242466

PCPF

EXPERIMENT - 1

AIM WRITE A C++ PROGRAM TO DEMONSTRATE ENCAPSULATION AND INHERITANCE

FEATURES OF OOP

1. ASTRACTION:
   - HIDING COMPLEX DETAILS AND ONLY DISPLAYING ESSENTIAL FEATURS AND INFORMATION

2. ENCAPSULATION:
   - BUNDLING OF DATA MEMBERS AND MEMBER FUNCTIONS [PROPERTIES AND METHODS] WITHIN A CLASS

3. INHERITANCE:
   - DERIVING A CHILD-CLASS FROM A PARENT OR BASE-CLASS

4. POLYMORPHISM:
   - ABILITY OF A FUNCTION OR METHOD TO OPERATE IN DIFFERENT WAYS DEPENDING ON THE TYPE OF OBJECT ITS ACTING ON

5. OBJECTS AND CLASSES:
   - OBJECTS ARE INSTANCES OF CLASSES, WHICH DEFINE THE STRUCTURE AND BEHAVIOUR OF OBJECTS WHICH ARE A REPRESENTATION OF A REAL WORLD ENTITY OR THING

HOW TO ENCAPSULATE

- USE CLASSES TO BUNDLE DATA MEMBERS AND MEMBER FUNCTIONS
- CONTROL THEIR ACCESS VIA ACCESS SPECIFIER

HOW TO INHERIT

- CREATE A CHILD CLASS FROM AN EXISTING CLASS [BASE CLASS]
- REUSE CODE AND ESTABLISH A HEIRARCHIAL RELATIONSHIP
- CONTROL ACCESS VIA ACCESS SPECIFIER

DIFF. B/W C++ AND JAVA

No.	FEATURE	C++	JAVA
(1)	PLATFORM INDEPENDENT	NO	YES
(2)	MEMORY MANAGMENT	MANUAL	AUTOMATIC
(3)	PARADIGM	POP, OOP	OOP
(4)	PERFORMANCE	FASTER	SLOWER
(5)	MULTIPLE INHERITANCE	YES	NO
(6)	POINTERS	YES	NO

TYPES OF INHERITANCE

- SINGLE
- MULTI-LEVEL
- HIERARCHIAL
- HYBRID
- MULTIPLE

CONCLUSION

(1) WHAT WAS PERFORMED:

- PERFORMED A C++ PROGRAM TO DEMONSTRATE INHERITANCE AND ENCAPSULATIO

(2) WHICH TOOLS WERE USED:

- C++ PROGRAMMING LANGUAGE, VS CODE, AND STANDARD C++ LIBRARIES

(3) WHAT WAS OBTAINED:

- SUCCESSFULLY IMPLEMENTED ENCAPSULATION AND INHERITANCE IN C++

PCPF

EXPERIMENT - 2

WRITE A C++ PROGRAM TO DEMONSTRATE INITIALIZATION AND FINALIZATION

DIFFERENTIATE BETWEEN CONSTRUCTOR & DESTRUCTOR

	CONSTRUCTORS	DESTRUCTORS
1	INITIALIZES AN OBJECT WHEN IT IS CREATED, ALLOCATING RESOURCES	CLEANS UP AND DEALLOCATES RESOURCES WHEN OBJET DESTROYED
2	AUTOMATICALLY CALLED WHEN OBJET IS CREATED OR INSTANTIATED	AUTOMATICALLY CALLED WHEN OBJECT GOES OUT OF SCOPE
3	CAN BE OVERLOADED	CANT BE OVERLOADED
4	CAN HAVE PARAMETERS	CANT BE PARAMETERS
5	EXECUTES UPON OBJET CREATION OR INSTANTIATION	EXECUTES AT THE TIME OF OBJET DESTRUCTION
6	SAME NAME AS CLASS NAME	SAME NAME AS CLASS BUT PREFIXED WITH (~)
7	NO RETURN TYPE	NO RETURN TYPE
8	CAN BE CALLED EXPLICITLY	EXCEPT C++, CANT BE CALLED EXPUCITLY

WHAT IS INITIALIZATION

- INITIALIZATION IN C++ IS THE PROCESS OF ASSIGNING AN INITIAL VALUE TO A VARIABLE OR AN OBJECT WHEN IT IS CREATED
- THIS PROCESS SETS UP THE OBJECT OR VARIABLE IN A VALID STATE, SO IT CAN BE USED
- TYPES :
  - DIRECT
    - int x (10); // EXAMPLE
  - COPY
    - int y = 20; // EXAMPLE
  - LIST
    - int z {30}; // EXAMPLE
  - DEFAULT
    - int a; // EXAMPLE

WHAT IS FINALIZATION

- IT REFERS TO THE PROCESS OF CLEANING UP OR RELEASING RESOURCES USED BY AN OBJECT WHEN NOT NEEDED.
- THE FINALIZATION PROCESS ENSURES THAT RESOURCES SUCH AS MEMORY, FILES, CONNECTIONS ARE PROPERLY RELEASED.
- IT PRIMARILY DONE DESTRUCTORS IN C++, SINCE GARBAGE AND MEMORY ALLOCATION IS MANUAL

TYPES OF CONSTRUCTORS

DEFAULT :
- CONSTRUCTOR WITH NO PARAMETERS

PARAMETERIZED :
- CONSTRUCTOR THAT TAKES ARGUMENTS TO INITIALIZE OBJECTS WITH VALUES

COPY :
- CONSTRUCTOR THAT CREATES A NEW OBJECT AS A COPY OF AN EXISTING OBJECT.

CONCLUSION

1. WHAT WAS PERFORMED :
   - PERFORMED A C++ PROGRAM TO DEMONSTRATE INITIALIZATION AND FINALIZATION

2. TOOLS USED :
   - C++, VS CODE

3. WHAT WAS OBTAINED :
   - SUCCESSFULLY IMPLEMENTED INITIALIZATION AND FINALIZATION

EXPERIMENT – 3

WRITE A PROGRAM IN SWI-PROLOG TO FIND FACTORIAL OF A NUMBER

WHAT ARE FACTS

- REPRESENTS STATEMENTS ABOUT INFORMATION THAT CONSIDERED TO BE UNCONDITIONALLY TRUE IN THE SYSTEM
- THEY DESCRIBE RELATIONSHIPS BETWEEN OBJECTS OR PROPERTIES OF OBJECTS
- IN PROLOG, FACTS ARE WRITTEN AS PREDICATE LOGIC STATEMENTS
- A FACT CONSISTS OF A PREDICATE FOLLOWED BY ITS ARGUMENTS
- Eg.

```
parent (john, mary)
likes (mary, chocolate)
```

- HERE, THE FIRST LINE IS A FACT STATING THAT "JOHN" IS "PARENT" OF "MARY", THE SECOND IS A FACT STATING THAT "MARY" LIKES "CHOCOLATE"

WHAT ARE RULES

- DEFINES CONDITIONAL LOGIC AND ARE USED TO EXPRESS FACTS THAT DEPEND ON OTHER FACTS.
- A RULE CONSISTS OF A HEAD AND A BODY
- ITS READ AS "HEAD IS TRUE IF BODY IS TRUE"
- THE BODY OF RULE CAN CONTAIN MULTIPLE CONDITION

Eg.

```
grandparent(X,Y):-
        parent(X,Z),
        parent(Z,Y).
```

HERE, THE RULE STATES THAT "X" IS THE GRANDPARENT OF "Y" IF "X" IS THE PARENT OF "Z" AND "Z" IS THE PARENT OF "Y" THE PART BEFORE ":-" IS THE HEAD AND THE PART AFTER IS THE BODY

WHAT IS A DATABASE

PROLOG DATABASE IS A COLLECTION OF FACTS AND RULES THAT THE PROLOG PROGRAM CONTAINS. THE DATABASE SERVES AS KNOWLEDGE BASE WHERE PROLOG QUERIES ARE EXECUTED WHEN QUERIED, PROLOG SEARCHES THE DATABASE OF FACTS AND RULES TO INFER ANSWERS, PLUS THE DATABASE IS DYNAMIC, MEANING IT CAN BE UPDATED DURING EXECUTION

Eg.

```
parent(john, mary)          % FACTS
parent(mary, susan)         % FACTS

grandparent(X,Y):-          % RULES
        parent(X,Z),
        parent(Z,Y).
```

WHAT IS RESOLUTION OR UNIFICATION

IN ORDER TO DERIVE NEW STATEMENTS, A LOGIC PROGRAMMING SYSTEM, COMBINES EXISTING STATEMENTS CANCELLING LIKE TERMS, VIA A PROCESS CALLED RESOLUTION

Eg.

C ← A, B

D ← C

∴ D ← A, B

DURING RESOLUTION, FREE VARIABLES MAY ACQUIRE VALUES THROUGH UNIFICATION WITH EXPRESSIONS IN MATCHING TERMS, MUCH AS VARIABLES ACQUIRE TYPES IN "ML"

Eg.

flowery(X) ← rainy(X)

rainy(Rochester)

∴ flowery(Rochester)

CONCLUSION

WHAT WAS PERFORMED:

- INSTALLING SWI-PROLOG AND WRITING A BASIC PROGRAM TO GET FACTORIAL OF A NUMBER

TOOLS USED:

- SWI-PROLOG

WHAT ARE RESULTS:

- PROLOG IS INSTALLED SUCESSFULLY AND DEMONSTRATED HOW TO GET THE FACTORIAL OF A NUMBER

PCPF

EXPERIMENT - 4

WRITE A PROGRAM IN SWI-PROLOG TO DEMONSTRATE RECURSION USING TOWER OF HANOI PUZZLE

WHAT IS LISTS

A LIST IN PROLOG IS A FUNDAMENTAL DATA STRUCTURE USED TO STORE SEQUENCES OF ELEMENTS

LISTS IN PROLOG ARE VERY FLEXIBLE AND ARE VERY FLEXIBLE ARE OFTEN USED IN RECURSIVE PROGRAMMING AND PATTERN MATCHING.

Eg.

```
[1,2,3,4]        % NON EMPTY
[]               % EMPTY
```

OPERATIONS:

- member ⇒ CHECK IF ELEMENT IS IN LIST
  - member(X, [1,2,3])
    % RETURNS X=1, THEN X=2, THEN X=3
- concate ⇒ JOIN TWO LISTS
  - append([1,2], [3,4], L)
    % L = [1,2,3,4]
- length ⇒ GET LENGTH OF A LIST
  - length([1,2,3], N)
    % N = 3
- add ⇒ ADD TO START OF A LIST
  - X = [a | [b,c]]
    % X = [a,b,c]

WHAT IS BRACKTRACKING

BACKTRACKING IS ONE OF THE FUNDAMENTAL MECHANISMS IN PROLOG, USED TO FIND ALL POSSIBLE SOLUTIONS TO A QUERY.
WHEN PROLOG TRIES TO SOLVE A GOAL, IT EXPLORES ALL POSSIBLE SOLUTIONS BY SYSTEMATICALLY SEARCHING RULES AND FACTS.

WORKING:

- PROLOG TRIES TO SATISFY A QUERY BY SELECTING FACTS OR RULES THAT MATCH THE GOAL
- IF A RULE OR FACT FAILS TO SATISFY A QUERY, PROLOG BACKTRACKS TO THE PREVIOUS DECISION POINT AND TRIES ANOTHER OPTION, CONTINUING UNTIL ALL POSSIBLE SOLUTIONS ARE FOUND OR NO FURTHER OPTIONS REMAIN.

Eg.

```
% FACTS
parent (john, mary)
parent (john, tom)
parent (susan, mary)
parent (susan, tom)

% QUERY
?- parent (john, X)
```

WHEN YOU ASK PROLOG THE QUERY parent(john,X)
AND BINDS "X" TO MARY. ONCE THIS SOLUTION
IS FOUND, PROLOG WILL BACKTRACK TO TRY OTHER
POSSIBILITIES, FINDING "TOM" AS THE SOLUTION
FOR "X". IT CONTINUES UNTIL ALL POSSIBLE
SOLUTIONS ARE EXPLORED.

CONCLUSION

WHAT WAS PERFORMED?

INVOLVED INSTALLING THE SWI-PROLOG
ENVIRONMENT UNDERSTANDING ITS INTERFACE,
WRITING A PROLOG PROGRAM TO COMPUTE
THE FACTORIAL OF A NUMBER.

WRITE WHICH TOOLS YOU USED?

SWI-PROLOG

WHAT WAS OBTAINED?

SUCESSFULLY LEARNT HOW TO RECURSIVE LOGIC
IN PROLOG

PCPF

EXPERIMENT - 5

WRITE A PROGRAM IN SWI-PROLOG TO PRINT FIBONACCI SERIES

BASIC LIST OPERATIONS

IN PROLOG, LISTS ARE SEQUENCES ENCLOSED IN SQUARE BRACKETS AND ARE CENTERAL TO DATA MANIPULATION

OPERATIONS:

- CREATIONS => LISTS ARE CREATED VIA "[]"
- CONCAT => USE "append" TO COMBINE LISTS
- HEAD => ACCESS FIRST ELEMENT
- TAIL => ACCESS ENTIRE EXCEPT FIRST
- MEMBER => CHECK IF ELEMENT IN LIST
- LENGTH => RETURNS LENGTH OF LIST
- REVERSE => REVERSE A LIST

DATABASE OPERATIONS

OPERATIONS:

- ADDING => TO ADD FACTS TO THE DATABASE WE CAN USE THE "ASSERT/1" OR "ASSERT 2/1" PREDICATE
  - ASSERT/1 => ADD TO START
  - ASSERT 2/1 => ADD TO END
- DELETING => TO REMOVE FACTS WE USE retract/1 OR "retractall/1"
  - retract/1 => REMOVES FIRST INSTANCE THAT MATCHES

-retractall/1 => REMOVES ALL INSTANCES THAT MATCH ARGUMENT

- SELECT => RETRIEVE FACTS FROM DATABASE
- UPDATE => FIRST "retract" THE VALUE THEN "ASSERT" UPDATED VALUE

CONCLUSION

1. WHAT WAS PERFORMED:
PERFORMED A PROGRAM TO PRINT FIBONACCI SERIES OF A NUMBER, AND WHAT ARE LIST AND DATABASE OPERATIONS

2. TOOLS USED:
VS CODE, GHC, PROLOG

3. WHAT WAS OBTAINED:
SUCCESSFULLY DEMONSTRATED THE FIBONACCI SERIES OF A NUMBER USING LIST

PCPF

EXPERIMENT - 6

INSTALLATION OF HASKELL COMPILER AND PERFORMING SIMPLE OPERATIONS.

ARITHMETIC OPERATIONS

```
2+3
2-3
2*3
2*(-3)
2/3
50*100-4999
50*(100-4999)
50/5+(-3000)+40*100
```

LOGICAL OPERATIONS

```
False
True
False || True
False || False
True && False
True && True
not [True && False]
not [False || True]
False && False && True
True && True && False
False || True && True
False || False && True
True || False && False
```

COMPARATING OPERATIONS

- "a" == "b"
- "a" > "b"
- "a" < "b"
- 2 > 3
- 2 < 3
- 2 == 3
- 2 ≥ 3
- 2 ≤ 3
- "a" ≥ "b"
- "a" ≤ "b"
- not ("a" > "b")
- not (4 < 5)

	IMPERAITVE	FUNCTIONAL
1	FOCUSES ON HOW TO PERFORMS TASKS VIA STAIEMENTS THAT CHANGE PROGRAM STAIE	FOCUSES ON WHAT RESULT WILL BE THROUGH PROGRAM FUNCTIONS
2	MODIFIES STAIE	DOSENT MODIFY STAIE
3	COMMONLY USES LOOPS AND CONDITIONALS	USES BUILT-IN FUNCTION
4	Eg. C, C++	Eg. Haskell, Lisp

CONCLUSION

WHAT WAS PERFORMED:
INSTALLED HASKELL COMPILER AND PERFORMED BASIC ARITHMETIC OPERATIONS

TOOLS USED:
VS CODE, GITBASH, HASKELL, GHC

WHAT WAS OBTAINED:
SUCCESSFULLY INSTALLED HASKELL AND PERFORMED BASIC ARITHMETIC OPERATIONS

EXPERIMENT - 7

TO IMPLEMENT FUNCTIONS IN HASKELL

IMPLEMENT FOLLOWING INBUILT FUNCTIONS

succ 6

succ (succ 5)

min 5 6

max 5 6

max 101 101

succ 9 + max 5 4 + 1

(max 5 4) + (succ 9) + 1

div 92 10

div 3 4

div 4 3

4 / 3

mod 7 5

mod 3 1

IMPLEMENT FUNCTIONS TO

PRINT "Hello World"

ADD TWO NUMBERS

SUBTRACT TWO NUMBERS

MULTIPY TWO NUMBERS

DIVIDE TWO NUMBERS

PRINT FIBONACCI SERIES

NIXI Pg

MAP LIST [1..5] TO PRODUCE A LIST [2..6]
MAP LIST OF EVEN [1..10], [2..20]
FILTER FUNCTION TO GET ODD FROM [1..20]
FILTER FUNCTION TO GET NUMBERS <4 IN [1..10]

LAMBDA FUNCTIONS

DOUBLE A NUMBER
MULTIPLY TWO NUMBER
DIVISIBLE BY 5 IN [1..20]

EAGER EVALATION

- ALSO CALLED AS STRICT EVALUATION, IT COMPUTES ALL FUNCTION PARAMETERS BEFORE EXECUTING THE FUNCTION
- THIS CAN LEAD TO UNNECESSARY CALCULATIONS IF NOT ALL ARGUMENTS ARE USED BUT GIVES IMMEDIATE RESULTS

LAZY EVALUATION

- HERE, FUNCTION PARAMETERS ARE EVALUTED ONLY NEEDED FOR EXECUTION
- THIS ALLOWS FOR EFFICIENT MEMORY USE AND THE ABILITY TO HANDLE AN INFINTE AMOUNT OF DATA STRUCTURE BUT INCREASES COMPLEXITY.

WHAT ARE HIGHER ORDER FUNCTIONS

- HIGHER ORDER FUNCTIONS ARE THOSE FUNCTIONS THAT TAKE OTHER FUNCTIONS AS ARGUMENTS OR RETURN THEM AS RESULT, ENABLING FLEXIBLE AND RESOURCEFUL PROGRAMMING
- TYPES:
  - map => TAKES A FUNCTION & A LIST
    Eg: map (* 2) [1,2,3,4]
    ⟶ [1,4,9,16]
  - filter => filter even [1,2,3,4,5]
    Eg: FILTERS THE LIST BY THE FUNCTION
    ⟶ [2,4,]

CONCLUSION

1) WHAT WAS PERFORMED:
- IMPLEMENTED VARIOUS FUNCTIONS IN HASKELL, INCLUDING ARITHMETIC, HIGH ORDER & HIGHER ORDER FUNCTIONS.

2) TOOLS USED:
BASH, GHC, VS CODE

3) WHAT WAS OBTAINED:
SUCCESSFULLY IMPLEMENTED BOTH SIMPLE AND HIGHER ORDER FUNCTIONS IN HASKELL

PCPF

EXPERIMENT - 8

IMPLEMENTATION OF UST AND UST COMPREHSION IN HASKELL

UST PROCESSING FUNCTIONS

TYPES:

- map => APPLIES A FUNCTION TO EACH UST ELEMENT, RETURNING A NEW UST
  Eg: map (*2) [1,2,3]
  → [1,4,6]
- filter => SELECTS ELEMENTS OF UST THAT SATISFY CONDITION
  Eg: filter (>3) [2,3,4,5]
  → [4,5]
- concat => JOINS TWO UST
  Eg: [1,2] ++ [3,4]
  → [1,2,3,4]
- null => CHECKS IF A UST IS EMPTY, RETURNS A BOOLEAN
  Eg: null []
  → True
- length => RETURNS THE LENGTH OF THE UST AS AN INTEGER
  Eg: length [1,2,3]
  → 3
- head => RETURNS ONLY FIRST ELEMENT
  Eg: tail [1,2,3]
  → 1

LIST COMPREHENSION

- IT IS A CONCISE WAY OF GENERATING LISTS BASED ON EXISTING LISTS AND APPLYING CONDITIONS OR TRANSFORMATION

Eg:

```
[x * 2 | x ← [1..5], x > 2]
```

→ [6, 8, 10]

PATTERN MATCHING

- ALLOWS DECOMPOSING LIST INTO ITS HEAD & TAIL FOR PROCESSING. USED IN RECURSIVE FUNCTIONS THAT PROCESS LISTS

Eg:

```
sumList :: [Int] → Int
sumList [] = 0
sumList (x : xs) = x + sumList xs
```

IF sumList = [1, 2, 3, 4, 5]

→ 15

CONCLUSION

WHAT WAS PERFORMED:

LIST OPERATIONS, LIST COMPREHENSIONS AND FUNCTIONS IN HASKELL AND THEIR USES-CASES

OLS USED:

S CODE, BASH, GHC

HAT WAS OBTAINED:

UCCESSFULLY DEMONSTRATED DIFFERENT

ST OPERATIONS AND LIST COMPREHESION

EXPERIMENT - 9

IMPLEMENTING EXCEPTION HANDLING AND GARBAGE COLLECTIONS IN C++ AND JAVA

TRY / CATCH BLOCK

THE "try-catch" BLOCK IS USED TO HANDLE EXCEPTIONS THAT MAY OCCUR DURING PROGRAM EXECUTION.

IT ALLOWS YOU TO CATCH EXCEPTIONS, HANDLE THEM AS REQUIRED AND CONTINUE OR STOP THE PROGRAM BASED ON THE ERROR.

SYNTAX:

```
try {
    // CODE THAT MIGHT
    // FAIL OR RAISE EXCEPTION
} catch (Exception_Type obj) {

    // DEAL WITH EXCEPTION
}
```

"try" CONTAINS BLOCK OF CODE THAT MIGHT THROW AN EXCEPTION DURING RUNTIME, THEN REST OF THE BLOCK IS SKIP

"catch" BLOCK HANDLES THE EXCEPTION, YOU CAN MULTIPLE "catch" BLOCKS ACCORDING TO THE EXCEPTION RAISED IN THE "try" BLOCK.

NXT → Pg