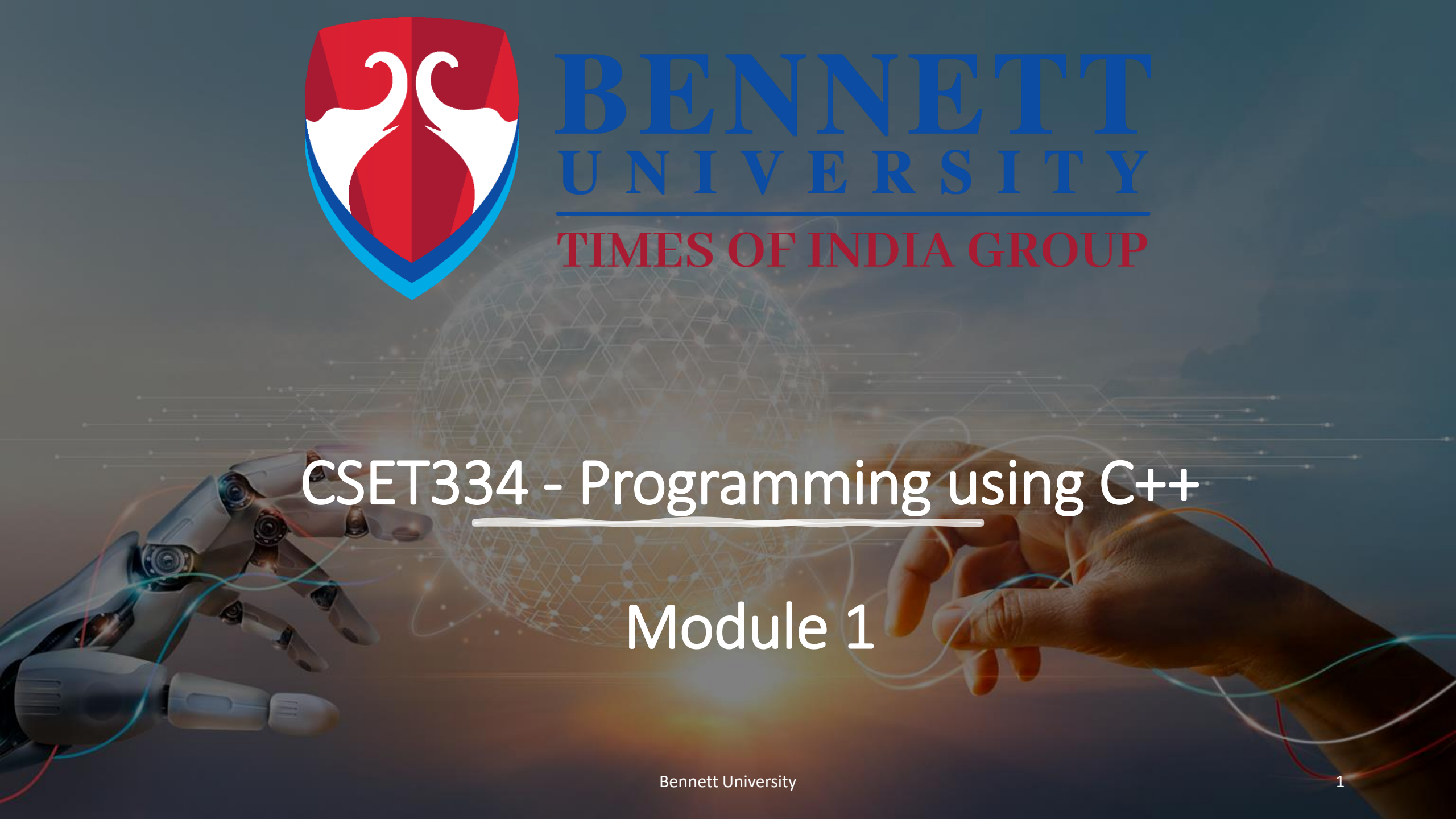


CSET334 - Programming using C++

Module 1

Course Specific Learning Outcomes

No.	Unit Learning Outcome
CO1	To explain the fundamental programming concepts and methodologies to building C++ programs.
CO2	To implement various OOPs concepts including memory allocation/deallocation procedures and member functions.

List of Contents

1. Principles of Object Oriented Programming

- OOP is a programming paradigm that encapsulates data and behavior into objects, enhancing security, scalability, and maintainability.

- Unlike POP, which divides problems into functions and relies on shared global data, OOP encapsulates data within objects, reducing interdependencies and improving data security.

- Core principles of OOP:
  - **Encapsulation:** Combines data and functions, restricting external access using access specifiers.
  - **Inheritance:** Enables code reusability and simplifies debugging by inheriting properties and methods.

1. Principles of Object Oriented Programming

- **Polymorphism:** Allows functions or methods to adapt their behavior based on the class context.
- **Abstraction:** Hides complex implementation, providing a simplified interface for users.

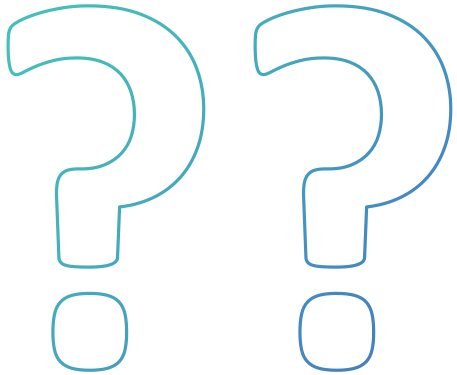
- OOP promotes modularity, secure data handling, scalability, and ease of maintenance.

- It is ideal for enterprise software, offering a structured way to manage complex data interactions and support code reuse through inheritance.

- Objects in OOP act as independent entities, facilitating secure communication via message passing.

1. Principles of Object Oriented Programming

- **Contrast in POP and OOP while implementing a problem**



1. Principles of Object Oriented Programming

- Exercise:

i. Create a class diagram for an online movie booking application system

1. Principles of Object Oriented Programming

- Students Exercise:

i. Create a class diagram for an online movie booking application system.

Hints: User class, Movie class, Theater class, Booking class

i. Create a class diagram for the student admission application for a university.

ii. Create a class diagram for teachers-student time table.

2. Data Types

- Defines the type of data that the variable can store.
- Data type must be specified at the time of the declaration of the variables.
- C++ compiler associates memory to the variable based on the data types.
- Data types restrict operations and defines a range of values a variable can hold.
- **Syntax:**

```
data_type variable_name;
data_type variable_name = value;
```

e.g. int average;

e.g. float salary = 10000;

2. Data Types

- Categories of Data Types in C++

S. No.	Type	Description	Data Types
1	**Basic Data Types**	Built-in or primitive data types that are used to store simple values.	**int, float, double, char, bool, void**
2	Derived Data Types	Data types derived from basic types.	**array, pointer, reference, function**
3	User Defined Data Types	Custom data types created by the programmer according to their need.	**class, struct, union, typedef, using**

2. Data Types

- Questions

Match the following with respect to C++ data types :

a.	User defined type	i.	Qualifier
b.	Built in type	ii.	Union
c.	Derived Type	iii.	Void
d.	Long double	iv.	Pointer

Code :

A. a-ii, b-iii, c-iv, d-i
B. a-iii, b-i, c-iv, d-ii
C. a-iv, b-i, c-ii, d-iii
D. a-iii, b-iv, c-i, d-ii

UGC-NET CSE

2012

2. Data Types

- Questions

Match the following with respect to C++ data types :

a.	User defined type	i.	Qualifier
b.	Built in type	ii.	Union
c.	Derived Type	iii.	Void
d.	Long double	iv.	Pointer

Code :

A. a-ii, b-iii, c-iv, d-i
B. a-iii, b-i, c-iv, d-ii
C. a-iv, b-i, c-ii, d-iii
D. a-iii, b-iv, c-i, d-ii

Answer: A

3. Symbolic Constants

- Constants in C++ are the values that cannot be changed.

- Once they are defined, the value of the constants remains same throughtout the program execution.

- Symbolic constants can be defined using the following three types in C++:
    i. **const** keyword
    ii. **constexpr** keyword
    iii. **#define** preprocessor

- **Example:** const int a = 5; int constexpr days_in_week = 7; #define PI 3.1416

4. Reference Variables

- When a variable is declared as a reference, it becomes an alternative name for an existing variable.
- A variable can be declared as a reference by putting ‘&’ in the declaration.
- ‘&’ is used for signifying the address of a variable or any memory.
- Variables associated with reference variables can be accessed either by its name or by the reference variable associated with it.
- Syntax:

data_type & ref = variable; e.g. int a = 10; int & ref = a;

5. Operators

- Operators are used to define an operation for one or more variables.

- In C++, operators are of following types:

  i. Arithmetic Operators
  ii. Assignment Operators
  iii. Relational Operators
  iv. Logical Operators
  v. Bitwise Operators

5. Operators

- Arithmetic Operators (Binary)

Name	Symbol	Description	Example
Addition	+	Adds two operands	int a = 3, b = 6; int c = a+b; // c = 9
Subtraction	–	Subtracts second operand from the first	int a = 9, b = 6; int c = a-b; // c = 3
Multiplication	*	Multiplies two operands	int a = 3, b = 6; int c = a*b; // c = 18
Division	/	Divides first operand by the second operand	int a = 12, b = 6; int c = a/b; // c = 2
Modulo Operation	%	Returns the remainder an integer division	int a = 8, b = 6; int c = a%b; // c = 2

5. Operators

- Arithmetic Operators (Unary)

Name	Symbol	Description	Example
Increment Operator	++	Increases the integer value of the variable by one	int a = 5; a++; // returns 6
Decrement Operator	—	Decreases the integer value of the variable by one	int a = 5; a–; // returns 4

5. Operators

- Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

5. Operators

- Relational Operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

5. Operators

- Logical Operators

Name	Symbol	Description	Example

Logical AND	&&	Returns true only if all the operands are true or non-zero	int a = 3, b = 6;
			a&&b;
			// returns true

Logical OR	||	Returns true if either of the operands is true or non-zero	int a = 3, b = 6;
			a||b;
			// returns true

Logical NOT	!	Returns true if the operand is false or zero	int a = 3;
			!a;
			// returns false

5. Operators

- Bitwise Operators

Bitwise Operators		
For all examples below consider a = 10 and b = 5		
Operator	**Description**	**Example**
&	Bitwise AND	a & b gives 0
|	Bitwise OR	a | b gives 15
^	Bitwise Ex-OR	a ^ b gives 15
~	1's complement (NOT)	~a gives some negative value
<<	Left shift	a << 1 gives 20
>>	Right shift	a >> 1 gives 5

5. Operators

- Questions

6. Operator Precedence

Operator	Name	Associativity
() [] -> .	Function call, Subscript, Member access	Left
++ --	Increment/Decrement	Right
! ~ - +	Logical/Bitwise NOT, Unary plus/minus	Right
* / %	Multiplication, Division, Modulus	Left
+ -	Addition, Subtraction	Left
<< >>	Bitwise shift	Left
< <= > >=	Relational operators	Left
== !=	Equality operators	Left
&	Bitwise AND	Left
^	Bitwise XOR	Left
|	Bitwise OR	Left
&&	Logical AND	Left
||	Logical OR	Left
?:	Ternary conditional	Right
= += -= *= /= %= &= ^= |= <<= >>=	Assignment and compound assignment	Right
,	Comma	Left


THANK YOU!