

Kandula Srinivasa Reddy Memorial College of Engineering (Autonomous)

Kadapa-516003. AP

(Approved by AICTE, Affiliated to JNTUA, Ananthapuramu, Accredited by NAAC)

(An ISO 9001-2008 Certified Institution)

Department of Electronics and Communication Engineering



Certification Course

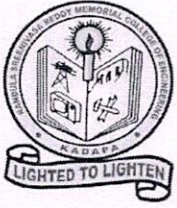
On

“C++ language”

Resource Person : Sri S Khaja Khizar

Course Coordinator: Dr. S L Prathapa Reddy

Duration : 09-04-2022 to 24-04-2022



K.S.R.M. COLLEGE OF ENGINEERING

(UGC - AUTONOMOUS)

Kadapa, Andhra Pradesh, India - 516003

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.

An ISO 14001:2004 & 9001: 2015 Certified Institution



Lr./KSRMCE/ (Department of ECE)/2020-21

Date: 02/04/2022

To
The Principal
KSRM College of Engineering
Kadapa, AP.

Sub: KSRMCE - (Department of ECE) – Permission to conduct a certification course on “C++ Language”
–Request– reg.

---***---

Respected Sir,

With reference to the cited, the Department of ECE is planning to conduct a certification course on “C++ Language” for All B.Tech IV SEM students from 09-04-2022 to 24-04-2022. In this regard, I kindly request you to grant us permission to conduct a certification course. This is submitted for your kind perusal.

Thanking you sir,

S. L. Prathapa Reddy
Yours Faithfully

Coordinator

Dr .S L Prathapa Reddy

Cc:

To The Director for Information

To All Deans/HODs

Permitted
V. S. S. Mm/5
PRINCIPAL
K.S.R.M. COLLEGE OF ENGINEERING
KADAPA-516005, (A.P.)



/ksrmce.ac.in

Follow Us:



/ksrmceofficial



K.S.R.M. COLLEGE OF ENGINEERING

(UGC - AUTONOMOUS)

Kadapa, Andhra Pradesh, India - 516003

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.

An ISO 14001:2004 & 9001: 2015 Certified Institution



Date: 02/04/2022

Circular

All the B.Tech IV SEM students are hereby informed that the Department of ECE is going to conduct certification course on "C++ Language" from 09/04/2022 to 24/04/2022. Interested students may register their names with respective faculty members on or before 06/04/2022.

For any queries contact,

Coordinator

Dr .S L Prathapa Reddy, Associate Professor, ECE Dept.,

HOD

Professor & H.O.D.
Department of E.C.E.
K.S.R.M. College of Engineering
KADAPA - 516 003

Cc to:

The Management /Director / All Deans / All HODS/Staff / Students for information

The IQAC Cell for Documentation



/ksrmce.ac.in

Follow Us:



/ksrmceofficial



K.S.R.M. COLLEGE OF ENGINEERING

(UGC - AUTONOMOUS)

Kadapa, Andhra Pradesh, India - 516003

Approved by AICTE, New Delhi & Affiliated to JNTUA,
Ananthapuramu.

An ISO 14001:2004 & 9001: 2015 Certified Institution



Department of Electronics & Communication Engineering Certificate Course on C++ Language Registration Form

S.No.	Roll.No.	Name of the Student	Branch	Sem	Signature
1	209YIA0401	A. Harsha vardhan	ECE	IV	Harsha
2	209YIA0402	A. SIDDIQ	ECE	IV	SIDDIQ
3	209YIA0405	A. SHASHIKALA	ECE	IV	shashikala
4	209YIA0409	Bandi Kalyani	ECE	IV	Bandi Kalyani
5	209YIA0412	Banne Nareesh	ECE	IV	Banne Nareesh
6	209YIA0417	BOHEM SANGEETHA	ECE	IV	Bohem Sangeetha
7	209YIA0419	BOYA SUDHEER	ECE	IV	Boya Sudheer
8	209YIA0421	C. Shaik Abdulillah	ECE	IV	Shaik
9	209YIA0422	Chakali. Prabhakar	ECE	IV	Chakali
10	209YIA0425	chamanchi Venkata laksh	ECE	IV	Venkata laksh
11	209YIA0427	C. Rajeshwara Reddy	ECE	IV	Rajeshwara
12	209YIA0430	C. MADHAVI	ECE	IV	Madhavi
13	209YIA0433	C. Haris Priya	ECE	IV	Priya
14	179YIA0437	D. Sivaiah	ECE	IV	D Sivaiah
15	209YIA0441	D. Swetha	ECE	IV	D. Swetha
16	209YIA0445	D. Jaya chandra	ECE	IV	D. Jaya chandra
17	209YIA0449	G. 'Vamsi'	ECE	IV	Vamsi
18	209YIA0454	G. Sai Prathap	ECE	IV	G. Sai Prathap
19	209YIA0457	G. BHARGAVI	ECE	IV	G. Bhargavi
20	209YIA0461	G. Hari Krishna	ECE	IV	G. Hari Krishna
21	209YIA0467	K. Anitha	ECE	IV	K. Anitha
22	209YIA0470	K. Ganesh	ECE	IV	Ganesh
23	209YIA0474	K. Lavanya	ECE	IV	K. Lavanya

24	209YIA0478	K. Poojitha	ECE	IV	K. Poojitha
25	209YIA0482	M. Sujitha Rani	ECE	IV	M. Sujitha
26	209YIA0485	M. Krishna Sai Goud	ECE	IV	M. Krishna Sai
27	209YIA0490	M. Girisath Reddy	ECE	IV	M. Girisath
28	209YIA0493	M. Rajiv	ECE	IV	M. Raj
29	209YIA0497	N. Indra Kalyan Reddy	ECE	IV	N. Indrakalyan
30	209YIA04A0	N. Subbarayudu	ECE	IV	Subbarayudu
31	209YIA04A3	N. Hazathi	ECE	IV	N. Hazathi
32	209YIA04A8	P. Chandra Sekhar	ECE	IV	P. Chandra Sekhar
33	209YIA04B2	P. Usha Rani	ECE	IV	P. Usha
34	209YIA04B6	P. Girisath Reddy	ECE	IV	Girisath
35	209YIA04C0	S. Arshiga	ECE	IV	S. Arshiga
36	209YIA04C2	S. Vasanth	ECE	IV	S. Vasanth
37	209YIA04C5	S. Beebi Ayesha Siddika	ECE	IV	S. Beebi Ayesha
38	209YIA04C8	S. Mohammed Faizan	ECE	IV	S. Mohan
39	209YIA04D1	S. Haik SATHIEL	ECE	IV	S. Haik
40	209YIA04DU	S. Venkata Sai	ECE	IV	S. Venkatesai
41	209YIA04D6	S. Sumanth Reddy	ECE	IV	Sumanth
42	209YIA04E1	T. Reddikala	ECE	IV	T. Reddy
43	209YIA04E3	T. Nitish Kumar Reddy	ECE	IV	T. Nitish Kumar
44	209YIA04E7	T. Kusuma	ECE	IV	T. Kusuma
45	209YIA04F1	V. Nandini	ECE	IV	V. Nandini
46	209YIA04F5	V. Swappa	ECE	IV	V. Swappa
47	209YIA04F7	Y. Bhoorathi	ECE	IV	Y. Bhoorathi
48	209YIA04F9	Y. Saikiran Reddy	ECE	IV	Y. Saikiran
49	219Y5A04D2	B. Pallavi	ECE	IV	B. Pallavi
50	219Y5A0403	B. Parimala	ECE	IV	B. Parimala
51	219Y5A0408	G. MAINA	ECE	IV	G. MAINA
52	219Y5A0409	M. Mounika	ECE	IV	M. Mounika
53	219Y5A0411	M. Mounika	ECE	IV	M. Mounika
54	219Y5A0412	M. Malli Shiva	ECE	IV	M. Shiva
55	219Y5A0413	P. Nagendra	ECE	IV	P. Nagendra
56	219Y5A0414	P. Faruq Khan	ECE	IV	P. Faruq Khan
57	219Y5A04K	S. Arun Kumar	ECE	IV	S. Arun Kumar
58	219Y5A0416	S. Charan Kumar Reddy	ECE	IV	S. Charan Kumar

59	21945A0417	Shaik Mahammad Sharif	ECE	IV	Mohammad
60	21945A0418	Vandadi Baby	ECE	IV	Vandadi
61	21945A0405	G. Kanya	ECE	IV	G. Kanya
62	21945A0406	G. Madhukumar	ECE	IV	G. Madhukumar
63					

S. S. Reddy
Coordinator(s)

G. H. H.
HOD

Professor & H.O.D.
Department of E.C.E.
K.S.R.M. College of Engineering
KADAPA - 516 003

C++ Language Syllabus

Overview:

C++ language is a superset of the 'C' language and was initially known as "C with Classes". In "C" operator ++ is used to increment the value by 1. That means to the language 'C', developers have added some extra features and hence named as C++. C++ is a general purpose programming language and supports object oriented programming features.

Course Objectives:

- Understanding about Structured Programming.
- Understanding about object oriented programming.
- Gain knowledge about the capability to store information together in an object.
- Learn how to store one object inside another object
- Learn use of one method can be used in variety of different ways
- Understanding the process of exposing the essential data to the outside of the world and hiding the low level data.
- Create and process data in files using file I/O functions.

Module 1:

C Language Fundamentals

Data Types, Constant and Variables,
Statements, Expressions, Operators,
Control structures, Decision making and Branching, Decision making & looping,
Functions, Arrays, Pointers, and File Handling
FAQs
Assignment - I

Module 2:

C++ Overview, Functions and Variables

C++ Characteristics
Object-Oriented Terminology
Polymorphism
Object-Oriented Paradigm
Abstract Data Types
I/O Services
Standard Template Library
Functions: Declaration and Definition

Variables: Definition, Declaration, and Scope
Variables: Dynamic Creation and Derived Data
Arrays and Strings in C++

Module 3:

Classes in C++, Operator Overloading

Defining Classes in C++
Classes and Encapsulation
Member Functions
Instantiating and Using Classes
Using Constructors
Multiple Constructors and Initialization Lists
Using Destructors to Destroy Instances
Friendship
Operator Overloading
Working with Overloaded Operator Methods
Initialization vs. Assignment
The Copy Constructor
Assigning Values
Specialized Constructors and Methods
Constant and Static Class Members
FAQs
Assignments-II from Module2 & Module3

Module 4:

Storage Management, Inheritance and Polymorphism

Memory Allocation
Dynamic Allocation: new and delete
Overview of Inheritance
Defining Base and Derived Classes
Constructor and Destructor Calls
Overview of Polymorphism
FAQs
Assignments – III

Module 5:

Streams & Templates

Standard Streams
Manipulators
Unformatted Input and Output
File Input and Output
Template Overview
Customizing a Templated Method
Standard Template Library Containers
FAQs
Assignments - IV

Textbook:

1. Programming in ANSI C - Balaguruswami, TMH
2. E. Balaguruswamy, Object-Oriented Programming in C++. 4 ed, Tata McGraw-Hill.
3. Ashok N. Kamthane, Object-Oriented Programming with ANSI & Turbo C++, Pearson Education.

At the end of the course participants will be able to

- Work with Data types, control statements and loops.
- Work with Arrays, Functions and pointers.
- Work with Object Oriented Programming concepts
- Work with Inheritance, Polymorphism and Templates
- Work with File Streams.



K.S.R.M. COLLEGE OF ENGINEERING

(UGC-AUTONOMOUS)

Kadapa, Andhra Pradesh, India- 516 005

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.

An ISO 14001:2004 & 9001: 2015 Certified Institution



Department of Electronics & Communication Engineering

Certificate Course on C++ language

Schedule

S.No	Date	Time	Faculty	Topic
1	09/04/2022	3 PM to 4PM	Dr G Hemalatha Sri Khaja Khizar Dr S L Prathapa Reddy	Inauguration
2	9/04/2022	4PM to 5PM	Sri Khaja Khizar	Introduction, C Language Fundamentals.
3	11/04/2022	3PM to 4PM	Sri Khaja Khizar	Data Types
4	11/04/2022	4PM to 5PM	Sri Khaja Khizar	Constant and Variables,
5	12/04/2022	3PM to 4PM	Sri Khaja Khizar	Statements, Expressions, Operators
6	12/04/2022	4PM to 5PM	Sri Khaja Khizar	Control structures, Decision making and Branching, Decision making & looping
7	13/04/2022	3PM to 4PM	Sri Khaja Khizar	Functions, Arrays, Pointers, and File Handling
8	13/04/2022	4PM to 5PM	Sri Khaja Khizar	C++ Overview, Functions and Variables
9	14/04/2022	3PM to 4PM	Sri Khaja Khizar	C++ Characteristics Object-Oriented Terminology
10	14/04/2022	4PM to 5PM	Sri Khaja Khizar	Polymorphism Object-Oriented Paradigm
11	15/04/2022	3PM to 4PM	Sri Khaja Khizar	Abstract Data Types I/O Services Standard Template Library
12	15/04/2022	4PM to 5PM	Sri Khaja Khizar	Functions: Declaration and Definition
13	16/04/2022	3PM to 4PM	Sri Khaja Khizar	Variables: Definition, Declaration, and Scope
14	16/04/2022	4PM to 5PM	Sri Khaja Khizar	Variables: Dynamic Creation and Derived Data
15	17/04/2022	3PM to 4PM	Sri Khaja Khizar	Arrays and Strings in C++
17	17/04/2022	4 PM to 5PM	Sri Khaja Khizar	Classes in C++, Operator Overloading



K.S.R.M. COLLEGE OF ENGINEERING (UGC-AUTONOMOUS)

Kadapa, Andhra Pradesh, India- 516 005

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.

An ISO 14001:2004 & 9001: 2015 Certified Institution



18	18/04/2022	3PM to 4PM	Sri Khaja Khizar	Defining Classes in C++ Classes and Encapsulation
19	18/04/2022	4 PM to 5PM	Sri Khaja Khizar	Member Functions Instantiating and Using Classes
20	19/04/2022	3PM to 4PM	Sri Khaja Khizar	Using Constructors Multiple Constructors and Initialization Lists
21	19/04/2022	4 PM to 5PM	Sri Khaja Khizar	Using Destructors to Destroy Instances Friendship Operator Overloading
22	20/04/2022	3PM to 4PM	Sri Khaja Khizar	Working with Overloaded Operator Methods
23	20/04/2022	4 PM to 5PM	Sri Khaja Khizar	Assigning Values Specialized Constructors and Methods
24	21/04/2022	2PM to 3PM	Sri Khaja Khizar	Memory Allocation Dynamic Allocation: new and delete
25	21/04/2022	3 PM to 4PM	Sri Khaja Khizar	Overview of Inheritance
26	21/04/2022	4PM to 5PM	Sri Khaja Khizar	Defining Base and Derived Classes
27	22/04/2022	3 PM to 4PM	Sri Khaja Khizar	Constant and Static Class Members
28	22/04/2022	4PM to 5PM	Sri Khaja Khizar	Initialization vs. Assignment The Copy Constructor
29	23/04/2022	4 PM to 5PM	Sri Khaja Khizar	Constructor and Destructor Calls
30	23/04/2022	3PM to 4PM	Sri Khaja Khizar	Overview of Polymorphism
31	24/04/2022	4 PM to 5PM	Sri Khaja Khizar	Exam
32	24/04/2022	3PM to 4PM	Dr G Hemalatha Sri Khaja Khizar Dr S L Prathapa Reddy	Exam and certificate distribution

S. L. Hemalatha
COORDINATOR

G. H. Hemalatha
HOD

Professor & H.O.D.
Department of E.C.E.
K.S.R.M. College of Engineering
KADAPA - 516 003



Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.

Department of Electronics & Communication Engineering

Attendance Sheet

[illegible]

[illegible]

46	209Y1A04F5	VEERAPURAM SWAPNA	P	P	P	A	P	P	A	P	P	P	P	P	P	P	P	A	P
47	209Y1A04F7	YAKASI BHARATH	P	A	A	P	P	P	P	P	P	P	P	P	P	P	A	P	P
48	209Y1A04F9	YELAMPALLE SAIKIRAN REDDY	P	P	P	P	A	A	P	P	P	P	P	P	P	P	P	P	P
49	219Y5A0402	BODDU PALLAVI	A	P	P	P	P	A	P	P	P	P	P	P	P	P	P	A	P
50	219Y5A0403	BOGGULA PARIMALA	P	P	P	P	P	P	A	P	A	P	A	A	P	P	P	P	A
51	219Y5A0405	G KAVYA	P	P	P	A	P	P	P	A	A	P	P	P	P	P	P	P	P
52	219Y5A0406	GANJI KUNTA MADHU KUMAR	P	P	P	P	P	P	P	A	P	P	A	P	P	P	P	P	P
53	219Y5A0408	GORLA MAINA	A	P	P	P	P	P	P	P	P	A	A	P	P	P	P	P	P
54	219Y5A0409	KARELLA RAGHAVENDRA)	P	P	P	P	P	P	P	P	A	A	P	A	P	P	P	P	P
55	219Y5A0411	MEESALA MOUNIKA	P	A	P	P	P	P	A	P	P	P	P	P	P	P	P	P	P
56	219Y5A0412	MUKKA MALLA SHIVA	P	A	P	P	P	P	P	A	P	P	P	P	P	P	P	P	P
57	219Y5A0413	PASUPULETI NAGENDRA	P	A	P	P	P	P	A	A	P	A	A	A	P	A	P	P	P
58	219Y5A0414	PATAN FARUQ KHAN	P	P	P	P	P	P	P	P	A	P	P	P	P	A	P	A	P
59	219Y5A0415	SANE ARUN KUMAR	A	P	A	P	P	P	P	P	P	P	P	A	P	P	P	P	P
60	219Y5A0416	SANGALA CHARAN KUMAR REDDY	P	A	P	A	P	A	P	A	P	A	P	P	A	P	A	P	P
61	219Y5A0417	SHAIK MAHAMMAD SHARIF	P	P	P	P	P	P	P	A	P	P	A	P	P	P	P	P	A
62	219Y5A0418	VANDADI BABY	P	A	A	P	A	P	P	P	P	P	A	A	P	P	P	P	P

S. L. Reddy
Coordinator

S. H. Reddy
HOD
Professor & H.O.D.
Department of E.C.E.
K.S.R.M. College of Engineering
KADAPA - 616 003



K.S.R.M. COLLEGE OF ENGINEERING

(UGC - Autonomous)

Kadapa, Andhra Pradesh, India - 516003

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.



Department of ECE

Certification Course on "C++ Language"

Venue
CRI LAB

DATE(S)
09-04-2022 to
24-04-2022

Cordinators

Dr.S.L.Prathap Reddy
Assoc.Professor, Dept of ECE

Resource Persons

Sri.S.Khaja Khizar

Dr. G. Hemalatha H.O.D	Dr. V.S.S. Murthy Principal	Dr. Kandula Chandra Obul Reddy Managing Director	Smt. K. Rajeswari Correspondent Secretary, Tresurer	Sri K. Madan Mohan Reddy Vice-Chairman	Sri K. Raja Mohan Reddy Chairman
---------------------------	--------------------------------	---	---	---	-------------------------------------



K.S.R.M. COLLEGE OF ENGINEERING

(UGC-AUTONOMOUS)

Kadapa, Andhra Pradesh, India- 516 003

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.

An ISO 14001:2004 & 9001: 2015 Certified Institution



ACTIVITY REPORT

Certification Course

On

“C++ Language”

09th April 2022 to 24th April 2022

Target Group	:	IV SEM Students
Details of Participants	:	63 Students
Coordinator(s)	:	Dr S L Prathapa Reddy, Assoc. Prof, Dept. of ECE
Organizing Department	:	Department of Electronics and Communication Engineering
Venue	:	CRI lab

Description:

Certification course on “C++ Language” was organized by Dept. of ECE from 09-04-2022 to 24-04-2022. Sri S Khaja Khizar acted as Course instructor. C++ is a general-purpose programming and coding language. C++ is used in developing browsers, operating systems, and applications, as well as in-game programming, software engineering, data structures, learning was explained.



K.S.R.M. COLLEGE OF ENGINEERING

(UGC - Autonomous)

Kadapa, Andhra Pradesh, India - 516003

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.



KSNR
lives on.

Department of ECE

Certification Course on "C++ Language"

Venue
CRI LAB

DATE(S)
09-04-2022 to
24-04-2022

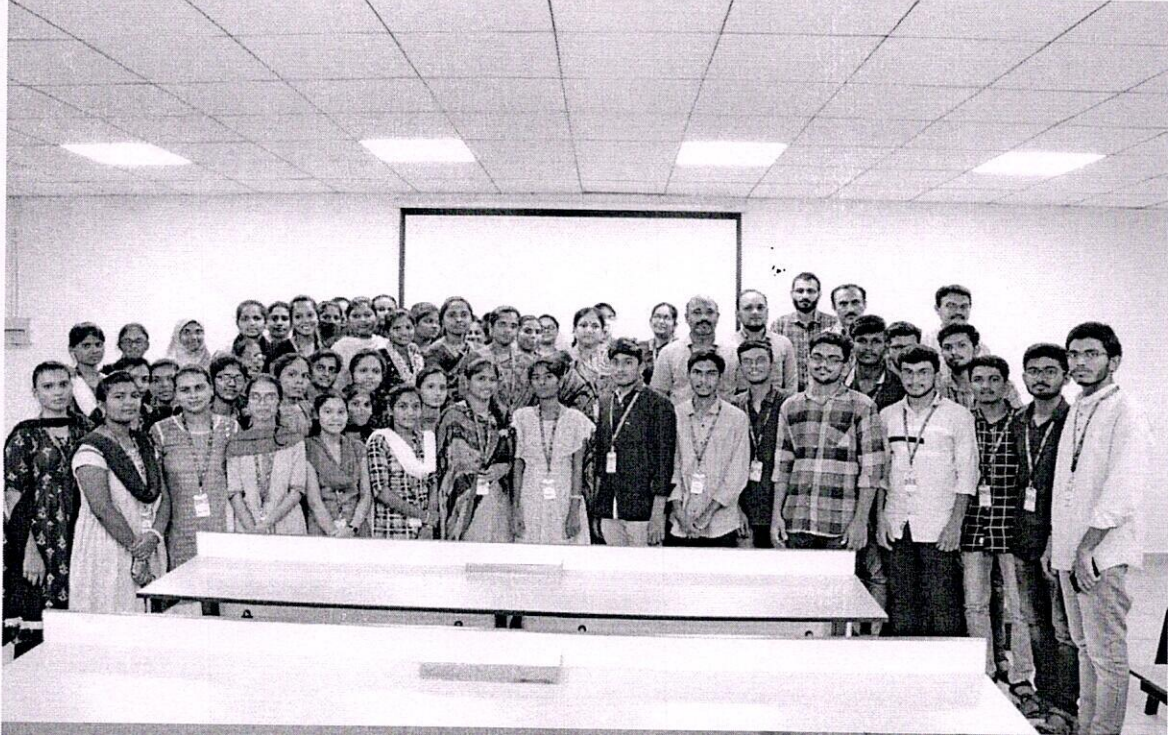
Cordinators

Dr.S.L.Prathap Reddy
Assoc.Professor, Dept of ECE

Resource Persons

Sri.S.Khaja Khizar

Dr. G. Hemalatha H.O.D	Dr. V.S.S. Murthy Principal	Dr. Kandula Chandra Obul Reddy Managing Director	Smt. K. Rajeswari Correspondent Secretary, Tresurer	Sri K. Madan Mohan Reddy Vice-Chairman	Sri K. Raja Mohan Reddy Chairman
---------------------------	--------------------------------	---	---	---	-------------------------------------



S. S. Reddy
Coordinator(s)



/ksrmce.ac.in

Follow Us:

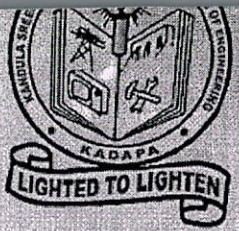


/ksrmceofficial

G. H. HOD
HOD
Professor & H.O.D.

Department of E.C.E.

K.S.R.M. College of Engineering
KADAPA - 516 093



K.S.R.M. COLLEGE OF ENGINEERING

(UGC - Autonomous)

Kadapa, Andhra Pradesh, India - 516003

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.



KSNR
lives on..

Certificate of Participation

This is to certify that

Mr/Ms D. SWETHA **with**

Roll.No. 209Y1A0441

**has attended the Certification course on "C++ Language" from
09-04-22 to 24-04-22 organized by Dept. of Electronics and
Communication Engineering**

Dr. G. Hemalatha
HOD, ECE

Prof V S S Murthy
Principal



K.S.R.M. COLLEGE OF ENGINEERING

(UGC - Autonomous)

Kadapa, Andhra Pradesh, India - 516003

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.

KSNR
lives on.

Certificate of Participation

This is to certify that

Mr/Ms B. KALYANI with

Roll.No. 20971A0409

has attended the Certification course on "C++ Language" from
09-04-22 to 24-04-22 organized by Dept. of Electronics and
Communication Engineering

Dr. G. Hemalatha
HOD, ECE

Prof V S S Murthy
Principal



K.S.R.M. COLLEGE OF ENGINEERING

(UGC - Autonomous)

Kadapa, Andhra Pradesh, India - 516003

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.



KSNR
lives on..

Certificate of Participation

This is to certify that

Mr/Ms V. Baby with

Roll.No. 219V5A0418

has attended the Certification course on "C++ Language" from
09-04-22 to 24-04-22 organized by Dept. of Electronics and
Communication Engineering

Dr. G. Hemalatha
HOD, ECE

Prof V S S Murthy
Principal

Feedback form on Certificate Course

C++ Language(09-04-2022 to 24-04-2022)

* Required

1. Roll Number *

2. Name of the Student *

3. B.Tech Semester *

Mark only one oval.

☐ I SEM

☐ II SEM

☐ III SEM

☐ IV SEM

☐ V SEM

☐ VI SEM

☐ VII SEM

☐ VIII SEM

4. Branch *

Mark only one oval.

☐ Civil Engineering

☐ EEE

☐ ME

☐ ECE

☐ CSE

☐ AI&ML

5. Email ID *

6. Is the course content meet your expectation. *

Mark only one oval.

☐ Yes

☐ No

7. Is the lecture sequence well planned. *

Mark only one oval.

☐ Strongly disagree

☐ Disagree

☐ Neutral

☐ Agree

☐ Strongly agree

8. The contents of the course is explained with examples. *

Mark only one oval.

- ☐ Strongly disagree
☐ Disagree
☐ Neutral
☐ Agree
☐ Strongly Agree

9. Is the level of course high. *

Mark only one oval.

- ☐ Strongly disagree
☐ Disagree
☐ Neutral
☐ Agree
☐ Strongly Agree

10. Is the course exposed you to the new knowledge and practice. *

Mark only one oval.

- ☐ Strongly disagree
☐ Disagree
☐ Neutral
☐ Agree
☐ Strongly Agree

11. Is the lecture clear and easy to understand. *

Mark only one oval.

☐ Strongly disagree

☐ Disagree

☐ Neutral

☐ Agree

☐ Strongly agree

12. Rate the value of the course increasing your skills. *

Mark only one oval.

☐ Strongly disagree

☐ Disagree

☐ Neutral

☐ Agree

☐ Strongly Agree

13. Any suggestions

This content is neither created nor endorsed by Google.

Google Forms



K.S.R.M. COLLEGE OF ENGINEERING

(UGC - AUTONOMOUS)

Kadapa, Andhra Pradesh, India - 516003

Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu.

Department of Electronics and Communication Engineering

Feedback Form

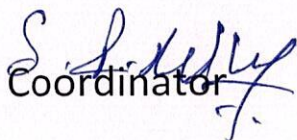
S.No.	Email address	Name of the student	Year & Semester	Branch	Roll Num	Is the course content met your expectation	Is the lecture sequence well planned	The contents of the course is explained with examples	Is the level of course high	Is the course exposed you to the new knowledge and practices	Is the lecturer clear and easy to understand	Rate the value of course in increasing your skills	Any issues
1	209Y1A0401@ksr.mce.ac.in	A. HARSHA VARDHAN	B.Tech IVsem	ECE	209Y1A0401	Yes	Yes	Agree	Agree	Strongly agree	4	5	Nothing
2	209Y1A0402@ksr.mce.ac.in	AKKULAYAPALLE SIDDIQ	B.Tech IVsem	ECE	209Y1A0402	Yes	Yes	Agree	Agree	Strongly agree	5	5	Nothing
3	209Y1A0405@ksr.mce.ac.in	AYALURI SHASHIKALA	B.Tech IVsem	ECE	209Y1A0405	Yes	Yes	Agree	Agree	Strongly agree	4	5	Good
4	209Y1A0409@ksr.mce.ac.in	BANDI KALYANI	B.Tech IVsem	ECE	209Y1A0409	Yes	Yes	Agree	Agree	Strongly agree	5	5	nothing
5	209Y1A0412@ksr.mce.ac.in	BANNE NARESH	B.Tech IVsem	ECE	209Y1A0412	Yes	Yes	Agree	Agree	Strongly agree	5	5	Good
6	209Y1A0417@ksr.mce.ac.in	BOGEM SANGEETHA	B.Tech IVsem	ECE	209Y1A0417	Yes	Yes	Agree	Agree	Strongly agree	4	5	very good
7	209Y1A0419@ksr.mce.ac.in	BOYA SUDHEER	B.Tech IVsem	ECE	209Y1A0419	Yes	Yes	Strongly agree	Agree	Strongly agree	4	3	Nothing

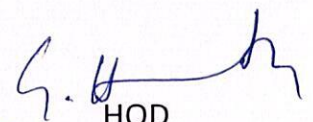
8	209Y1A0421@ksr.mce.ac.in	CHABUKSAWAR SHAIK ABDULLAH	B.Tech IVsem	ECE	209Y1A042 1	Yes	Yes	agree	Agree	Strongly agree	4	4	no
9	209Y1A0422@ksr.mce.ac.in	CHAKALI PRABHAKAR	B.Tech IVsem	ECE	209Y1A042 2	Yes	Yes	Strongly agree	Agree	Strongly agree	5	5	Nothing
10	209Y1A0425@ksr.mce.ac.in	CHAMANCHI VENKATA LOHITH	B.Tech IVsem	ECE	209Y1A042 5	Yes	Yes	Strongly agree	Agree	Strongly agree	5	5	Good
11	209Y1A0427@ksr.mce.ac.in	CHAVVA RAJESHWARA	B.Tech IVsem	ECE	209Y1A042 7	Yes	Yes	Agree	Agree	Strongly agree	5	4	Good
12	209Y1A0430@ksr.mce.ac.in	CHINNAKOTLA MADHAVI	B.Tech IVsem	ECE	209Y1A043 0	Yes	Yes	agree	Agree	Strongly agree	5	5	Good
13	209Y1A0433@ksr.mce.ac.in	CHINTHA HARIPRIYA	B.Tech IVsem	ECE	209Y1A043 3	Yes	Yes	agree	Agree	Strongly agree	3	5	Good
14	209Y1A0437@ksr.mce.ac.in	DANDU SIVIAIAH	B.Tech IVsem	ECE	179Y1A043 7	Yes	Yes	agree	Agree	Strongly agree	5	4	very good
15	209Y1A0441@ksr.mce.ac.in	DEVAGANI SWETHA	B.Tech IVsem	ECE	209Y1A044 1	Yes	Yes	agree	Agree	Strongly agree	4	4	very good
16	209Y1A0445@ksr.mce.ac.in	DUDEKULA JAYA CHANDRA	B.Tech IVsem	ECE	209Y1A044 5	Yes	Yes	agree	Agree	Strongly agree	5	4	very good
17	209Y1A0449@ksr.mce.ac.in	GAMPA VAMSI	B.Tech IVsem	ECE	209Y1A044 9	Yes	Yes	agree	Agree	Strongly agree	3	5	no
18	209Y1A0454@ksr.mce.ac.in	GODLAVETI SAI PRATHAP	B.Tech IVsem	ECE	209Y1A045 4	Yes	Yes	agree	Agree	Strongly agree	4	5	nithing
19	209Y1A0457@ksr.mce.ac.in	GORLA BHARGAVI	B.Tech IVsem	ECE	209Y1A045 7	Yes	Yes	Strongly agree	Agree	Strongly agree	4	5	Good
20	209Y1A0461@ksr.mce.ac.in	GUDISENAPALLE HARI KRISHNA	B.Tech IVsem	ECE	209Y1A046 1	Yes	Yes	Strongly agree	Agree	Strongly agree	4	4	Good
21	209Y1A0467@ksr.mce.ac.in	KAMMARI ANITHA	B.Tech IVsem	ECE	209Y1A046 7	Yes	Yes	Strongly agree	Agree	Strongly agree	4	3	Good
22	209Y1A0470@ksr.mce.ac.in	KARROLLA GANESH	B.Tech IVsem	ECE	209Y1A047 0	Yes	Yes	agree	Agree	Strongly agree	4	4	Good
23	209Y1A0474@ksr.mce.ac.in	KOVVURU LAVANYA	B.Tech IVsem	ECE	209Y1A047 4	Yes	Yes	agree	Agree	Strongly agree	5	4	Good

24	209Y1A0478@ksr.mce.ac.in	LAKSHMIGARI POOJITHA	B.Tech IVsem	ECE	209Y1A0478	Yes	Yes	Strongly agree	Agree	Strongly agree	5	4	Good
25	209Y1A0482@ksr.mce.ac.in	MADDEPALLI SUJITHA RANI	B.Tech IVsem	ECE	209Y1A0482	Yes	Yes	agree	Agree	Strongly agree	5	5	Good
26	209Y1A0485@ksr.mce.ac.in	MALLISETTY KRISHNA SAI GOUD	B.Tech IVsem	ECE	209Y1A0485	Yes	Yes	agree	Agree	Strongly agree	5	5	Nothing
27	209Y1A0490@ksr.mce.ac.in	MEKALA GIRINATH REDDY	B.Tech IVsem	ECE	209Y1A0490	Yes	Yes	agree	Agree	Strongly agree	5	5	no
28	209Y1A0493@ksr.mce.ac.in	MURABOYANA RAJIV	B.Tech IVsem	ECE	209Y1A0493	Yes	Yes	agree	Agree	Strongly agree	3	4	no
29	209Y1A0497@ksr.mce.ac.in	NAGURU INDRA KALYAN REDDY	B.Tech IVsem	ECE	209Y1A0497	Yes	Yes	Strongly agree	Agree	Strongly agree	3	4	no
30	209Y1A04A0@ksr.mce.ac.in	NANDYALA SUBBARAYUDU	B.Tech IVsem	ECE	209Y1A04A0	Yes	Yes	Strongly agree	Agree	Strongly agree		5	no
31	209Y1A04A3@ksr.mce.ac.in	NEELOLLA HARATHI	B.Tech IVsem	ECE	209Y1A04A3	Yes	Yes	Strongly agree	Agree	Strongly agree	5	4	nothing
32	209Y1A04A8@ksr.mce.ac.in	CHANDRA SEKHAR	B.Tech IVsem	ECE	209Y1A04A8	Yes	Yes	agree	Agree	Strongly agree	5	5	Nothing
33	209Y1A04B2@ksr.mce.ac.in	PENDEM USHA RANI	B.Tech IVsem	ECE	209Y1A04B2	Yes	Yes	agree	Agree	Strongly agree	5	4	no
36	209Y1A04B6@ksr.mce.ac.in	PULLALAREVUA GIRINATH REDDY	B.Tech IVsem	ECE	209Y1A04B6	Yes	Yes	agree	Agree	Strongly agree	5	5	Good
38	209Y1A04C0@ksr.mce.ac.in	S ARSHIYA	B.Tech IVsem	ECE	209Y1A04C0	Yes	Yes	Strongly agree	Agree	Strongly agree	5	5	Good
39	209Y1A04C2@ksr.mce.ac.in	SANA YASHWANTH	B.Tech IVsem	ECE	209Y1A04C2	Yes	Yes	Strongly agree	Agree	Strongly agree	5	5	Good
40	209Y1A04C5@ksr.mce.ac.in	SHAIK BEEBI AYESHA SIDDIKA	B.Tech IVsem	ECE	209Y1A04C5	Yes	Yes	Strongly agree	Agree	Strongly agree	5	5	Good
41	209Y1A04C8@ksr.mce.ac.in	MOHAMMED FAIZAN	B.Tech IVsem	ECE	209Y1A04C8	Yes	Yes	agree	Agree	Strongly agree	4	4	Good

42	209Y1A04D1@ksr.mce.ac.in	SHAIK SAHEEL	B.Tech IVsem	ECE	209Y1A04D 1	Yes	Yes	agree	Agree	Strongly agree	4	5	Good
43	209Y1A04D4@ksr.mce.ac.in	SINGANAMALA VENKATA SAI	B.Tech IVsem	ECE	209Y1A04D 4	Yes	Yes	agree	Agree	Strongly agree	4	5	Good
44	209Y1A04D6@ksr.mce.ac.in	SUDHA SUMANTH REDDY	B.Tech IVsem	ECE	209Y1A04D 6	Yes	Yes	agree	Agree	Strongly agree	3	5	Good
45	209Y1A04E1@ksr.mce.ac.in	TARIGONDA REDDIKALA	B.Tech IVsem	ECE	209Y1A04E 1	Yes	Yes	agree	Agree	Strongly agree	3	5	Nothing
46	209Y1A04E3@ksr.mce.ac.in	TATIGOTLA NITISH KUMAR	B.Tech IVsem	ECE	209Y1A04E 3	Yes	Yes	Strongly agree	Agree	Strongly agree	2	5	Nothing
47	209Y1A04E7@ksr.mce.ac.in	THUNGA KUSUMA	B.Tech IVsem	ECE	209Y1A04E 7	Yes	Yes	agree	Agree	Strongly agree	2	5	very good
48	209Y1A04F1@ksr.mce.ac.in	VAKA NANDINI	B.Tech IVsem	ECE	209Y1A04F 1	Yes	Yes	agree	Agree	Strongly agree	4	5	very good
49	209Y1A04F5@ksr.mce.ac.in	VEERAPURAM SWAPNA	B.Tech IVsem	ECE	209Y1A04F 5	Yes	Yes	Strongly agree	Agree	Strongly agree	5	5	very good
50	209Y1A04F7@ksr.mce.ac.in	YAKASI BHARATH	B.Tech IVsem	ECE	209Y1A04F 7	Yes	Yes	Strongly agree	Agree	Strongly agree	4	5	nothing
51	209Y1A04F9@ksr.mce.ac.in	YELAMPALLE SAIKIRAN REDDY	B.Tech IVsem	ECE	209Y1A04F 9	Yes	Yes	agree	Agree	Strongly agree	4	5	Good
52	219Y5A0402@ksr.mce.ac.in	BODDU PALLAVI	B.Tech IVsem	ECE	219Y5A040 2	Yes	Yes	agree	Agree	Strongly agree	4	5	Good
53	219Y5A0403@ksr.mce.ac.in	BOGGULA PARIMALA	B.Tech IVsem	ECE	219Y5A040 3	Yes	Yes	agree	Agree	Strongly agree	4	5	nothing
54	219Y5A0405@ksr.mce.ac.in	G KAVYA	B.Tech IVsem	ECE	219Y5A040 5	Yes	Yes	agree	Agree	Strongly agree	4	5	nothing
55	219Y5A0406@ksr.mce.ac.in	GANJI KUNTA MADHU KUMAR	B.Tech IVsem	ECE	219Y5A040 6	Yes	Yes	agree	Agree	Strongly agree	4	5	nothing
56	219Y5A0408@ksr.mce.ac.in	GORLA MAINA	B.Tech IVsem	ECE	219Y5A040 8	Yes	Yes	agree	Agree	Strongly agree	4	5	Good

57	219Y5A0409@ksr_mce.ac.in	KARELLA RAGHAVENDRA)	B.Tech IVsem	ECE	219Y5A0409	Yes	Yes	agree	Agree	Strongly agree	5	5	Good
58	219Y5A0411@ksr_mce.ac.in	MEESALA MOUNIKA	B.Tech IVsem	ECE	219Y5A0411	Yes	Yes	agree	Agree	Strongly agree	5	5	very good
59	219Y5A0412@ksr_mce.ac.in	MUKKA MALLA SHIVA	B.Tech IVsem	ECE	219Y5A0412	Yes	Yes	Strongly agree	Agree	Strongly agree	5	5	very good
60	219Y5A0413@ksr_mce.ac.in	PASUPULETI NAGENDRA	B.Tech IVsem	ECE	219Y5A0413	Yes	Yes	Strongly agree	Agree	Strongly agree	5	5	nothing
61	219Y5A0414@ksr_mce.ac.in	PATAN FARUQ KHAN	B.Tech IVsem	ECE	219Y5A0414	Yes	Yes	agree	Agree	Strongly agree	5	5	no
62	219Y5A0415@ksr_mce.ac.in	SANE ARUN KUMAR	B.Tech IVsem	ECE	219Y5A0415	Yes	Yes	agree	Agree	Strongly agree	5	5	Nothing
63	219Y5A0416@ksr_mce.ac.in	SANGALA CHARAN KUMAR REDDY	B.Tech IVsem	ECE	219Y5A0416	Yes	Yes	agree	Agree	Strongly agree	3	4	no
64	219Y5A0417@ksr_mce.ac.in	SHAIK MAHAMMAD SHARIF	B.Tech IVsem	ECE	219Y5A0417	Yes	Yes	agree	Agree	Strongly agree	5	5	nothing
65	219Y5A0418@ksr_mce.ac.in	VANDADI BABY	B.Tech IVsem	ECE	219Y5A0418	Yes	Yes	agree	Agree	Strongly agree	5	5	Good


Coordinator


HOD
Professor & H.O.D.
Department of E.C.E.
K J S O College of Engineering
RAJAHMUNDRAM 526 003

K.S.R.M. COLLEGE OF ENGINEERING (AUTONOMOUS), KADAPA-516003
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
VALUE ADDED/CERTIFICATE COURSE ON
C++ PROGRAMMING FROM 09/04/2022 TO 24/04/2022
AWARD LIST

S.No	Roll Number	Name of the Student	Marks Obtained
1.	209Y1A0401	A. HARSHA VARDHAN	13
2.	209Y1A0402	AKKULAYAPALLE SIDDIQ	12
3.	209Y1A0405	AYALURI SHASHIKALA	15
4.	209Y1A0409	BANDI KALYANI	16
5.	209Y1A0412	BANNE NARESH	13
6.	209Y1A0417	BOGEM SANGEETHA	14
7.	209Y1A0419	BOYA SUDHEER	14
8.	209Y1A0421	CHABUKSAWAR SHAIK ABDULLAH	16
9.	209Y1A0422	CHAKALI PRABHAKAR	10
10.	209Y1A0425	CHAMANCHI VENKATA LOHITH	12
11.	209Y1A0427	CHAVVA RAJESHWARA REDDY	12
12.	209Y1A0430	CHINNAKOTLA MADHAVI	13
13.	209Y1A0433	CHINTHA HARIPRIYA	08
14.	179Y1A0437	DANDU SIVAIAH	09
15.	209Y1A0441	DEVAGANI SWETHA	15
16.	209Y1A0445	DUDEKULA JAYA CHANDRA	15
17.	209Y1A0449	GAMPA VAMSI	16
18.	209Y1A0454	GODLAVETI SAI PRATHAP	13
19.	209Y1A0457	GORLA BHARGAVI	12
20.	209Y1A0461	GUDISENAPALLE HARI KRISHNA	10

21.	209Y1A0467	KAMMARI ANITHA	13
22.	209Y1A0470	KARROLLA GANESH	12
23.	209Y1A0474	KOVVURU LAVANYA	11
24.	209Y1A0478	LAKSHMIGARI POOJITHA	14
25.	209Y1A0482	MADDEPALLI SUJITHA RANI	16
26.	209Y1A0485	MALLISETTY KRISHNA SAI GOUD	16
27.	209Y1A0490	MEKALA GIRINATH REDDY	15
28.	209Y1A0493	MURABOYANA RAJIV	13
29.	209Y1A0497	NAGURU INDRA KALYAN REDDY	10
30.	209Y1A04A0	NANDYALA SUBBARAYUDU	12
31.	209Y1A04A3	NEELOLLA HARATHI	14
32.	209Y1A04A8	PANDLA CHANDRA SEKHAR	12
33.	209Y1A04B2	PENDEM USHA RANI	14
34.	209Y1A04B6	PULLALAREVUA GIRINATH REDDY	15
35.	209Y1A04C0	S ARSHIYA	16
36.	209Y1A04C2	SANA YASHWANTH	10
37.	209Y1A04C5	SHAIK BEEBI AYESHA SIDDIKA	08
38.	209Y1A04C8	SHAIK MOHAMMED FAIZAN	12
39.	209Y1A04D1	SHAIK SAHEEL	13
40.	209Y1A04D4	SINGANAMALA VENKATA SAI	15
41.	209Y1A04D6	SUDHA SUMANTH REDDY	13
42.	209Y1A04E1	TARIGONDA REDDIKALA	11
43.	209Y1A04E3	TATIGOTLA NITISH KUMAR REDDY	12

44.	209Y1A04E7	THUNGA KUSUMA	13
45.	209Y1A04F1	VAKA NANDINI	13
46.	209Y1A04F5	VEERAPURAM SWAPNA	11
47.	209Y1A04F7	YAKASI BHARATH	12
48.	209Y1A04F9	YELAMPALLE SAIKIRAN REDDY	11
49.	219Y5A0402	BODDU PALLAVI	13
50.	219Y5A0403	BOGGULA PARIMALA	12
51.	219Y5A0405	G KAVYA	14
52.	219Y5A0406	GANJI KUNTA MADHU KUMAR	13
53.	219Y5A0408	GORLA MAINA	15
54	219Y5A0409	KARELLA RAGHAVENDRA)	14
55	219Y5A0411	MEESALA MOUNIKA	16
56	219Y5A0412	MUKKA MALLA SHIVA	11
57	219Y5A0413	PASUPULETI NAGENDRA	14
58	219Y5A0414	PATAN FARUQ KHAN	12
59	219Y5A0415	SANE ARUN KUMAR	14
60	219Y5A0416	SANGALA CHARAN KUMAR REDDY	13
61	219Y5A0417	SHAIK MAHAMMAD SHARIF	18
62	219Y5A0418	VANDADI BABY	14

S. S. Reddy
Coordinator

G. J. m
Professor & H.O.D.
Department of E.C.E.
K.S.R.M. College of Engineering
KADAPA - 516 093

K.S.R.M. COLLEGE OF ENGINEERING (AUTONOMOUS), KADAPA-516003
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
VALUE ADDED /CERTIFICATE COURSE ON
C++ PROGRAMMING FROM 09/04/2022 TO 24/04/2022

ASSESSMENT TEST

Roll Number: _____ **Name of the Student:** _____

Time: 20 Min

(Objective Questions)

Max.Marks: 20

Note: Answer the following Questions and each question carries **one** mark.

1. Which of the following is used for implementing the late binding? []
a)Operator Functions
b)Virtual Functions
c)Constant Functions
d)All of above
2. Which one of the following cannot be used with the virtual keyword? []
a)Destructor
b)Member function
c)Constructor
d)None of the above
3. Which of the following statement is not true about C++? []
a)A class cannot have the private members
b)Members of a class are public by default
c)A structure can have the member functions
d)All of the above
4. Which of the following is the correct syntax to add the header file in the C++ program? []
a)#include "userdefined.h"
b)#include<userdefined>
c)<include> "userdefined.h"
d)Both A & B
5. Which of the following statements is correct about the friend function in C++ programming language? []
a)A friend function can access the private members of a class
b)A friend function is able to access private members of a class
c)A friend function is able to access the public members of a class
d)All of the above
6. Which of the following statements is correct about the class? []
a)A class is an instance of its object
b)An object is the instance of the data type of that class
c)An object is an instance of its class
d)All of above
7. Which of the following can be used to create an abstract class in the C++ programming language? []
a)By using the pure virtual function in the class
b)By declaring a virtual function in the base class

- c)By declaring the virtual keyword afterward, the class Declaration
d)None of the above
8. Which of the following can be considered as the members that can be inherited but not accessible in any class? []
- a)Protected
b)Public
c)Private
d)None of the above
9. Which of the following is the correct syntax to print the message in C++ language? []
- a)Out <<"Hello world!;
b)cout <<"Hello world!";
c)Cout << Hello world! ;
d)None of the above
10. Which of the following can be considered as the correct syntax for declaring an array of pointers of integers that has a size of 10 in C++? []
- a)int *arr = new int*[10]
b)int *arr = new int[10];
c)int arr = new int[10];
d)int **arr = new int*[10];
11. Which one of the following statements correctly refers to the Delete and Delete[] in C++ programming language? []
- a)The "Delete" is used for deleting the standard objects, while on the other hand, the "Delete[]" is used to delete the pointer objects
b)The "Delete" is a type of keyword, whereas the "Delete[]" is a type of identifier
c)The "Delete" is used for deleting a single standard object, whereas the "Delete[]" is used for deleting an array of the multiple objects
d)Delete is syntactically correct although, if the Delete[] is used, it will obtain an error
12. Which of the following statement is correct about Virtual Inheritance? []
- a)It is a technique to ensure that a private member of a base class can be accessed
b)It is a C++ technique to avoid multiple copies of the base class into the derived or child classes
c)It is a technique to optimize the multiple inheritances
d)It is a technique to avoid the multiple inheritances of the classes
13. Elements of a one-dimensional array are numbered as 0,1,2,3,4,5, and so on; these numbers are known as ____ []
- a)Members of Array
b)Index values
c)Subscript of Array
d)Both 2 & 3
14. How many types of elements can an array store? []
- a)Same types of elements
b)Only char types
c)Char and int type
d)All of the above
15. Which of the following can be considered as the object of an array? []

- a)Elements of the Array
- b)Index of an array
- c)Functions of the Array
- d)All of the above

16. Which types of arrays are always considered as linear arrays? []

- a)Multi-dimensional
- b)Single-dimensional
- c)All of above
- d)None of the above

17. What did we call an array of the one-dimensional array? []

- a)Multi-Dimensional array
- b)Single Dimensional array
- c)2D Array (or 2-Dimensional array)
- d)All of above

18. Which one of the following is the correct definition of the “is_array();” function in C++? []

- a)It checks that the specified array of single dimension or not
- b)It checks that the array specified of multi-dimension or not
- c)It checks that the specified variable is of the array or not
- d)All of above

19. In C++, for what purpose the “rank()” is used? []

- a)It returns the maximum number of elements that can be stored in the array
- b)It returns the size of each dimension
- c)It returns the dimension of the specified array
- d)None of the above

20. How many types of the array are there in the C++ programming language? []

- a)In the C++ programming language, there are four types of arrays
- b)In the C++ programming language, there are three types of arrays
- c)In the C++ programming language, there are two types of arrays
- d)All of above

14/20

K.S.R.M. COLLEGE OF ENGINEERING (AUTONOMOUS), KADAPA-516003
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
VALUE ADDED /CERTIFICATE COURSE ON
C++ PROGRAMMING FROM 09/04/2022 TO 24/04/2022

ASSESSMENT TEST

Roll Number: 219Y5A0418 Name of the Student: V. Baby

Time: 20 Min

(Objective Questions)

Max.Marks: 20

Note: Answer the following Questions and each question carries **one** mark.

1. Which of the following is used for implementing the late binding? [b]
a)Operator Functions
b)Virtual Functions
c)Constant Functions
d)All of above
2. Which one of the following cannot be used with the virtual keyword? [c]
a)Destructor
b)Member function
c)Constructor
d)None of the above
3. Which of the following statement is not true about C++? [b]
a)A class cannot have the private members
b)Members of a class are public by default
c)A structure can have the member functions
d)All of the above
4. Which of the following is the correct syntax to add the header file in the C++ program? [d]
a)#include "userdefined.h"
b)#include<userdefined>
c)<include> "userdefined.h"
d)Both A & B
5. Which of the following statements is correct about the friend function in C++ programming language? [d]
a)A friend function can access the private members of a class
b)A friend function is able to access private members of a class
c)A friend function is able to access the public members of a class
d)All of the above
6. Which of the following statements is correct about the class? [b]
a)A class is an instance of its object
b)An object is the instance of the data type of that class
c)An object is an instance of its class
d)All of above
7. Which of the following can be used to create an abstract class in the C++ programming language? [a]
a)By using the pure virtual function in the class
b)By declaring a virtual function in the base class

- c) By declaring the virtual keyword afterward, the class Declaration
d) None of the above

8. Which of the following can be considered as the members that can be inherited but not accessible in any class? [c]

- a) Protected
b) Public
c) Private
d) None of the above

9. Which of the following is the correct syntax to print the message in C++ language? [b]

- a) Out << "Hello world!;
b) cout << "Hello world!";
c) Cout << Hello world! ;
d) None of the above

10. Which of the following can be considered as the correct syntax for declaring an array of pointers of integers that has a size of 10 in C++? [c]

- a) int *arr = new int*[10]
b) int *arr = new int[10];
c) int arr = new int[10];
d) int **arr = new int*[10];

11. Which one of the following statements correctly refers to the Delete and Delete[] in C++ programming language? [c]

- a) The "Delete" is used for deleting the standard objects, while on the other hand, the "Delete[]" is used to delete the pointer objects
b) The "Delete" is a type of keyword, whereas the "Delete[]" is a type of identifier
c) The "Delete" is used for deleting a single standard object, whereas the "Delete[]" is used for deleting an array of the multiple objects
d) Delete is syntactically correct although, if the Delete[] is used, it will obtain an error

12. Which of the following statement is correct about Virtual Inheritance? [b]

- a) It is a technique to ensure that a private member of a base class can be accessed
b) It is a C++ technique to avoid multiple copies of the base class into the derived or child classes
c) It is a technique to optimize the multiple inheritances
d) It is a technique to avoid the multiple inheritances of the classes

13. Elements of a one-dimensional array are numbered as 0,1,2,3,4,5, and so on; these numbers are known as [d]

- a) Members of Array
b) Index values
c) Subscript of Array
d) Both 2 & 3

14. How many types of elements can an array store? [a]

- a) Same types of elements
b) Only char types
c) Char and int type
d) All of the above

15. Which of the following can be considered as the object of an array? [a]

- a) Elements of the Array
- b) Index of an array
- c) Functions of the Array
- d) All of the above

16. Which types of arrays are always considered as linear arrays?

- a) Multi-dimensional
- b) Single-dimensional
- c) All of above
- d) None of the above

17. What did we call an array of the one-dimensional array?

- a) Multi-Dimensional array
- b) Single Dimensional array
- c) 2D Array (or 2-Dimensional array)
- d) All of above

18. Which one of the following is the correct definition of the "is_array();" function in C++?

- a) It checks that the specified array of single dimension or not
- b) It checks that the array specified of multi-dimension or not
- c) It checks that the specified variable is of the array or not
- d) All of above

19. In C++, for what purpose the "rank()" is used?

- a) It returns the maximum number of elements that can be stored in the array
- b) It returns the size of each dimension
- c) It returns the dimension of the specified array
- d) None of the above

20. How many types of the array are there in the C++ programming language?

- a) In the C++ programming language, there are four types of arrays
- b) In the C++ programming language, there are three types of arrays
- c) In the C++ programming language, there are two types of arrays
- d) All of above

[b]

[b]

[c]

[c]

[c]

16/10

K.S.R.M. COLLEGE OF ENGINEERING (AUTONOMOUS), KADAPA-516003
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
VALUE ADDED /CERTIFICATE COURSE ON
C++ PROGRAMMING FROM 09/04/2022 TO 24/04/2022

ASSESSMENT TEST

Roll Number: 2091A0409 Name of the Student: B. Kalyani

Time: 20 Min **(Objective Questions)** **Max.Marks: 20**

Note: Answer the following Questions and each question carries **one** mark.

1. Which of the following is used for implementing the late binding?
a)Operator Functions
b)Virtual Functions
c)Constant Functions
d)All of above
[B]
2. Which one of the following cannot be used with the virtual keyword?
a)Destructor
b)Member function
c)Constructor
d)None of the above
[C]
3. Which of the following statement is not true about C++?
a)A class cannot have the private members
b)Members of a class are public by default
c)A structure can have the member functions
d)All of the above
[C]
4. Which of the following is the correct syntax to add the header file in the C++ program?
a)#include "userdefined.h"
b)#include<userdefined>
c)<include> "userdefined.h"
d)Both A & B
[D]
5. Which of the following statements is correct about the friend function in C++ programming language?
a)A friend function can access the private members of a class
b)A friend function is able to access private members of a class
c)A friend function is able to access the public members of a class
d)All of the above
[B]
6. Which of the following statements is correct about the class?
a)A class is an instance of its object
b)An object is the instance of the data type of that class
c)An object is an instance of its class
d)All of above
[A]
7. Which of the following can be used to create an abstract class in the C++ programming language?
a)By using the pure virtual function in the class
b)By declaring a virtual function in the base class
[A]

- c) By declaring the virtual keyword afterward, the class Declaration
d) None of the above

8. Which of the following can be considered as the members that can be inherited but not accessible in any class?

[C]

- a) Protected
b) Public
c) Private
d) None of the above

9. Which of the following is the correct syntax to print the message in C++ language?

[B]

- a) Out << "Hello world!";
b) cout << "Hello world!";
c) Cout << Hello world! ;
d) None of the above

10. Which of the following can be considered as the correct syntax for declaring an array of pointers of integers that has a size of 10 in C++?

[D]

- a) int *arr = new int*[10]
b) int *arr = new int[10];
c) int arr = new int[10];
d) int **arr = new int*[10];

11. Which one of the following statements correctly refers to the Delete and Delete[] in C++ programming language?

[B]

- a) The "Delete" is used for deleting the standard objects, while on the other hand, the "Delete[]" is used to delete the pointer objects
b) The "Delete" is a type of keyword, whereas the "Delete[]" is a type of identifier
c) The "Delete" is used for deleting a single standard object, whereas the "Delete[]" is used for deleting an array of the multiple objects
d) Delete is syntactically correct although, if the Delete[] is used, it will obtain an error

12. Which of the following statement is correct about Virtual Inheritance?

[B]

- a) It is a technique to ensure that a private member of a base class can be accessed
b) It is a C++ technique to avoid multiple copies of the base class into the derived or child classes

c) It is a technique to optimize the multiple inheritances

d) It is a technique to avoid the multiple inheritances of the classes

13. Elements of a one-dimensional array are numbered as 0,1,2,3,4,5, and so on; these numbers are known as _____

[D]

- a) Members of Array
b) Index values
c) Subscript of Array
d) Both 2 & 3

14. How many types of elements can an array store?

[B]

- a) Same types of elements
b) Only char types
c) Char and int type
d) All of the above

15. Which of the following can be considered as the object of an array?

[a]

- a)Elements of the Array
 - b)Index of an array
 - c)Functions of the Array
 - d)All of the above
16. Which types of arrays are always considered as linear arrays? [B]
- a)Multi-dimensional
 - b)Single-dimensional
 - c)All of above
 - d)None of the above
17. What did we call an array of the one-dimensional array? [C]
- a)Multi-Dimensional array
 - b)Single Dimensional array
 - c)2D Array (or 2-Dimensional array)
 - d)All of above
18. Which one of the following is the correct definition of the "is_array();" function in C++? [D]
- a)It checks that the specified array of single dimension or not
 - b)It checks that the array specified of multi-dimension or not
 - c)It checks that the specified variable is of the array or not
 - d)All of above
19. In C++, for what purpose the "rank()" is used? [C]
- a)It returns the maximum number of elements that can be stored in the array
 - b)It returns the size of each dimension
 - c)It returns the dimension of the specified array
 - d)None of the above
20. How many types of the array are there in the C++ programming language? [C]
- a)In the C++ programming language, there are four types of arrays
 - b)In the C++ programming language, there are three types of arrays
 - c)In the C++ programming language, there are two types of arrays
 - d)All of above

15/20

K.S.R.M. COLLEGE OF ENGINEERING (AUTONOMOUS), KADAPA-516003
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
VALUE ADDED /CERTIFICATE COURSE ON
C++ PROGRAMMING FROM 09/04/2022 TO 24/04/2022

ASSESSMENT TEST

Roll Number: 219415A0408 **Name of the Student:** G. maina

Time: 20 Min

(Objective Questions)

Max.Marks: 20

Note: Answer the following Questions and each question carries **one** mark.

1. Which of the following is used for implementing the late binding?
a)Operator Functions
b)Virtual Functions
c)Constant Functions
d)All of above
[A]
2. Which one of the following cannot be used with the virtual keyword?
a)Destructor
b)Member function
c)Constructor
d)None of the above
[C]
3. Which of the following statement is not true about C++?
a)A class cannot have the private members
b)Members of a class are public by default
c)A structure can have the member functions
d)All of the above
[C]
4. Which of the following is the correct syntax to add the header file in the C++ program?
a)#include "userdefined.h"
b)#include<userdefined>
c)<include> "userdefined.h"
d)Both A & B
[D]
5. Which of the following statements is correct about the friend function in C++ programming language?
a)A friend function can access the private members of a class
b)A friend function is able to access private members of a class
c)A friend function is able to access the public members of a class
d)All of the above
[B]
6. Which of the following statements is correct about the class?
a)A class is an instance of its object
b)An object is the instance of the data type of that class
c)An object is an instance of its class
d)All of above
[A]
7. Which of the following can be used to create an abstract class in the C++ programming language?
a)By using the pure virtual function in the class
b)By declaring a virtual function in the base class
[A]

- 05/21
- c) By declaring the virtual keyword afterward, the class Declaration
d) None of the above
8. Which of the following can be considered as the members that can be inherited but not accessible in any class? [C]
- a) Protected
 - b) Public
 - c) Private
 - d) None of the above
9. Which of the following is the correct syntax to print the message in C++ language? [C]
- a) Out << "Hello world!;
 - b) cout << "Hello world!";
 - c) Cout << Hello world! ;
 - d) None of the above
10. Which of the following can be considered as the correct syntax for declaring an array of pointers of integers that has a size of 10 in C++? [D]
- a) int *arr = new int*[10]
 - b) int *arr = new int[10];
 - c) int arr = new int[10];
 - d) int **arr = new int*[10];
11. Which one of the following statements correctly refers to the Delete and Delete[] in C++ programming language? [D]
- a) The "Delete" is used for deleting the standard objects, while on the other hand, the "Delete[]" is used to delete the pointer objects
 - b) The "Delete" is a type of keyword, whereas the "Delete[]" is a type of identifier
 - c) The "Delete" is used for deleting a single standard object, whereas the "Delete[]" is used for deleting an array of the multiple objects
 - d) Delete is syntactically correct although, if the Delete[] is used, it will obtain an error
12. Which of the following statement is correct about Virtual Inheritance? [B]
- a) It is a technique to ensure that a private member of a base class can be accessed
 - b) It is a C++ technique to avoid multiple copies of the base class into the derived or child classes
 - c) It is a technique to optimize the multiple inheritances
 - d) It is a technique to avoid the multiple inheritances of the classes
13. Elements of a one-dimensional array are numbered as 0,1,2,3,4,5, and so on; these numbers are known as ____ [D]
- a) Members of Array
 - b) Index values
 - c) Subscript of Array
 - d) Both 2 & 3
14. How many types of elements can an array store? [B]
- a) Same types of elements
 - b) Only char types
 - c) Char and int type
 - d) All of the above
15. Which of the following can be considered as the object of an array? [A]

- a) Elements of the Array
- b) Index of an array
- c) Functions of the Array
- d) All of the above

16. Which types of arrays are always considered as linear arrays?

- a) Multi-dimensional
- b) Single-dimensional
- c) All of above
- d) None of the above

[B]

17. What did we call an array of the one-dimensional array?

- a) Multi-Dimensional array
- b) Single Dimensional array
- c) 2D Array (or 2-Dimensional array)
- d) All of above

[C]

18. Which one of the following is the correct definition of the "is_array();" function in C++?

- a) It checks that the specified array of single dimension or not
- b) It checks that the array specified of multi-dimension or not
- c) It checks that the specified variable is of the array or not
- d) All of above

[D]

19. In C++, for what purpose the "rank()" is used?

- a) It returns the maximum number of elements that can be stored in the array
- b) It returns the size of each dimension
- c) It returns the dimension of the specified array
- d) None of the above

[C]

20. How many types of the array are there in the C++ programming language?

- a) In the C++ programming language, there are four types of arrays
- b) In the C++ programming language, there are three types of arrays
- c) In the C++ programming language, there are two types of arrays
- d) All of above

[C]

C++ Language

Table of contents

Table of contents	3
Basics of C++	4
Structure of a program	4
Variables, Data Types	8
Constants	14
Operators.....	18
Basic Input/Output	26
Control Structures	31
Control Structures	31
Functions (I).....	38
Functions (II).....	44
Compound data types	51
Arrays.....	51
Character Sequences	58
Pointers.....	60
Dynamic Memory	71
Data structures.....	74
Other Data Types.....	79
Object Oriented Programming	83
Classes (I)	83
Classes (II)	92
Friendship and inheritance	97
Polymorphism	104
Advanced concepts	110
Templates	110
Namespaces	117
Exceptions	120
Type Casting.....	124
Preprocessor directives	130

C++ Standard Library.....	135
Input/Output with files.....	135

Basics of C++

Structure of a program

Probably the best way to start learning a programming language is by writing a program. Therefore, here is our first program:

```
// my first program in C++                                Hello World!

#include <iostream>
using namespace std;

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

The first panel shows the source code for our first program. The second one shows the result of the program once compiled and executed. The way to edit and compile a program depends on the compiler you are using. Depending on whether it has a Development Interface or not and on its version. Consult the compilers section and the manual or help included with your compiler if you have doubts on how to compile a C++ console program.

The previous program is the typical program that programmer apprentices write for the first time, and its result is the printing on screen of the "Hello World!" sentence. It is one of the simplest programs that can be written in C++, but it already contains the fundamental components that every C++ program has. We are going to look line by line at the code we have just written:

// my first program in C++

This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program. The programmer can use them to include short explanations or observations within the source code itself. In this case, the line is a brief description of what our program is.

#include <iostream>

Lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor. In this case the directive `#include <iostream>` tells the preprocessor to include the `iostream` standard file. This specific file (`iostream`) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.

using namespace std;

All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name `std`. So in order to access its functionality we declare with this expression that we will be using these entities. This line is very frequent in C++ programs that use the standard library, and in fact it will be included in most of the source codes included in these tutorials.

int main ()

This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs start their execution, independently of its location within the source code. It does not matter whether there are other functions with other names defined before or after it - the instructions contained within this function's definition will always be the first ones to be executed in any C++ program. For that same reason, it is essential that all C++ programs have a `main` function.

The word `main` is followed in the code by a pair of parentheses (`()`). That is because it is a function declaration: In C++, what differentiates a function declaration from other types of expressions are these parentheses that follow its name. Optionally, these parentheses may enclose a list of parameters within them.

Right after these parentheses we can find the body of the main function enclosed in braces (`{}`). What is contained within these braces is what the function does when it is executed.


```
cout << "Hello World!";
```

This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect. In fact, this statement performs the only action that generates a visible effect in our first program.

`cout` represents the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case the `Hello World` sequence of characters) into the standard output stream (which usually is the screen).

`cout` is declared in the `iostream` standard file within the `std` namespace, so that's why we needed to include that specific file and to declare that we were going to use this specific namespace earlier in our code.

Notice that the statement ends with a semicolon character (;). This character is used to mark the end of the statement and in fact it must be included at the end of all expression statements in all C++ programs (one of the most common syntax errors is indeed to forget to include some semicolon after a statement).

```
return 0;
```

The return statement causes the main function to finish. `return` may be followed by a return code (in our example is followed by the return code 0). A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution. This is the most usual way to end a C++ console program.

You may have noticed that not all the lines of this program perform actions when the code is executed. There were lines containing only comments (those beginning by //). There were lines with directives for the compiler's preprocessor (those beginning by #). Then there were lines that began the declaration of a function (in this case, the main function) and, finally lines with statements (like the insertion into `cout`), which were all included within the block delimited by the braces ({}) of the main function.

The program has been structured in different lines in order to be more readable, but in C++, we do not have strict rules on how to separate instructions in different lines. For example, instead of

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

We could have written:

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

All in just one line and this would have had exactly the same meaning as the previous code.

In C++, the separation between statements is specified with an ending semicolon (;) at the end of each one, so the separation in different code lines does not matter at all for this purpose. We can write many statements per line or write a single statement that takes many code lines. The division of code in different lines serves only to make it more legible and schematic for the humans that may read it.

Let us add an additional instruction to our first program:


```
// my second program in C++                                Hello World! I'm a C++ program

#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello World! ";
    cout << "I'm a C++ program";
    return 0;
}
```

In this case, we performed two insertions into cout in two different statements. Once again, the separation in different lines of code has been done just to give greater readability to the program, since main could have been perfectly valid defined this way:

```
int main () { cout << " Hello World! "; cout << " I'm a C++ program "; return 0; }
```

We were also free to divide the code into more lines if we considered it more convenient:

```
int main ()
{
    cout <<
        "Hello World!";
    cout
        << "I'm a C++ program";
    return 0;
}
```

And the result would again have been exactly the same as in the previous examples.

Preprocessor directives (those that begin by #) are out of this general rule since they are not statements. They are lines read and processed by the preprocessor and do not produce any code by themselves. Preprocessor directives must be specified in their own line and do not have to end with a semicolon (;).

Comments

Comments are parts of the source code disregarded by the compiler. They simply do nothing. Their purpose is only to allow the programmer to insert notes or descriptions embedded within the source code.

C++ supports two ways to insert comments:

```
// line comment
/* block comment */
```

The first of them, known as line comment, discards everything from where the pair of slash signs (//) is found up to the end of that same line. The second one, known as block comment, discards everything between the /* characters and the first appearance of the */ characters, with the possibility of including more than one line. We are going to add comments to our second program:


```
/* my second program in C++          Hello World! I'm a C++ program
   with more comments */

#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello World! ";        // prints Hello
    World!
    cout << "I'm a C++ program"; // prints I'm a
    C++ program
    return 0;
}
```

If you include comments within the source code of your programs without using the comment characters combinations `//`, `/*` or `*/`, the compiler will take them as if they were C++ expressions, most likely causing one or several error messages when you compile it.

Variables. Data Types.

The usefulness of the "Hello World" programs shown in the previous section is quite questionable. We had to write several lines of code, compile them, and then execute the resulting program just to obtain a simple sentence written on the screen as result. It certainly would have been much faster to type the output sentence by ourselves. However, programming is not limited only to printing simple texts on the screen. In order to go a little further on and to become able to write programs that perform useful tasks that really save us work we need to introduce the concept of variable.

Let us think that I ask you to retain the number 5 in your mental memory, and then I ask you to memorize also the number 2 at the same time. You have just stored two different values in your memory. Now, if I ask you to add 1 to the first number I said, you should be retaining the numbers 6 (that is $5+1$) and 2 in your memory. Values that we could now for example subtract and obtain 4 as result.

The whole process that you have just done with your mental memory is a simile of what a computer can do with two variables. The same process can be expressed in C++ with the following instruction set:

```
a = 5;
b = 2;
a = a + 1;
result = a - b;
```

Obviously, this is a very simple example since we have only used two small integer values, but consider that your computer can store millions of numbers like these at the same time and conduct sophisticated mathematical operations with them.

Therefore, we can define a variable as a portion of memory to store a determined value.

Each variable needs an identifier that distinguishes it from the others, for example, in the previous code the variable identifiers were `a`, `b` and `result`, but we could have called the variables any names we wanted to invent, as long as they were valid identifiers.

Identifiers

A valid identifier is a sequence of one or more letters, digits or underscore characters (`_`). Neither spaces nor punctuation marks or symbols can be part of an identifier. Only letters, digits and single underscore characters are valid. In addition, variable identifiers always have to begin with a letter. They can also begin with an underline character (`_`), but in some cases these may be reserved for compiler specific keywords or external identifiers, as well as identifiers containing two successive underscore characters anywhere. In no case they can begin with a digit.

Another rule that you have to consider when inventing your own identifiers is that they cannot match any keyword of the C++ language nor your compiler's specific ones, which are *reserved keywords*. The standard reserved keywords are:

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

Additionally, alternative representations for some operators cannot be used as identifiers since they are reserved words under some circumstances:

```
and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq
```


Your compiler may also include some additional specific reserved keywords.

Very important: The C++ language is a "case sensitive" language. That means that an identifier written in capital letters is not equivalent to another one with the same name but written in small letters. Thus, for example, the `RESULT` variable is not the same as the `result` variable or the `Result` variable. These are three different variable identifiers.

Fundamental data types

When programming, we store the variables in our computer's memory, but the computer has to know what kind of data we want to store in them, since it is not going to occupy the same amount of memory to store a simple number than to store a single letter or a large number, and they are not going to be interpreted the same way.

The memory in our computers is organized in bytes. A byte is the minimum amount of memory that we can manage in C++. A byte can store a relatively small amount of data: one single character or a small integer (generally an integer between 0 and 255). In addition, the computer can manipulate more complex data types that come from grouping several bytes, such as long numbers or non-integer numbers.

Next you have a summary of the basic fundamental data types in C++, as well as the range of values that can be represented with each one:

Name	Description	Size*	Range*
<code>char</code>	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
<code>short int</code> (<code>short</code>)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
<code>int</code>	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
<code>long int</code> (<code>long</code>)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
<code>bool</code>	Boolean value. It can take one of two values: true or false.	1byte	true or false
<code>float</code>	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
<code>double</code>	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
<code>long double</code>	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
<code>wchar_t</code>	Wide character.	2 or 4 bytes	1 wide character

* The values of the columns **Size** and **Range** depend on the system the program is compiled for. The values shown above are those found on most 32-bit systems. But for other systems, the general specification is that `int` has the natural size suggested by the system architecture (one "word") and the four integer types `char`, `short`, `int` and `long` must each one be at least as large as the one preceding it, with `char` being always 1 byte in size. The same applies to the floating point types `float`, `double` and `long double`, where each one must provide at least as much precision as the preceding one.

Declaration of variables

In order to use a variable in C++, we must first declare it specifying which data type we want it to be. The syntax to declare a new variable is to write the specifier of the desired data type (like `int`, `bool`, `float`...) followed by a valid variable identifier. For example:


```
int a;  
float mynumber;
```

These are two valid declarations of variables. The first one declares a variable of type `int` with the identifier `a`. The second one declares a variable of type `float` with the identifier `mynumber`. Once declared, the variables `a` and `mynumber` can be used within the rest of their scope in the program.

If you are going to declare more than one variable of the same type, you can declare all of them in a single statement by separating their identifiers with commas. For example:

```
int a, b, c;
```

This declares three variables (`a`, `b` and `c`), all of them of type `int`, and has exactly the same meaning as:

```
int a;  
int b;  
int c;
```

The integer data types `char`, `short`, `long` and `int` can be either signed or unsigned depending on the range of numbers needed to be represented. Signed types can represent both positive and negative values, whereas unsigned types can only represent positive values (and zero). This can be specified by using either the specifier `signed` or the specifier `unsigned` before the type name. For example:

```
unsigned short int NumberOfSisters;  
signed int MyAccountBalance;
```

By default, if we do not specify either `signed` or `unsigned` most compiler settings will assume the type to be signed, therefore instead of the second declaration above we could have written:

```
int MyAccountBalance;
```

with exactly the same meaning (with or without the keyword `signed`)

An exception to this general rule is the `char` type, which exists by itself and is considered a different fundamental data type from `signed char` and `unsigned char`, thought to store characters. You should use either `signed` or `unsigned` if you intend to store numerical values in a `char`-sized variable.

`short` and `long` can be used alone as type specifiers. In this case, they refer to their respective integer fundamental types: `short` is equivalent to `short int` and `long` is equivalent to `long int`. The following two variable declarations are equivalent:

```
short Year;  
short int Year;
```

Finally, `signed` and `unsigned` may also be used as standalone type specifiers, meaning the same as `signed int` and `unsigned int` respectively. The following two declarations are equivalent:

```
unsigned NextYear;  
unsigned int NextYear;
```

To see what variable declarations look like in action within a program, we are going to see the C++ code of the example about your mental memory proposed at the beginning of this section:


```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
    // declaring variables:
    int a, b;
    int result;

    // process:
    a = 5;
    b = 2;
    a = a + 1;
    result = a - b;

    // print out the result:
    cout << result;

    // terminate the program:
    return 0;
}
```

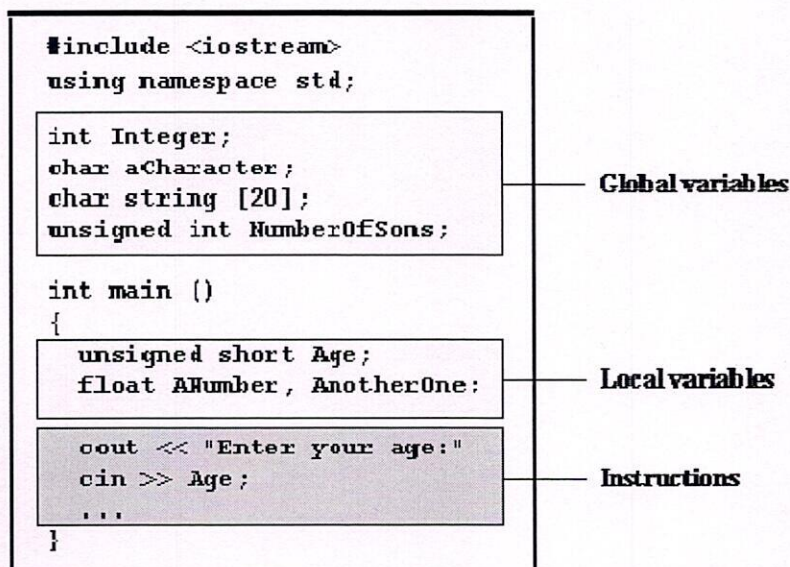
4

Do not worry if something else than the variable declarations themselves looks a bit strange to you. You will see the rest in detail in coming sections.

Scope of variables

All the variables that we intend to use in a program must have been declared with its type specifier in an earlier point in the code, like we did in the previous code at the beginning of the body of the function `main` when we declared that `a`, `b`, and `result` were of type `int`.

A variable can be either of global or local scope. A global variable is a variable declared in the main body of the source code, outside all functions, while a local variable is one declared within the body of a function or a block.



Global variables can be referred from anywhere in the code, even inside functions, whenever it is after its declaration.

The scope of local variables is limited to the block enclosed in braces (`{}`) where they are declared. For example, if they are declared at the beginning of the body of a function (like in function `main`) their scope is between its declaration point and the end of that function. In the example above, this means that if another function existed in addition to `main`, the local variables declared in `main` could not be accessed from the other function and vice versa.

Initialization of variables

When declaring a regular local variable, its value is by default undetermined. But you may want a variable to store a concrete value at the same moment that it is declared. In order to do that, you can initialize the variable. There are two ways to do this in C++:

The first one, known as c-like, is done by appending an equal sign followed by the value to which the variable will be initialized:

```
type identifier = initial_value ;
```

For example, if we want to declare an `int` variable called `a` initialized with a value of 0 at the moment in which it is declared, we could write:

```
int a = 0;
```

The other way to initialize variables, known as constructor initialization, is done by enclosing the initial value between parentheses (`()`):

```
type identifier (initial_value) ;
```

For example:

```
int a (0);
```

Both ways of initializing variables are valid and equivalent in C++.

```
// initialization of variables
#include <iostream>
using namespace std;

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

    a = a + 3;
    result = a - b;
    cout << result;

    return 0;
}
```

6

Introduction to strings

Variables that can store non-numerical values that are longer than one single character are known as strings.

The C++ language library provides support for strings through the standard `string` class. This is not a fundamental type, but it behaves in a similar way as fundamental types do in its most basic usage.

A first difference with fundamental data types is that in order to declare and use objects (variables) of this type we need to include an additional header file in our source code: `<string>` and have access to the `std` namespace (which we already had in all our previous programs thanks to the `using namespace std` statement).

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
    string mystring = "This is a string";
    cout << mystring;
    return 0;
}
```

As you may see in the previous example, strings can be initialized with any valid string literal just like numerical type variables can be initialized to any valid numerical literal. Both initialization formats are valid with strings:

```
string mystring = "This is a string";
string mystring ("This is a string");
```

Strings can also perform all the other basic operations that fundamental data types can, like being declared without an initial value and being assigned values during execution:

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
    string mystring;
    mystring = "This is the initial string content";
    cout << mystring << endl;
    mystring = "This is a different string content";
    cout << mystring << endl;
    return 0;
}
```

For more details on C++ strings, you can have a look at the [string class reference](#).

Constants

Constants are expressions with a fixed value.

Literals

Literals are used to express particular values within the source code of a program. We have already used these previously to give concrete values to variables or to express messages we wanted our programs to print out, for example, when we wrote:

```
a = 5;
```

the 5 in this piece of code was a literal constant.

Literal constants can be divided in Integer Numerals, Floating-Point Numerals, Characters, Strings and Boolean Values.

Integer Numerals

```
1776  
707  
-273
```

They are numerical constants that identify integer decimal values. Notice that to express a numerical constant we do not have to write quotes (") nor any special character. There is no doubt that it is a constant: whenever we write 1776 in a program, we will be referring to the value 1776.

In addition to decimal numbers (those that all of us are used to use every day) C++ allows the use as literal constants of octal numbers (base 8) and hexadecimal numbers (base 16). If we want to express an octal number we have to precede it with a 0 (zero character). And in order to express a hexadecimal number we have to precede it with the characters 0x (zero, x). For example, the following literal constants are all equivalent to each other:

```
75          // decimal  
0113        // octal  
0x4b        // hexadecimal
```

All of these represent the same number: 75 (seventy-five) expressed as a base-10 numeral, octal numeral and hexadecimal numeral, respectively.

Literal constants, like variables, are considered to have a specific data type. By default, integer literals are of type `int`. However, we can force them to either be unsigned by appending the `u` character to it, or long by appending `l`:

```
75          // int  
75u         // unsigned int  
75l         // long  
75ul        // unsigned long
```

In both cases, the suffix can be specified using either upper or lowercase letters.

Floating Point Numbers

They express numbers with decimals and/or exponents. They can include either a decimal point, an `e` character (that expresses "by ten at the Xth height", where X is an integer value that follows the `e` character), or both a decimal point and an `e` character:


```
3.14159    // 3.14159
6.02e23    // 6.02 x 10^23
1.6e-19    // 1.6 x 10^-19
3.0        // 3.0
```

These are four valid numbers with decimals expressed in C++. The first number is PI, the second one is the number of Avogadro, the third is the electric charge of an electron (an extremely small number) -all of them approximated- and the last one is the number three expressed as a floating-point numeric literal.

The default type for floating point literals is `double`. If you explicitly want to express a `float` or `long double` numerical literal, you can use the `f` or `l` suffixes respectively:

```
3.14159L    // long double
6.02e23f    // float
```

Any of the letters that can be part of a floating-point numerical constant (`e`, `f`, `l`) can be written using either lower or uppercase letters without any difference in their meanings.

Character and string literals

There also exist non-numerical constants, like:

```
'z'
'p'
"Hello world"
"How do you do?"
```

The first two expressions represent single character constants, and the following two represent string literals composed of several characters. Notice that to represent a single character we enclose it between single quotes (`'`) and to express a string (which generally consists of more than one character) we enclose it between double quotes (`"`).

When writing both single character and string literals, it is necessary to put the quotation marks surrounding them to distinguish them from possible variable identifiers or reserved keywords. Notice the difference between these two expressions:

```
x
'x'
```

`x` alone would refer to a variable whose identifier is `x`, whereas `'x'` (enclosed within single quotation marks) would refer to the character constant `'x'`.

Character and string literals have certain peculiarities, like the escape codes. These are special characters that are difficult or impossible to express otherwise in the source code of a program, like newline (`\n`) or tab (`\t`). All of them are preceded by a backslash (`\`). Here you have a list of some of such escape codes:

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

For example:

```
'\n'
'\t'
"Left \t Right"
"one\ntwo\nthree"
```

Additionally, you can express any character by its numerical ASCII code by writing a backslash character (`\`) followed by the ASCII code expressed as an octal (base-8) or hexadecimal (base-16) number. In the first case (octal) the digits must immediately follow the backslash (for example `\23` or `\40`), in the second case (hexadecimal), an `x` character must be written before the digits themselves (for example `\x20` or `\x4A`).

String literals can extend to more than a single line of code by putting a backslash sign (`\`) at the end of each unfinished line.

```
"string expressed in \
two lines"
```

You can also concatenate several string constants separating them by one or several blank spaces, tabulators, newline or any other valid blank character:

```
"this forms" "a single" "string" "of characters"
```

Finally, if we want the string literal to be explicitly made of wide characters (`wchar_t`), instead of narrow characters (`char`), we can precede the constant with the `L` prefix:

```
L"This is a wide character string"
```

Wide characters are used mainly to represent non-English or exotic character sets.

Boolean literals

There are only two valid Boolean values: `true` and `false`. These can be expressed in C++ as values of type `bool` by using the Boolean literals `true` and `false`.

Defined constants (`#define`)

You can define your own names for constants that you use very often without having to resort to memory-consuming variables, simply by using the `#define` preprocessor directive. Its format is:

`#define identifier value`

For example:

```
#define PI 3.14159
#define NEWLINE '\n'
```

This defines two new constants: `PI` and `NEWLINE`. Once they are defined, you can use them in the rest of the code as if they were any other regular constant, for example:

```
// defined constants: calculate circumference 31.4159

#include <iostream>
using namespace std;

#define PI 3.14159
#define NEWLINE '\n'

int main ()
{
    double r=5.0;           // radius
    double circle;

    circle = 2 * PI * r;
    cout << circle;
    cout << NEWLINE;

    return 0;
}
```

In fact the only thing that the compiler preprocessor does when it encounters `#define` directives is to literally replace any occurrence of their identifier (in the previous example, these were `PI` and `NEWLINE`) by the code to which they have been defined (3.14159 and `'\n'` respectively).

The `#define` directive is not a C++ statement but a directive for the preprocessor; therefore it assumes the entire line as the directive and does not require a semicolon (;) at its end. If you append a semicolon character (;) at the end, it will also be appended in all occurrences within the body of the program that the preprocessor replaces.

Declared constants (const)

With the `const` prefix you can declare constants with a specific type in the same way as you would do with a variable:

```
const int pathwidth = 100;
const char tabulator = '\t';
```

Here, `pathwidth` and `tabulator` are two typed constants. They are treated just like regular variables except that their values cannot be modified after their definition.

Operators

Once we know of the existence of variables and constants, we can begin to operate with them. For that purpose, C++ integrates operators. Unlike other languages whose operators are mainly keywords, operators in C++ are mostly made of signs that are not part of the alphabet but are available in all keyboards. This makes C++ code shorter and more international, since it relies less on English words, but requires a little of learning effort in the beginning.

You do not have to memorize all the content of this page. Most details are only provided to serve as a later reference in case you need it.

Assignment (=)

The assignment operator assigns a value to a variable.

```
a = 5;
```

This statement assigns the integer value 5 to the variable `a`. The part at the left of the assignment operator (`=`) is known as the *lvalue* (left value) and the right one as the *rvalue* (right value). The *lvalue* has to be a variable whereas the *rvalue* can be either a constant, a variable, the result of an operation or any combination of these. The most important rule when assigning is the *right-to-left* rule: The assignment operation always takes place from right to left, and never the other way:

```
a = b;
```

This statement assigns to variable `a` (the *lvalue*) the value contained in variable `b` (the *rvalue*). The value that was stored until this moment in `a` is not considered at all in this operation, and in fact that value is lost.

Consider also that we are only assigning the value of `b` to `a` at the moment of the assignment operation. Therefore a later change of `b` will not affect the new value of `a`.

For example, let us have a look at the following code - I have included the evolution of the content stored in the variables as comments:

```
// assignment operator
#include <iostream>
using namespace std;

int main ()
{
    int a, b;           // a:?, b:?
    a = 10;              // a:10, b:?
    b = 4;               // a:10, b:4
    a = b;               // a:4, b:4
    b = 7;               // a:4, b:7

    cout << "a:";
    cout << a;
    cout << " b:";
    cout << b;

    return 0;
}
```

a:4 b:7

This code will give us as result that the value contained in `a` is 4 and the one contained in `b` is 7. Notice how `a` was not affected by the final modification of `b`, even though we declared `a = b` earlier (that is because of the *right-to-left* rule).

A property that C++ has over other programming languages is that the assignment operation can be used as the rvalue (or part of an rvalue) for another assignment operation. For example:

```
a = 2 + (b = 5);
```

is equivalent to:

```
b = 5;
a = 2 + b;
```

that means: first assign 5 to variable `b` and then assign to `a` the value 2 plus the result of the previous assignment of `b` (i.e. 5), leaving `a` with a final value of 7.

The following expression is also valid in C++:

```
a = b = c = 5;
```

It assigns 5 to all the three variables: `a`, `b` and `c`.

Arithmetic operators (+, -, *, /, %)

The five arithmetical operations supported by the C++ language are:

+	addition
-	subtraction
*	multiplication
/	division
%	modulo

Operations of addition, subtraction, multiplication and division literally correspond with their respective mathematical operators. The only one that you might not be so used to see is *modulo*; whose operator is the percentage sign (%). Modulo is the operation that gives the remainder of a division of two values. For example, if we write:

```
a = 11 % 3;
```

the variable `a` will contain the value 2, since 2 is the remainder from dividing 11 between 3.

Compound assignment (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

When we want to modify the value of a variable by performing an operation on the value currently stored in that variable we can use compound assignment operators:

expression	is equivalent to
value += increase;	value = value + increase;
a -= 5;	a = a - 5;
a /= b;	a = a / b;
price *= units + 1;	price = price * (units + 1);

and the same for all other operators. For example:


```
// compound assignment operators

#include <iostream>
using namespace std;

int main ()
{
    int a, b=3;
    a = b;
    a+=2;           // equivalent to a=a+2
    cout << a;
    return 0;
}
```

5

Increase and decrease (++ , --)

Shortening even more some expressions, the increase operator (++) and the decrease operator (--) increase or reduce by one the value stored in a variable. They are equivalent to +=1 and to -=1, respectively. Thus:

```
c++;
c+=1;
c=c+1;
```

are all equivalent in its functionality: the three of them increase by one the value of c.

In the early C compilers, the three previous expressions probably produced different executable code depending on which one was used. Nowadays, this type of code optimization is generally done automatically by the compiler, thus the three expressions should produce exactly the same executable code.

A characteristic of this operator is that it can be used both as a prefix and as a suffix. That means that it can be written either before the variable identifier (++a) or after it (a++). Although in simple expressions like a++ or ++a both have exactly the same meaning, in other expressions in which the result of the increase or decrease operation is evaluated as a value in an outer expression they may have an important difference in their meaning: In the case that the increase operator is used as a prefix (++a) the value is increased before the result of the expression is evaluated and therefore the increased value is considered in the outer expression; in case that it is used as a suffix (a++) the value stored in a is increased after being evaluated and therefore the value stored before the increase operation is evaluated in the outer expression. Notice the difference:

Example 1	Example 2
B=3; A=++B; // A contains 4, B contains 4	B=3; A=B++; // A contains 3, B contains 4

In Example 1, B is increased before its value is copied to A. While in Example 2, the value of B is copied to A and then B is increased.

Relational and equality operators (==, !=, >, <, >=, <=)

In order to evaluate a comparison between two expressions we can use the relational and equality operators. The result of a relational operation is a Boolean value that can only be true or false, according to its Boolean result.

We may want to compare two expressions, for example, to know if they are equal or if one is greater than the other is. Here is a list of the relational and equality operators that can be used in C++:

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

Here there are some examples:

```
(7 == 5)    // evaluates to false.
(5 > 4)     // evaluates to true.
(3 != 2)    // evaluates to true.
(6 >= 6)    // evaluates to true.
(5 < 5)     // evaluates to false.
```

Of course, instead of using only numeric constants, we can use any valid expression, including variables. Suppose that $a=2$, $b=3$ and $c=6$,

```
(a == 5)    // evaluates to false since a is not equal to 5.
(a*b >= c)   // evaluates to true since (2*3 >= 6) is true.
(b+4 > a*c)  // evaluates to false since (3+4 > 2*6) is false.
((b=2) == a) // evaluates to true.
```

Be careful! The operator $=$ (one equal sign) is not the same as the operator $==$ (two equal signs), the first one is an assignment operator (assigns the value at its right to the variable at its left) and the other one ($==$) is the equality operator that compares whether both expressions in the two sides of it are equal to each other. Thus, in the last expression $((b=2) == a)$, we first assigned the value 2 to b and then we compared it to a , that also stores the value 2, so the result of the operation is true.

Logical operators (!, &&, ||)

The Operator $!$ is the C++ operator to perform the Boolean operation NOT, it has only one operand, located at its right, and the only thing that it does is to inverse the value of it, producing false if its operand is true and true if its operand is false. Basically, it returns the opposite Boolean value of evaluating its operand. For example:

```
!(5 == 5)    // evaluates to false because the expression at its right (5 == 5) is true.
!(6 <= 4)    // evaluates to true because (6 <= 4) would be false.
!true        // evaluates to false
!false       // evaluates to true.
```

The logical operators $\&\&$ and $||$ are used when evaluating two expressions to obtain a single relational result. The operator $\&\&$ corresponds with Boolean logical operation AND. This operation results true if both its two operands are true, and false otherwise. The following panel shows the result of operator $\&\&$ evaluating the expression $a \ \&\& \ b$:

$\&\&$ OPERATOR

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

The operator $||$ corresponds with Boolean logical operation OR. This operation results true if either one of its two operands is true, thus being false only when both operands are false themselves. Here are the possible results of $a \ || \ b$:

|| OPERATOR

a	b	a b
true	true	true
true	false	true
false	true	true
false	false	false

For example:

```
( (5 == 5) && (3 > 6) ) // evaluates to false ( true && false ).
( (5 == 5) || (3 > 6) ) // evaluates to true ( true || false ).
```

Conditional operator (?)

The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false. Its format is:

```
condition ? result1 : result2
```

If condition is true the expression will return result1, if it is not it will return result2.

```
7==5 ? 4 : 3 // returns 3, since 7 is not equal to 5.
7==5+2 ? 4 : 3 // returns 4, since 7 is equal to 5+2.
5>3 ? a : b // returns the value of a, since 5 is greater than 3.
a>b ? a : b // returns whichever is greater, a or b.
```

```
// conditional operator
```

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

```
int main ()
{
    int a,b,c;

    a=2;
    b=7;
    c = (a>b) ? a : b;

    cout << c;

    return 0;
}
```

```
7
```

In this example a was 2 and b was 7, so the expression being evaluated (a>b) was not true, thus the first value specified after the question mark was discarded in favor of the second value (the one after the colon) which was b, with a value of 7.

Comma operator (,)

The comma operator (,) is used to separate two or more expressions that are included where only one expression is expected. When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

For example, the following code:

```
a = (b=3, b+2);
```


Would first assign the value 3 to `b`, and then assign `b+2` to variable `a`. So, at the end, variable `a` would contain the value 5 while variable `b` would contain value 3.

Bitwise Operators (`&`, `|`, `^`, `~`, `<<`, `>>`)

Bitwise operators modify variables considering the bit patterns that represent the values they store.

operator	asm equivalent	description
<code>&</code>	AND	Bitwise AND
<code> </code>	OR	Bitwise Inclusive OR
<code>^</code>	XOR	Bitwise Exclusive OR
<code>~</code>	NOT	Unary complement (bit inversion)
<code><<</code>	SHL	Shift Left
<code>>></code>	SHR	Shift Right

Explicit type casting operator

Type casting operators allow you to convert a datum of a given type to another. There are several ways to do this in C++. The simplest one, which has been inherited from the C language, is to precede the expression to be converted by the new type enclosed between parentheses (`()`):

```
int i;
float f = 3.14;
i = (int) f;
```

The previous code converts the float number 3.14 to an integer value (3), the remainder is lost. Here, the typecasting operator was `(int)`. Another way to do the same thing in C++ is using the functional notation: preceding the expression to be converted by the type and enclosing the expression between parentheses:

```
i = int ( f );
```

Both ways of type casting are valid in C++.

sizeof()

This operator accepts one parameter, which can be either a type or a variable itself and returns the size in bytes of that type or object:

```
a = sizeof (char);
```

This will assign the value 1 to `a` because `char` is a one-byte long type. The value returned by `sizeof` is a constant, so it is always determined before program execution.

Other operators

Later in these tutorials, we will see a few more operators, like the ones referring to pointers or the specifics for object-oriented programming. Each one is treated in its respective section.

Precedence of operators

When writing complex expressions with several operands, we may have some doubts about which operand is evaluated first and which later. For example, in this expression:

```
a = 5 + 7 % 2
```


we may doubt if it really means:

```
a = 5 + (7 % 2)    // with a result of 6, or
a = (5 + 7) % 2    // with a result of 0
```

The correct answer is the first of the two expressions, with a result of 6. There is an established order with the priority of each operator, and not only the arithmetic ones (those whose preference come from mathematics) but for all the operators which can appear in C++. From greatest to lowest priority, the priority order is as follows:

Level	Operator	Description	Grouping
1	::	scope	Left-to-right
2	() [] . -> ++ -- dynamic_cast static_cast reinterpret_cast const_cast typeid	postfix	Left-to-right
3	++ -- ~ ! sizeof new delete	unary (prefix)	Right-to-left
	* &	indirection and reference (pointers)	
	+ -	unary sign operator	
4	(type)	type casting	Right-to-left
5	.* ->*	pointer-to-member	Left-to-right
6	* / %	multiplicative	Left-to-right
7	+ -	additive	Left-to-right
8	<< >>	shift	Left-to-right
9	< > <= >=	relational	Left-to-right
10	== !=	equality	Left-to-right
11	&	bitwise AND	Left-to-right
12	^	bitwise XOR	Left-to-right
13		bitwise OR	Left-to-right
14	&&	logical AND	Left-to-right
15		logical OR	Left-to-right
16	?:	conditional	Right-to-left
17	= *= /= %= += -= >>= <<= &= ^= =	assignment	Right-to-left
18	,	comma	Left-to-right

Grouping defines the precedence order in which operators are evaluated in the case that there are several operators of the same level in an expression.

All these precedence levels for operators can be manipulated or become more legible by removing possible ambiguities using parentheses signs (and), as in this example:

```
a = 5 + 7 % 2;
```


might be written either as:

```
a = 5 + (7 % 2);
```

or

```
a = (5 + 7) % 2;
```

depending on the operation that we want to perform.

So if you want to write complicated expressions and you are not completely sure of the precedence levels, always include parentheses. It will also become a code easier to read.

Basic Input/Output

Until now, the example programs of previous sections provided very little interaction with the user, if any at all. Using the standard input and output library, we will be able to interact with the user by printing messages on the screen and getting the user's input from the keyboard.

C++ uses a convenient abstraction called *streams* to perform input and output operations in sequential media such as the screen or the keyboard. A stream is an object where a program can either insert or extract characters to/from it. We do not really need to care about many specifications about the physical media associated with the stream - we only need to know it will accept or provide characters sequentially.

The standard C++ library includes the header file `iostream`, where the standard input and output stream objects are declared.

Standard Output (`cout`)

By default, the standard output of a program is the screen, and the C++ stream object defined to access it is `cout`.

`cout` is used in conjunction with the *insertion operator*, which is written as `<<` (two "less than" signs).

```
cout << "Output sentence"; // prints Output sentence on screen
cout << 120;               // prints number 120 on screen
cout << x;                 // prints the content of x on screen
```

The `<<` operator inserts the data that follows it into the stream preceding it. In the examples above it inserted the constant string `Output sentence`, the numerical constant `120` and variable `x` into the standard output stream `cout`. Notice that the sentence in the first instruction is enclosed between double quotes (") because it is a constant string of characters. Whenever we want to use constant strings of characters we must enclose them between double quotes (") so that they can be clearly distinguished from variable names. For example, these two sentences have very different results:

```
cout << "Hello"; // prints Hello
cout << Hello;   // prints the content of Hello variable
```

The insertion operator (`<<`) may be used more than once in a single statement:

```
cout << "Hello, " << "I am " << "a C++ statement";
```

This last statement would print the message `Hello, I am a C++ statement` on the screen. The utility of repeating the insertion operator (`<<`) is demonstrated when we want to print out a combination of variables and constants or more than one variable:

```
cout << "Hello, I am " << age << " years old and my zipcode is " << zipcode;
```

If we assume the `age` variable to contain the value `24` and the `zipcode` variable to contain `90064` the output of the previous statement would be:

```
Hello, I am 24 years old and my zipcode is 90064
```

It is important to notice that `cout` does not add a line break after its output unless we explicitly indicate it, therefore, the following statements:


```
cout << "This is a sentence.";
cout << "This is another sentence.";
```

will be shown on the screen one following the other without any line break between them:

```
This is a sentence.This is another sentence.
```

even though we had written them in two different insertions into `cout`. In order to perform a line break on the output we must explicitly insert a new-line character into `cout`. In C++ a new-line character can be specified as `\n` (backslash, n):

```
cout << "First sentence.\n ";
cout << "Second sentence.\nThird sentence.";
```

This produces the following output:

```
First sentence.
Second sentence.
Third sentence.
```

Additionally, to add a new-line, you may also use the `endl` manipulator. For example:

```
cout << "First sentence." << endl;
cout << "Second sentence." << endl;
```

would print out:

```
First sentence.
Second sentence.
```

The `endl` manipulator produces a newline character, exactly as the insertion of `'\n'` does, but it also has an additional behavior when it is used with buffered streams: the buffer is flushed. Anyway, `cout` will be an unbuffered stream in most cases, so you can generally use both the `\n` escape character and the `endl` manipulator in order to specify a new line without any difference in its behavior.

Standard Input (`cin`).

The standard input device is usually the keyboard. Handling the standard input in C++ is done by applying the overloaded operator of extraction (`>>`) on the `cin` stream. The operator must be followed by the variable that will store the data that is going to be extracted from the stream. For example:

```
int age;
cin >> age;
```

The first statement declares a variable of type `int` called `age`, and the second one waits for an input from `cin` (the keyboard) in order to store it in this integer variable.

`cin` can only process the input from the keyboard once the `RETURN` key has been pressed. Therefore, even if you request a single character, the extraction from `cin` will not process the input until the user presses `RETURN` after the character has been introduced.

You must always consider the type of the variable that you are using as a container with `cin` extractions. If you request an integer you will get an integer, if you request a character you will get a character and if you request a string of characters you will get a string of characters.


```
// i/o example
#include <iostream>
using namespace std;

int main ()
{
    int i;
    cout << "Please enter an integer value: ";
    cin >> i;
    cout << "The value you entered is " << i;
    cout << " and its double is " << i*2 << ".\n";
    return 0;
}
```

Please enter an integer value: 702
The value you entered is 702 and its double is 1404.

The user of a program may be one of the factors that generate errors even in the simplest programs that use `cin` (like the one we have just seen). Since if you request an integer value and the user introduces a name (which generally is a string of characters), the result may cause your program to misoperate since it is not what we were expecting from the user. So when you use the data input provided by `cin` extractions you will have to trust that the user of your program will be cooperative and that he/she will not introduce his/her name or something similar when an integer value is requested. A little ahead, when we see the `stringstream` class we will see a possible solution for the errors that can be caused by this type of user input.

You can also use `cin` to request more than one datum input from the user:

```
cin >> a >> b;
```

is equivalent to:

```
cin >> a;
cin >> b;
```

In both cases the user must give two data, one for variable `a` and another one for variable `b` that may be separated by any valid blank separator: a space, a tab character or a newline.

cin and strings

We can use `cin` to get strings with the extraction operator (`>>`) as we do with fundamental data type variables:

```
cin >> mystring;
```

However, as it has been said, `cin` extraction stops reading as soon as it finds any blank space character, so in this case we will be able to get just one word for each extraction. This behavior may or may not be what we want; for example if we want to get a sentence from the user, this extraction operation would not be useful.

In order to get entire lines, we can use the function `getline`, which is the more recommendable way to get user input with `cin`:


```
// cin with strings
#include <iostream>
#include <string>
using namespace std;

int main ()
{
    string mystr;
    cout << "What's your name? ";
    getline (cin, mystr);
    cout << "Hello " << mystr << ".\n";
    cout << "What is your favorite team? ";
    getline (cin, mystr);
    cout << "I like " << mystr << " too!\n";
    return 0;
}
```

What's your name? Juan SouliÃÂÃÂ
Hello Juan SouliÃÂÃÂ.
What is your favorite team? The Isotopes
I like The Isotopes too!

Notice how in both calls to `getline` we used the same string identifier (`mystr`). What the program does in the second call is simply to replace the previous content by the new one that is introduced.

stringstream

The standard header file `<sstream>` defines a class called `stringstream` that allows a string-based object to be treated as a stream. This way we can perform extraction or insertion operations from/to strings, which is especially useful to convert strings to numerical values and vice versa. For example, if we want to extract an integer from a string we can write:

```
string mystr ("1204");
int myint;
stringstream(mystr) >> myint;
```

This declares a `string` object with a value of "1204", and an `int` object. Then we use `stringstream`'s constructor to construct an object of this type from the `string` object. Because we can use `stringstream` objects as if they were streams, we can extract an integer from it as we would have done on `cin` by applying the extractor operator (`>>`) on it followed by a variable of type `int`.

After this piece of code, the variable `myint` will contain the numerical value 1204.

```
// stringstreams
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main ()
{
    string mystr;
    float price=0;
    int quantity=0;

    cout << "Enter price: ";
    getline (cin,mystr);
    stringstream(mystr) >> price;
    cout << "Enter quantity: ";
    getline (cin,mystr);
    stringstream(mystr) >> quantity;
    cout << "Total price: " << price*quantity <<
endl;
    return 0;
}
```

Enter price: 22.25
Enter quantity: 7
Total price: 155.75

In this example, we acquire numeric values from the standard input indirectly. Instead of extracting numeric values directly from the standard input, we get lines from the standard input (`cin`) into a `string` object (`mystr`), and then we extract the integer values from this string into a variable of type `int` (`quantity`).

Using this method, instead of direct extractions of integer values, we have more control over what happens with the input of numeric values from the user, since we are separating the process of obtaining input from the user (we now simply ask for lines) with the interpretation of that input. Therefore, this method is usually preferred to get numerical values from the user in all programs that are intensive in user input.