Example

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  cout << i << "\n";
}
```

C++ BREAK AND CONTINUE

C++ Continue

The `continue` statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

Example

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  cout << i << "\n";
}
```

TIME TO PRACTICE

A triangle can be of three types—one in which all the sides are equal, or one in which only two sides are equal, or one in which all the three sides are unequal. You are assigned to write a program that allows the user to enter the three sides of a triangle. The program should use three double variables to store the three sides of the triangle. The program should also be able to print the three angles of the triangle if the triangle is equilateral. If the triangle is isosceles, it should prompt the user to print the angle that is different and then print the remaining angles. Finally, if the triangle is scalene, the program should do nothing. Thus, there should be variables for holding the angles that are to be printed.

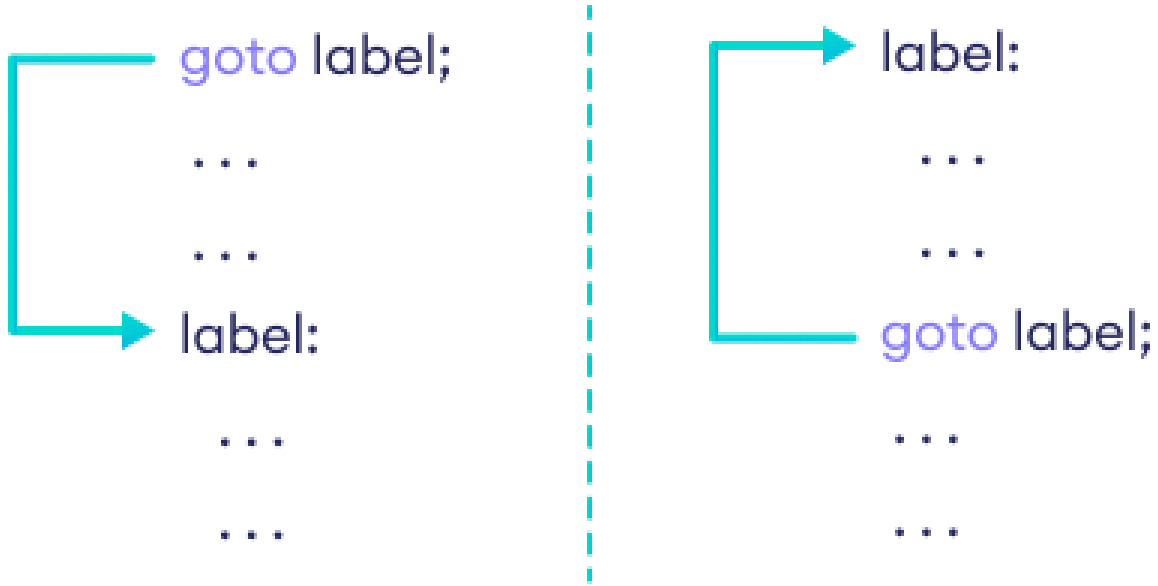
TIME TO PRACTICE

A primary school faces a challenge of teaching students the basic principles of mathematics. Most students fail to determine the position of digits in a number as ones, hundreds and so forth. But the curriculum requires a student to be able to determine the positions of digits in a number up to the thousands position. Moreover, the school does not have enough teachers to facilitate one to one tutoring for its students. You are assigned to write a program that will allow a student to enter a four-digit number and display the position of each digit at ones, tens, hundreds and thousands of the number. The program should alert the user if the number is out of the curriculum's range. Your program should let the user repeat the process until the user says she or he is done.

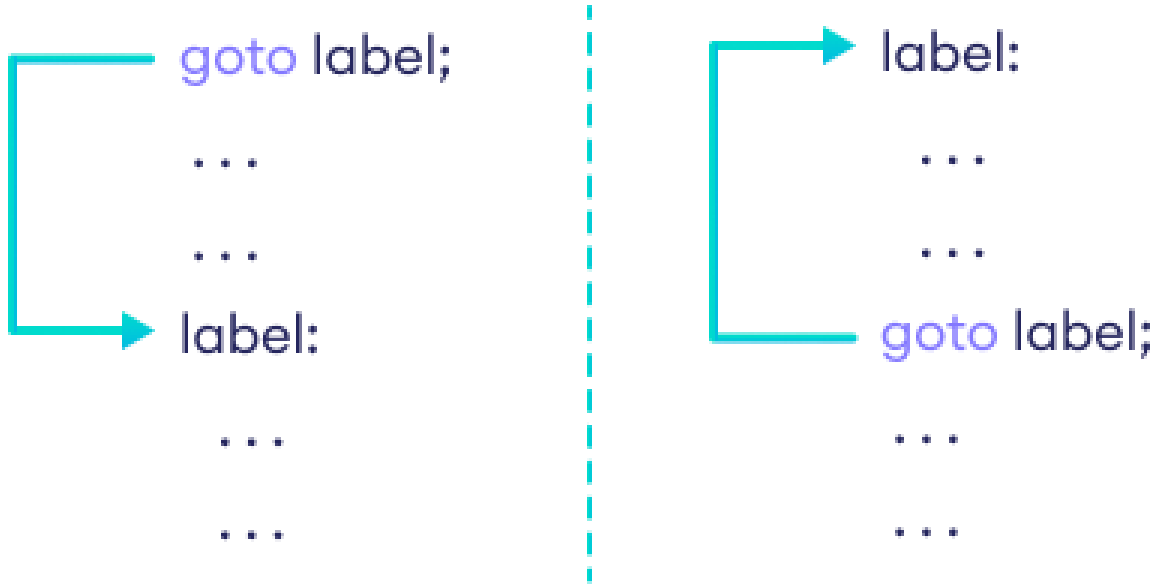
TIME TO PRACTICE

Write a program that scores a blackjack hand. In blackjack, a player receives from two to five cards. The cards 2 through 10 are scored as 2 through 10 points each. The face cards—jack, queen, and king—are scored as 10 points. The goal is to come as close to a score of 21 as possible without going over 21. Hence, any score over 21 is called "busted." The ace can count as either 1 or 11, whichever is better for the user. For example, an ace and a 10 can be scored as either 11 or 21. Since 21 is a better score, this hand is scored as 21. An ace and two 8s can be scored as either 17 or 27. Since 27 is a "busted" score, this hand is scored as 17.

C++ GOTO STATEMENT



C++ GOTO STATEMENT



C++ GOTO STATEMENT

Example: goto Statement

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

int main()
{
    float num, average, sum = 0.0;
    int i, n;

    cout << "Maximum number of inputs: ";
    cin >> n;

    for(i = 1; i <= n; ++i)
    {
        cout << "Enter n" << i << ": ";
        cin >> num;

        if(num < 0.0)
        {
           // Control of the program move to jump:
            goto jump;
        }
        sum += num;
    }

jump:
    average = sum / (i - 1);
    cout << "\nAverage = " << average;
    return 0;
}
```

Output

```
Maximum number of inputs: 10
Enter n1: 2.3
Enter n2: 5.6
Enter n3: -5.6

Average = 3.95
```

HOMEWORK

- https://www.codezclub.com/cpp-convert-uppercase-string-lowercase/
- https://www.codezclub.com/cpp-convert-uppercase-character-lowercase/
- https://www.codezclub.com/cpp-count-words-string/
- https://www.codezclub.com/cpp-program-reverse-string/
- https://www.hackerrank.com/challenges/c-tutorial-for-loop/problem?isFullScreen=true

C++ FUNCTIONS

Functions allow to structure programs in segments of code to perform individual tasks.

In C++, a function is a group of statements that is given a name, and which can be called from some point of the program. The most common syntax to define a function is:

```
type name ( parameter1, parameter2, ...) { statements }
```

Where:
- type is the type of the value returned by the function.
- name is the identifier by which the function can be called.
- parameters (as many as needed): Each parameter consists of a type followed by an identifier, with each parameter being separated from the next by a comma. Each parameter looks very much like a regular variable declaration (for example: `int x`), and in fact acts within the function as a regular variable which is local to the function. The purpose of parameters is to allow passing arguments to the function from the location where it is called from.
- statements is the function's body. It is a block of statements surrounded by braces { } that specify what the function actually does.

C++ FUNCTIONS

```
1  // function example
2  #include <iostream>
3  using namespace std;
4
5  int addition (int a, int b)
6  {
7    int r;
8    r=a+b;
9    return r;
10 }
11
12 int main ()
13 {
14   int z;
15   z = addition (5,3);
```

C++ FUNCTIONS

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

// declaring a function
void greet() {
    cout << "Hello there!";
}

int main() {

    // calling the function
    greet();

    return 0;
}
```

C++ FUNCTIONS

```cpp
// program to print a text

#include <iostream>
using namespace std;

// display a number
void displayNum(int n1, float n2) {
    cout << "The int number is " << n1;
    cout << "The double number is " << n2;
}

int main() {

     int num1 = 5;
     double num2 = 5.5;

    // calling the function
    displayNum(num1, num2);

    return 0;
}
```

C++ FUNCTIONS

Example 3: Add Two Numbers

```
// program to add two numbers using a function

#include <iostream>

using namespace std;

// declaring a function
int add(int a, int b) {
    return (a + b);
}

int main() {

    int sum;

    // calling the function and storing
    // the returned value in sum
    sum = add(100, 78);

    cout << "100 + 78 = " << sum << endl;

    return 0;
}
```

Output

```
100 + 78 = 178
```

C++ FUNCTIONS

Example 3: Add Two Numbers

```
// program to add two numbers using a function

#include <iostream>

using namespace std;

// declaring a function
int add(int a, int b) {
    return (a + b);
}

int main() {

    int sum;

    // calling the function and storing
    // the returned value in sum
    sum = add(100, 78);

    cout << "100 + 78 = " << sum << endl;

    return 0;
}
```

Output

```
100 + 78 = 178
```

C++ FUNCTIONS

Function Prototype

In C++, the code of function declaration should be before the function call. However, if we want to define a function after the function call, we need to use the function prototype. For example,

```
// function prototype
void add(int, int);

int main() {
    // calling the function before declaration.
    add(5, 3);
    return 0;
}

// function definition
void add(int a, int b) {
    cout << (a + b);
}
```

In the above code, the function prototype is:

```
void add(int, int);
```

C++ FUNCTION OVERLOADING

In this tutorial, we will learn about the function overloading in C++ with examples.

In C++, two functions can have the same name if the number and/or type of arguments passed is different.

These functions having the same name but different arguments are known as overloaded functions. For example:

```
// same name different arguments
int test() { }
int test(int a) { }
float test(double a) { }
int test(int a, double b) { }
```

Here, all 4 functions are overloaded functions.

Notice that the return types of all these 4 functions are not the same. Overloaded functions may or may not have different return types but they must have different arguments. For example,

```
// Error code
int test(int a) { }
double test(int b){ }
```

Here, both functions have the same name, the same type, and the same number of arguments. Hence, the compiler will throw an error.

C++ FUNCTION OVERLOADING

```
1   //Illustrates overloading the function name ave.
2   #include <iostream>
3
4   double ave(double n1, double n2);
5   //Returns the average of the two numbers n1 and n2.
6
7   double ave(double n1, double n2, double n3);
8   //Returns the average of the three numbers n1, n2, and n3.
9
10  int main( )
11  {
12      using namespace std;
13      cout << "The average of 2.0, 2.5, and 3.0 is "
14           << ave(2.0, 2.5, 3.0) << endl;
15
16      cout << "The average of 4.5 and 5.5 is "
17           << ave(4.5, 5.5) << endl;
18
19      return 0;
20  }
21
22  double ave(double n1, double n2)
23  {
24      return ((n1 + n2)/2.0);
25  }
26
27  double ave(double n1, double n2, double n3)
28  {
29      return ((n1 + n2 + n3)/3.0);
30  }
31
32
```

two arguments

three arguments

SCOPES

Example 1: Local variable

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

void test();

int main()
{
    // local variable to main()
    int var = 5;

    test();

    // illegal: var1 not declared inside main()
    var1 = 9;
}

void test()
{
    // local variable to test()
    int var1;
    var1 = 6;

    // illegal: var not declared inside test()
    cout << var;
}
```

SCOPES

Example 2: Global variable

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

// Global variable declaration
int c = 12;

void test();

int main()
{
    ++c;

    // Outputs 13
    cout << c <<endl;
    test();

    return 0;
}

void test()
{
    ++c;

    // Outputs 14
    cout << c;
}
```

Output

```
13
14
```

SCOPES

Example 3: Static local variable

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

void test()
{
    // var is a static variable
    static int var = 0;
    ++var;

    cout << var << endl;
}

int main()
{

    test();
    test();

    return 0;
}
```

Output

```
1
2
```

SCOPES

Block Scope Revisited

```
 1    #include <iostream>
 2    using namespace std;
 3
 4    const double GLOBAL_CONST = 1.0;
 5
 6    int function1(int param);
 7
 8    int main()
 9    {
10        int x;
11        double d = GLOBAL_CONST;
12
13        for (int i = 0; i < 10; i++)
14        {
15            x = function1(i);
16        }
17        return 0;
18    }
19
20    int function1(int param)
21    {
22        double y = GLOBAL_CONST;
23        ...
24        return 0;
25    }
```

Local and Global scope are examples of Block scope. A variable can be directly accessed only within its scope.

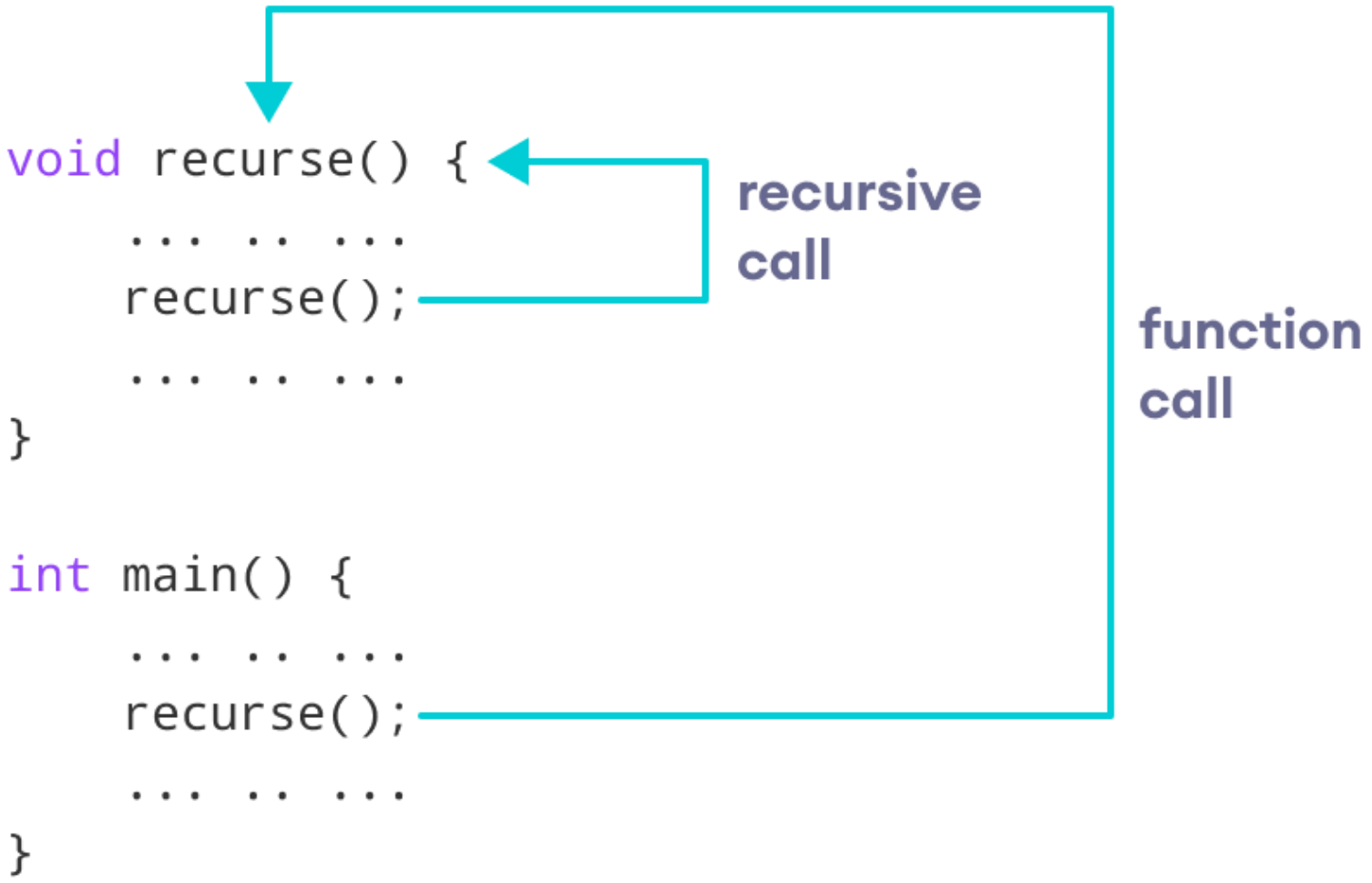
*Block scope: Variable **i** has scope from lines 13-16*

*Local scope to **main:** Variable **x** has scope from lines 10-18 and variable **d** has scope from lines 11-18*

*Global scope: The constant **GLOBAL_CONST** has scope from lines 4-25 and the function **function1** has scope from lines 6-25*

*Local scope to **function1:** Variable **param** has scope from lines 20-25 and variable **y** has scope from lines 22-25*

RECURSION


void recurse() {
... .
recurse();
... .
}
int main() {
... .
recurse();
... .
}
recursive call
function call

RECURSION

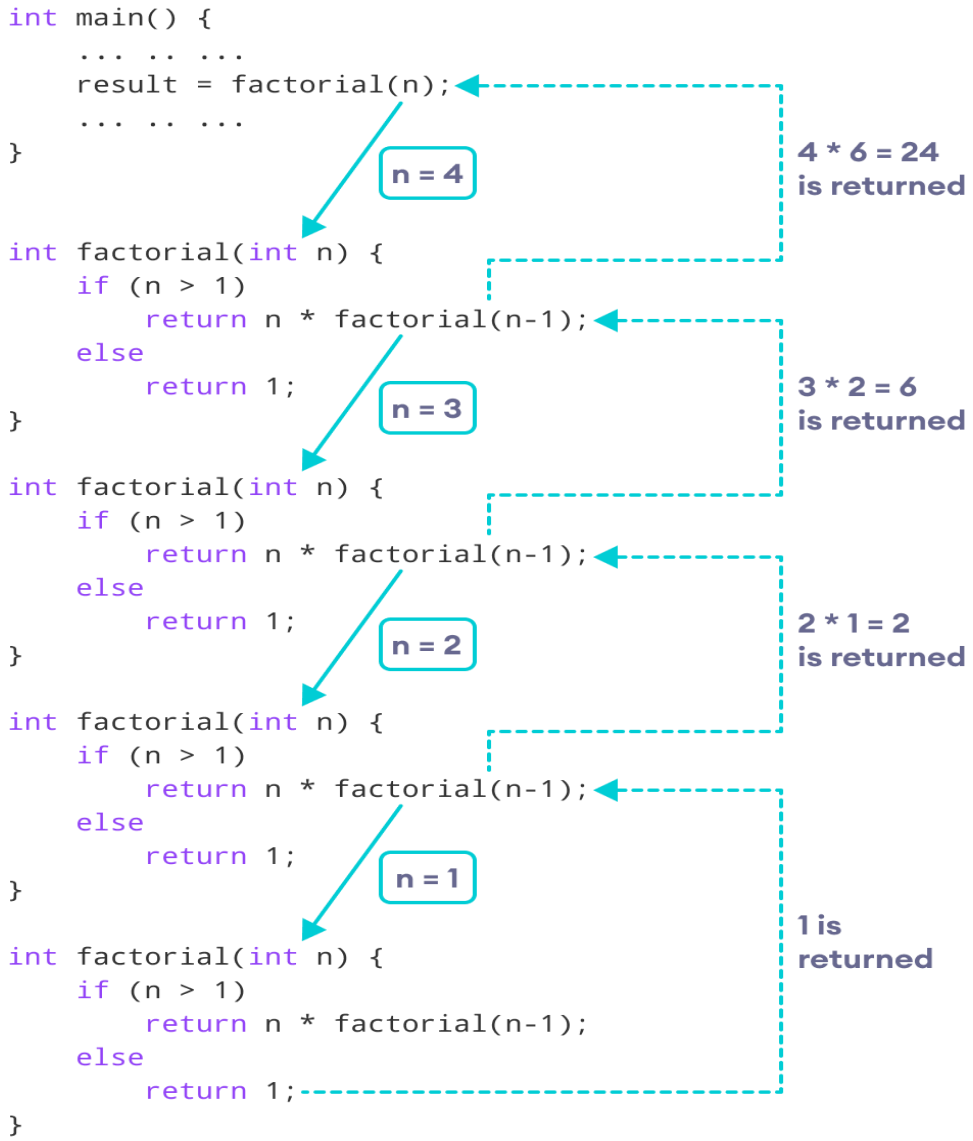
Example 1: Factorial of a Number Using Recursion

```cpp
// Factorial of n = 1*2*3*...*n

#include <iostream>
using namespace std;

int factorial(int);

int main() {
    int n, result;

    cout << "Enter a non-negative number: ";
    cin >> n;

    result = factorial(n);
    cout << "Factorial of " << n << " = " << result;
    return 0;
}

int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

Output

```
Enter a non-negative number: 4
Factorial of 4 = 24
```

RECURSION


int main() {
... .
result = factorial(n);
... .
}
n = 4
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
n = 3
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
n = 2
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
n = 1
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
4 * 6 = 24
is returned
3 * 2 = 6
is returned
2 * 1 = 2
is returned
1 is
returned

CALLING BY VALUE VS. CALLING BY REFERENCE

Difference between call by reference and call by reference (Example)

	Call by value	Call by reference
1	A copy of value is passed to the function	An address of value is passed to the function
2	Changes made inside the function is not reflected on other functions	Changes made inside the function is reflected outside the function also
3	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

CALLING BY VALUE VS. CALLING BY REFERENCE

```
// function definition to swap the values.
void swap(int &x, int &y) {
   int temp;
   temp = x; /* save the value at address x */
   x = y;    /* put y into x */
   y = temp; /* put x into y */

   return;
}
```

C++ BUILT-IN FUNCTIONS

Name	Description	Type of Arguments	Type of Value Returned	Example	Value	Library Header
sqrt	square root	*double*	*double*	sqrt(4.0)	2.0	cmath
pow	powers	*double*	*double*	pow(2.0,3.0)	8.0	cmath
abs	absolute value for *int*	*int*	*int*	abs(-7) abs(7)	7 7	cstdlib
labs	absolute value for *long*	*long*	*long*	labs(-70000) labs(70000)	70000 70000	cstdlib
fabs	absolute value for *double*	*double*	*double*	fabs(-7.5) fabs(7.5)	7.5 7.5	cmath
ceil	ceiling (round up)	*double*	*double*	ceil(3.2) ceil(3.9)	4.0 4.0	cmath
floor	floor (round down)	*double*	*double*	floor(3.2) floor(3.9)	3.0 3.0	cmath
srand	Seed random number generator	*none*	*none*	srand()	none	cstdlib
rand	Random number	*none*	*int*	rand()	0-RAND_MAX	cstdlib

SELF-TEST EXERCISES

12. Write a function definition for a function called `even` that takes one argument of type `int` and returns a `bool` value. The function returns `true` if its one argument is an even number; otherwise, it returns `false`.

13. Write a function definition for a function `is_digit` that takes one argument of type `char` and returns a `bool` value. The function returns `true` if the argument is a decimal digit; otherwise, it returns `false`.

14. Write a function definition for a function `is_root_of` that takes two arguments of type `int` and returns a `bool` value. The function returns `true` if the first argument is the square root of the second; otherwise, it returns `false`.

23. Write a function declaration and function definition for a function called `read_filter` that has no parameters and that returns a value of type `double`. The function `read_filter` prompts the user for a value of type `double` and reads the value into a local variable. The function returns the value read provided this value is greater than or equal to zero and returns zero if the value read is negative.

HOMEWORK

- https://www.hackerrank.com/challenges/c-tutorial-functions/problem?isFullScreen=true
- https://www.codezclub.com/cpp-program-check-prime-number/
- https://www.codezclub.com/cpp-swap-two-characters-integers-call-value/
- https://www.codezclub.com/cpp-add-two-time-passing-objects-function-argument-call-reference/
- https://www.codezclub.com/cpp-add-two-time-by-call-address/
- https://www.codezclub.com/cpp-print-series-using-function-x-x33-x55-xnn/
- https://www.codezclub.com/cpp-add-sub-mul-two-numbers-using-function/
- https://www.codezclub.com/cpp-find-factorial-number-functions/
- https://www.codezclub.com/cpp-swap-two-numbers-using-call-value/

C++ ARRAYS

Suppose we wish to write a program that reads in five test scores and performs some manipulations on these scores. For instance, the program might compute the highest test score and then output the amount by which each score falls short of the highest. The highest score is not known until all five scores are read in. Hence, all five scores must be retained in storage so that after the highest score is computed each score can be compared to it.

To retain the five scores, we will need something equivalent to five variables of type `int`. We could use five individual variables of type `int`, but five variables are hard to keep track of, and we may later want to change our program to handle 100 scores; certainly, 100 variables are impractical. An array is the perfect solution. An **array** behaves like a list of variables with a uniform naming mechanism that can be declared in a single line of simple code. For example, the names for the five individual variables we need might be `score[0]`, `score[1]`, `score[2]`, `score[3]`, and `score[4]`. The part that does not change—in this case, `score`—is the name of the array. The part that can change is the integer in the square brackets, `[ ]`.

C++ ARRAYS

Array Declaration

SYNTAX

```
Type_Name Array_Name[Declared_Size];
```

EXAMPLES

```
int big_array[100];
double a[3];
double b[5];
char grade[10], one_grade;
```

An array declaration, of the form shown, will define *Declared_Size* indexed variables, namely, the indexed variables *Array_Name*[0] through *Array_Name*[*Declared_Size*-1]. Each indexed variable is a variable of type *Type_Name*.

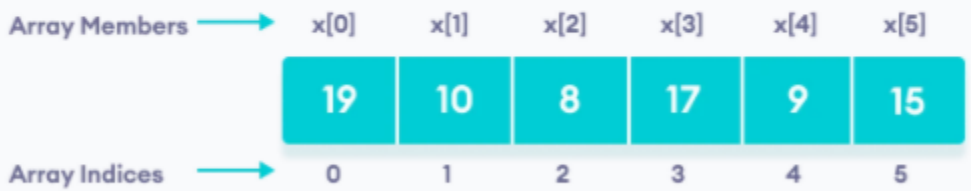
The array a consists of the indexed variables a[0], a[1], and a[2], all of type *double*. The array b consists of the indexed variables b[0], b[1], b[2], b[3], and b[4], also all of type *double*. You can combine array declarations with the declaration of simple variables such as the variable one_grade shown above.

C++ ARRAYS

C++ Array Initialization

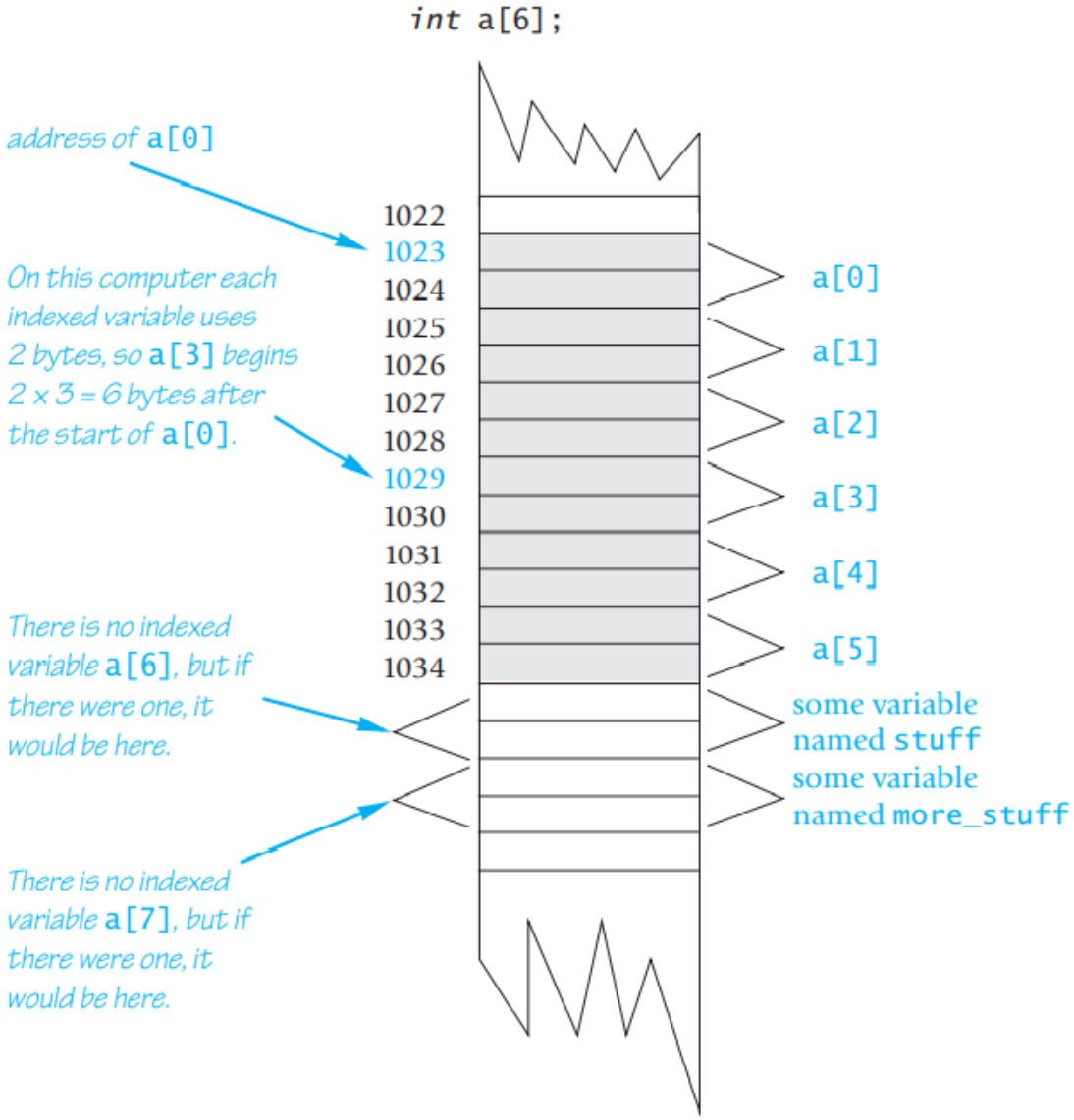
In C++, it's possible to initialize an array during declaration. For example,

```
// declare and initialize and array
int x[6] = {19, 10, 8, 17, 9, 15};
```



C++ Array elements and their data

C++ ARRAYS


int a[6];
address of a[0]
On this computer each indexed variable uses 2 bytes, so a[3] begins 2 x 3 = 6 bytes after the start of a[0].
There is no indexed variable a[6], but if there were one, it would be here.
There is no indexed variable a[7], but if there were one, it would be here.
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
some variable named stuff
some variable named more_stuff

C++ ARRAYS

C++ Array With Empty Members

In C++, if an array has a size `n`, we can store upto `n` number of elements in the array. However, what will happen if we store less than `n` number of elements.

For example,

```
// store only 3 elements in the array
int x[6] = {19, 10, 8};
```

Here, the array `x` has a size of `6`. However, we have initialized it with only 3 elements.

In such cases, the compiler assigns random values to the remaining places. Oftentimes, this random value is simply `0`.

Empty array members are automatically assigned the value 0

C++ ARRAYS

C++ Array With Empty Members

In C++, if an array has a size `n`, we can store upto `n` number of elements in the array. However, what will happen if we store less than `n` number of elements.

For example,

```
// store only 3 elements in the array
int x[6] = {19, 10, 8};
```

Here, the array `x` has a size of `6`. However, we have initialized it with only 3 elements.

In such cases, the compiler assigns random values to the remaining places. Oftentimes, this random value is simply `0`.

Empty array members are automatically assigned the value 0

C++ ARRAYS

How to insert and print array elements?

```
int mark[5] = {19, 10, 8, 17, 9}

// change 4th element to 9
mark[3] = 9;

// take input from the user
// store the value at third position
cin >> mark[2];


// take input from the user
// insert at ith position
cin >> mark[i-1];

// print first element of the array
cout << mark[0];

// print ith element of the array
cout >> mark[i-1];
```

C++ ARRAYS

Example 1: Displaying Array Elements

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

int main() {
    int numbers[5] = {7, 5, 6, 12, 35};

    cout << "The numbers are: ";

    //  Printing array elements
    // using range based for loop
    for (const int &n : numbers) {
        cout << n << "  ";
    }


    cout << "\nThe numbers are: ";

    //  Printing array elements
    // using traditional for loop
    for (int i = 0; i < 5; ++i) {
        cout << numbers[i] << "  ";
    }

    return 0;
}
```

Output

```
The numbers are: 7  5  6  12  35
The numbers are: 7  5  6  12  35
```

C++ ARRAYS

SELF-TEST EXERCISES

3. Identify any errors in the following array declarations.

   a. `int x[4] = { 8, 7, 6, 4, 3 };`

   b. `int x[ ] = { 8, 7, 6, 4 };`

   c. `const int SIZE = 4;`

   d. `int x[SIZE];`

4. What is the output of the following code?

```
char symbol[3] = {'a', 'b', 'c'};
for (int index = 0; index < 3; index++)
    cout << symbol[index];
```

5. What is the output of the following code?

```
double a[3] = {1.1, 2.2, 3.3};
cout << a[0] << " " << a[1] << " " << a[2] << endl;
a[1] = a[2];
cout << a[0] << " " << a[1] << " " << a[2] << endl;
```

C++ ARRAYS

SELF-TEST EXERCISES

6. What is the output of the following code?

```
int i, temp[10];

for (i = 0; i < 10; i++)
    temp[i] = 2 * i;

for (i = 0; i < 10; i++)
    cout << temp[i] << " ";
cout << endl;

for (i = 0; i < 10; i = i + 2)
    cout << temp[i] << " ";
```

7. What is wrong with the following piece of code?

```
int sample_array[10];

for (int index = 1; index <= 10; index++)
    sample_array[index] = 3 * index;
```

C++ POINTERS

A **pointer** is the memory address of a variable. Recall that the computer's memory is divided into numbered memory locations (called bytes) and that variables are implemented as a sequence of adjacent memory locations. Recall also that sometimes the C++ system uses these memory addresses as names for the variables. If a variable is implemented as, say, three memory locations, then the address of the first of these memory locations is sometimes used as a name for that variable. For example, when the variable is used as a call-by-reference argument, it is this address, not the identifier name of the variable, that is passed to the calling function.

An address that is used to name a variable in this way (by giving the address in memory where the variable starts) is called a *pointer* because the address can be thought of as "pointing" to the variable. The address "points" to the variable because it identifies the variable by telling *where* the variable is, rather than telling what the variable's name is. A variable that is, say, at location number 1007 can be pointed out by saying "it's the variable over there at location 1007."

C++ POINTERS

Pointer Variable Declarations

A variable that can hold pointers to other variables of type *Type_Name* is declared similarly to the way you declare a variable of type *Type_Name*, except that you place an asterisk at the beginning of the variable name.

SYNTAX

```
Type_Name *Variable_Name1, *Variable_Name2, . . .;
```

EXAMPLE

```
double *pointer1, *pointer2;
```

```
v1 = 0;
p1 = &v1;
*p1 = 42;
cout << v1 << endl;
cout << *p1 << endl;
```

This code outputs the following to the screen:

```
42
42
```

C++ POINTERS

The * and & Operators

The *operator in front of a pointer variable produces the variable it points to. When used this way, the *operator is called the **dereferencing operator.**

The operator & in front of an ordinary variable produces the address of that variable; that is, produces a pointer that points to the variable. The & operator is called the address-of operator.

For example, consider the declarations

```
double *p, v;
```

The following sets the value of p so that p points to the variable v:

```
p = &v;
```

*p produces the variable pointed to by p, so after the assignment above, *p and v refer to the same variable. For example, the following sets the value of v to 9.99, even though the name v is never explicitly used:

```
*p = 9.99;
```

C++ POINTERS

The * and & Operators

The *operator in front of a pointer variable produces the variable it points to. When used this way, the *operator is called the **dereferencing operator.**

The operator & in front of an ordinary variable produces the address of that variable; that is, produces a pointer that points to the variable. The & operator is called the address-of operator.

For example, consider the declarations

```
double *p, v;
```

The following sets the value of p so that p points to the variable v:

```
p = &v;
```

*p produces the variable pointed to by p, so after the assignment above, *p and v refer to the same variable. For example, the following sets the value of v to 9.99, even though the name v is never explicitly used:

```
*p = 9.99;
```

C++ POINTERS

DISPLAY 9.2 Basic Pointer Manipulations *(part 1 of 2)*

```
1    //Program to demonstrate pointers and dynamic variables.
2    #include <iostream>
3    using namespace std;
4
5    int main( )
6    {
7        int *p1, *p2;
8
9        p1 = new int;
10       *p1 = 42;
11       p2 = p1;
12       cout<< "*p1 == " << *p1 << endl;
13       cout<< "*p2 == " << *p2 << endl;

14       *p2 = 53;
15       cout<< "*p1 == " << *p1 << endl;
16       cout<< "*p2 == " << *p2 << endl;

17       p1 = new int;
18       *p1 = 88;
19       cout<< "*p1 == " << *p1 << endl;
20       cout<< "*p2 == " << *p2 << endl;
21       cout<< "Hope you got the point of this example!\n";
22       return 0;
23   }
```

(continued)

C++ POINTERS

DISPLAY 9.2 Basic Pointer Manipulations *(part 2 of 2)*

Sample Dialogue

```
*p1 == 42
*p2 == 42
*p1 == 53
*p2 == 53
*p1 == 88
*p2 == 53
Hope you got the point of this example!
```

C++ POINTERS

SELF-TEST EXERCISES

4. What is the output produced by the following code?

```
int *p1, *p2;
p1 = new int;
p2 = new int;
*p1 = 10;
*p2 = 20;
cout << *p1 << " " << *p2 << endl;
p1 = p2;
cout << *p1 << " " << *p2 << endl;
*p1 = 30;
cout << *p1 << " " << *p2 << endl;
```

How would the output change if you were to replace

```
*p1 = 30;
```

with the following?

```
*p2 = 30;
```

C++ POINTERS

SELF-TEST EXERCISES

5. What is the output produced by the following code?

```
int *p1, *p2;
p1 = new int;
p2 = new int;
*p1 = 10;
*p2 = 20;
cout << *p1 << " " << *p2 << endl;
*p1 = *p2; //This is different from Exercise 4
cout << *p1 << " " << *p2 << endl;
*p1 = 30;
cout << *p1 << " " << *p2 << endl;
```

HOMEWORK

- https://www.codezclub.com/cpp-find-maximum-largest-number-array/
- https://www.codezclub.com/cpp-enter-5-numbers-display-first-and-last/
- https://www.codezclub.com/cpp-reverse-array-elements-entered-user/
- https://www.codezclub.com/cpp-sort-array-elements-ascending-order/
- https://www.codezclub.com/cpp-find-duplicate-elements-array/

SOURCES

Books:

-Problem Solving with C++ ninth edition

Websites:

-Geeks for Geeks

-Tutorials point

-programiz

-w3schools

-cplusplus.com

- CodezClub

- HackerRank