CAP202: OBJECT ORIENTED PROGRAMMING

Section: D2104

Program: M.Sc. IT

Set: B

Questions:

Sr. No	Question	Marks	Pages
1	Explain all the operators with code example?	15	1-11
2	Explain the concept of compiling & linking? Difference between procedural and object-oriented programming?	15	12-16

All Example Source Code

Submitted to:

Priyanka Ma'am

UID: 27309

Submitted by:

Pramatma Vishwakarma

Reg. No. 12103282

Roll No. A32

Question ⇒ 1

Explain all the operators with code Example?

Answer:-

Operators :- An operator is an object that is capable of manipulating a value or operator.

eg:- In "1" + 2", the 1 and 2 are the operands and the plus is the operator.

In C++ there are 7 types of operators.

1. Basic Arthmetic Operators
2. Assignment Operators
3. Auto-increment and Auto-decrement Operators.
4. Logical Operators.
5. Comparision (realational) operators.
6. Bitwise Operators.
7. Ternary operator.

1. Basic Arthmetic Operators.

C++ provides four familiar arthmatic Operators.

+ for performing addition
- for performing subtraction
* for performing multiplication
/ for performing division.

Each can be applied to either real (double) or integer (int) [or character (char)]* operrands

Division :- However, / bhaves differently for int than for double operands.

9.0/5.0 → 1.8	9.0/5 → 1.8
9/5 → 1	9/5.0 → 1.8

Types of Operands	Kind of Divison Performed
Double	~~real~~ real
int	integer.

Integer divison calculates the quotient, but it also calculate the reminder. To find it we can use the Modulus operator %; for Example, 9%5 = 4.

More Example :-

456 / 100 = 4
456% 100 = 56

2/3 = 0
2%3 = 2

Program Ex:-
```
#include <iostream>
using namespace std;
int main() {
        int num1 = 240, num2 = 40;
        cout << "num1 + num2:" << (num1 + num2) << endl;
        cout << "num1 - num2:" << (num1 - num2) << endl;
        cout << "num1 * num2:" << (num1*num2) << endl;
        cout << "num1 / num2:" << (num1 / num2) << endl;
        cout << "num1 % num2" << (num1 % num2) << endl;
        return 0;
}
```

Output :-

```
num1 +  num2:   280
num1  - num2 :  200
num1 *  num2 :  9600
num1 /  num2:   6
num  %  num2:   0
```

2. Assignment Operator :-

One common form of assignment is one in which the value of a variable is changed by performing some operation on it;

e.g Add value to sum
Double number

These can be written as assignment statements;

```
sum = sum + value ;
number = number * 2 ;
```

but such operations occur so frequently that special shortcut operations are provided for them. Insted of writing the assignment as shown, we can use the shorter forms:

```
sum += value;
number *= 2 ;
```

Each of the arithmetic operators can be used in this way. Any statement of the form,

variable = variable Δ expression;

Where Δ is any of the operators can be used in this way.

Δ = +, -, *, / or % can be written as

variable Δ = Expr;

Each of the following is therefore an acceptable variation of assignment operators :-

+=, -=, *=, /=, %=

These operation are all right-associative and produce the value assigned as their result. This means that they could be chained together. but this is not good programming practice because it can be difficult to follow how resulting expression are Evaluated.

Program Eg:-

```
#include <iostream>
using namespace std;
int main() {
int num1 = 240, num2 = 40;
cout << " = Output:" << (num2 = num1) << endl;
cout << "+= Output:" << (num2 += num1) << endl;
cout << "-= Output:" << (num2 -= num1) << endl;
cout << " *= Output:" << (num2 *= num1) << endl;
cout << "/= Output:" << (num2 /= num1) << endl;
cout << "%= Output:" << (num2 %= num1) << endl;
 return 0;
}
```

Output :-

```
  = Output  :  240
 += Output  :  480
 -= Output  :  240
 *= Output  :  57600
 /= Output  :  240
 %= Output  :  0
```

3: Auto-Increment and Auto-Decrement Operations

→ Auto-Icrement :- Algorithms often Contain instruction of the form "
"Increment count by 1"
that can be encoded as
Count = count +1 ;
or with the preceding Shortcut assignments forms, as
Count += 1
This is a Special unary increment Operator ++ is provided for this Operation. It can be used as postfix operator.
variable ++
or as a prefix Operator,
++ variable.

→ Auto-decrement :- Just as an integer variables value can be incremented with the ++ operator, it can be decremented (i.e., 1 is Subtracted from it) by using the decrement operator (--). for Example, the assignment Statement
count = count -1 ;
can be written more Compactly as,
Postfix → Count -- ;
or
-- Count ; ⟵ Prefix

Program Eg:-

```
#include <iostream>
using namespace std;
int main () {
        int num1 = 240, num2 = 40;
        num1++;   num2--;
        cout<<"num1++ is:" << num1 << endl;
        cout << num2-- is:" << num 2 << endl;
        return0;
  }
```

Output :-

```
num1++   is : 241
num2~~+~~--   is : 39
```

4: Logical Operators :- Logical Operators are used with binary Operators variables. They are mainly used in conditional Statements and loops for Evaluating a Condition.

Logical Operators in C++ are: &&, ||, !

Lets say we have two boolean variable b1 and b2

b1 && b2 will return true if both b1 and b2 are true else it would false

b1 || b2 will return false if both b1 and b2 are false it would return true.

!b1 would return the opposite of b1, that means it would be true if b1 is false and it would return false if b1 is true.

Program Ex:-

```
#include <iostream>
using namespace std;
 int main (){
        bool b1 = true;
        bool b2 = false;
        cout << "b1 && b2:" << (b1&&b2) << endl;
        cout << "b1 || b2:" << (b1||b2) << endl;
        cout << "!(b1&&b2):" << !(b1&&b2);
        return 0;
 }
```

Output:-

```
 b1 && b2 : 0
 b1  || b2 : 1
!(b1 && b2) : 1
```

5. Relational Operators:- The relational Operators are test between two operands.

Relational Operator	Relation Tested
<	Is Less than
>	Is greater than
==	Is equal to
!=	Is not Equal to
<=	Is Less than or Equal to
>=	Is greater than or Equal

They are used in boolean Expression of the form
Expre$_1$, relational-operator Expre$_2$

where $Expr_1$ and $Expr_2$ have compatible type. for Example, if root, a, b, c are type double, count is int and answer is of type char, than following are boolean Expression formed using these relational operator:-

root < 1
b * b >= 4.0 * a * c
count == 100
answer != 'T'

if root has the value 0.7, then the Expression
root < 1
has the value $\boxed{\text{true}}$. Similarly, if count has the value 99, then the Expression
count == 100
Produced $\boxed{\text{false}}$, Similarly Expression != T char value may also be compared using relational operator. by using numeric value $\boxed{\text{(ASCII)}}$ Table.

if char1 and char2 are character variable initialized by
char char1 = 'A', char2 = 'B';
the boolean Expression (ASCII Value ⇒ A = 65, B = 66)
char1 < char2 → True

Similarly char char1 = 'a', char2 = 'b';
(ASCII Value ⇒ a = 97, b = 98.
char1 < char2; → true

Similarly char char1 = 'a', char2 = 'B';
(ASCII value ⇒ a = 7, A = 65.
char1 < char2 ⟶ false.

Programe:-

```
#includ<iostream>
using namespace std;
int main () {
        int a = 5,  int b = 9;
    cout << "5 is equal to 9 =" << (a==b) << Endl;
    cout << "5 is not Equal to 9 =" << (a != b) << endl;
    cout<< " 5 is Less than 9 = " << (a<b) << endl;
    cout << "5 is greater than 9 = " << (a>b) << Endl;
    cout<< "5 is not Less than 9 = " << ( a >= b ) << endl;
    cout<< " 5 is not greater than 9 =" << (a<=b) << endl;
}
```

Output:-

```
5 is equal to 9 = 0
5 is not Equal to 9 = 1
5 is less than 9 = 1
5 is greater than 9 = 0
5 is not Less than 9 = 0
5 is not greater than 9 = 1
5 is false = 0
```

6 Bitwise Opreators:- In C++ the following 6 opreators are bitwise opreators (work at bit-level).

6.1.→ &(bitwise AND) :- takes two numbers as operands and does AND on Every bit of two numbers. The result of is 1 only if both bits are 1.

6.2→ |(bitwise OR) :- takes two numbers as oprends and does OR on Every bit of two numbers. The result of OR is 1 if any of the two bits is 1.

6.3 → ^(bitwise XOR):- takes two numbers as operands and does XOR on Every bit of two numbers. The result of XOR is 1 if the two bits are different.

6.4 → <<(left Shift):- takes two numbers, left shifts the bits of the first operand, the second operands decides the number of place to shif.

6.5 → >>(right shift):- takes two numbers right shifts the bits of the first operand the second operands decides the number of place to shift.

6.6 → ~(bitwise NOT) takes one number and inverets all bits of it.

Program Eg:-
```
#include <iostream>
using namespace std;
int main () {
   int a=5, b=9;
   cout<< "a = "<<a<<", " <<"b="<<b<< endl;
   cout << "a&=" << (a&b) <<endl;
   cout<<"a|b="<< (a|b) << endl;
   cout<< "a^b=" << (a^b) << endl;
   cout<< "~(" << a<< ")=" << (~a) << endl;
   cout<< "b <<1" << "=" << (b<<1) << endl;
   cout <<"b>>1" << "=" << (b>>1) << endl;
    return0;
 }
```

Output:
```
a=5 , b=9
a&b = 1
a|b = 13
a^b = 12
~a = 250
b<<1 = 18
b>>1 = 4
```

7 Ternary Operator :- This Operator Evaluates a boolean Express and assign the value based on the Result Syntax:-

variable num1 = (expression) ? value if true : if false.

if the Expression results true then the first value before the Colon (:) is assigned to the variable num1 Else the 2nd value is assigned to the num1.

Program Eg:-

```
#include <iostream>
using namespace std;
int main () {
    int num1, num2; num1 = 99
  num2 = (num1 == 10) ? 100 : 200
  cout << "num2: " << num2 << endl;
  num2 = (num1 = 99) ? 100 : 200;
    cout << num2:" << num2;
    return 0;
}
```

Output:-

```
num2: 200
num: 100
```

Other Operators :- Scope (::) operator, Subscript [], function Call (), Size of → size in byte of an Object or type, indirect member pointer selection (→)

Question 2:-
Explain the Concept of Compiling & linking? Difference between procedural and OOPs.

Answer:-
When Programmers talk about Creating programs, they often say, "it Compiles fine" or when asked if the program works, "Let's Compile it and See". This Colloquial usage might later be a Source of confusion for new Programmers. Compiling is not quite the Same as Creating an Executable file! Insted, Creating an Executable is multistage process divided into two Components: Compilation and linking.

Compilation:- it refers to the processing of Source code file (.c, .cc or .cpp) and the Creation of an "Objectfile". This Step doesn't Create anything the user can actully run. Insted the compiler merely produces the machine language instructions that correspond to the Source code file that was compiled. For instance, if you compile (but don't link) three seprate files, you will have three Object file as created output, Each with the name <filename>.obj (the Extension depend on your compiler). Each of these files Contains a translation of your Source code file into a machine language file -- but you can't run them yet! you need to form them into Executables your OS can use. That's where the linker comes in.

Linking :- it refers to the Creation of single Executable file from multiple Object files. In this step, it is common that linker will Complain about undefined functions (commonly, main it self). During Compilation, if the couldn't find the definition for a particular function, it would just assume that the function was defined in another file. If this isn't the case, there's no way to compiler would Know -- it doesn't look at the contents of more than one file at a time. The linker, on the other hand, may look at multiple files and try to find references for the functions that weren't mentioned.

Why there are Separate Compilation and linking Steps? First it is probably Easier to implement things that way. The compiler does its things, and the linker does its things -- by keeping functions seprate, the Complexity of the program is reduced. Another advantage is that this allows the Creation of Large programs without having to redo the compilation step every time a file is changed. insted, using so called "conditional Compilation". it is necessary to compile only those source files that have changed; for the rest, the Object files are suffient input linker. finally, this makes it simple to implement libraries of pre-compiled code: just create obj files link them just like any other Objects files.

to get the full benifit of Condition Compilation, It is probably Easier to get a program to help you than to try and remember which files you have changed since you last compiled. If you are working with an integrated development ~~ID~~ Environment (IDE) it may already take care of this for you.

Differences between Procedural Programming and Object-Oriented Programming :-

Comparison Parameter	POPs	OOPs
Definition	Procedural Programming is one type of programming paradigm or programming model based on Structured Programming.	Based on the concep of Obj which contains data and code, OOPs is one type of Programming paradigm or model.
Approach	POP fallows Top-Down approch	OOPs follows Bottom-Up-approch.
Access Modifiers	POP does not support any Kind of access modifiers	OOPs language Support different Kinds of access modifiers Examples :- Public, Private, protected.
Security	Due to unavailability of any Kind of access modifier, POP is less Secure.	Due to access Modifiers class can contain Private and method hence OOP is more Secure.

Main Concepts.	POP is Procedures. Procedures are sequence of actions that need to be performed. Data is generally stored in the variables.	main concepts of OOPs is Objects and Classes. Data is generally stored in the form of Attributes and in the forms of Objects.
Importance.	In POP, Importance is given to the function over data. Importance is given to the sequence that need to be followed.	In OOP, importance is given to the data, the way one stores the data and how securily data is accessed. Security and accessibility of data is very important in OOP
Real World Orientation	POP is not inclined with real world Orientation because while understanding POP concepts, one cannot relate it with real world Examples.	OOPs approces ~~itself~~ itself is derived from looking at real world Examples. Hence, OOP Concepts can easily be understood using comparison with real world.
Complexity	POP is less Complex to write because there is no need to define class or access Specifiers But after writing once, it is hard to read the Program or to update the program relative OOP.	OOPs program is relatively complex to write because of availability of Classes, Inheritence, access Specifiers and Other OOP Paradigms. But it is Easy to update the program or read the programs.

Data [redacted] Movemdnt	Data movement is move freely from one function to another because there is not any acess specifiers.	Data Cannot move freely in OOP because of access specifiers. But Objects can communicate with each other through member funcions.
Data Sharing	Data can be shared with global scope or local scope	Data can be shared with global, local and Class level scope.
Code Reusability	Code reusability is very limited in POP. It can only be achived through functions.	Code reusability is one of the aim of OOP. it can be achived using the Concept of Inheritence.
Overloading	The concept of overloading is not present in POP	The overloading can be achived in the form of Operator-overloading and function in OOP
virtual functio and Virtual Classes.	Concept of virtual function is not available in OOPs	There are many OOPs Systems available that uses virtual functions and virtual Classess.
Size of Problem.	POP generally prefered when the size of Problem is very small.	OOPs is prefered when one have to implement large tasks and systems.
SPEED	POP is generally runs faster than OOP.	OOP is relatively slow in Execution compared to POP.
Programming Languages	C, COBOL, Pascal Fortran, Basic.	C++, Python, Java, Kotlin, Javascript, Dart.