

UNIT - VIII

① what are the programming Techniques in Scheme

1. programming in Scheme as in any functional language, relies on recursion to perform loops and other repetitive operations.

2. one standard technique for applying repeated operations to a list is to "cdr down and cons up" meaning we apply the operation recursively to the tail of a list and then collect the result with the cons operator by constructing a new list with the current result.

3. one example is the append procedure code ~~at the page~~. Another example is a procedure to square all the members in a list of numbers:

```
(define (square-list L)
```

```
(let ((null? L) '())
```

```
(cons (* (car L) (car L)) (square-list
```

```
(cdr L))))))
```

⑤ write a program by using operator overloading in C++

```

class Matrix
{
public const int size = 3;
private double [ ] m_matrix = new double [size, size];
// allow caller to initialize
public double this [int a, int b]
{
get { return m_matrix [a, b]; }
set { m_matrix [a, b] = value; }
}
// let user add matrices
public static Matrix operator + (Matrix mat1,
Matrix mat2)
{
Matrix newMat = new Matrix ();
for (int x=0; x < size; x++)
for (int y=0; y < size; y++)
newMat [x, y] = mat1 [x, y] + mat2 [x, y];
return newMat;
}
}

```

② Explain about Higher-order Functions

1. Since functions are first-class values in Scheme, we can write functions that take other functions as parameters and functions that return functions as values.

2. To give a simple example of a function with a function parameter, we can write a function `map` that applies another function to all the elements in a list and then pass it the `square` function to get the square list. Example:

```
(define (map f L)
```

```
  (if (null? L) '()
      (cons (f (car L)) (map f (cdr L)))))
```

```
(define (square x) (* x x))
```

```
(define (square-list L) (map square L))
```

3. Here is an example of a function that has a function parameter and also returns a function value:

```
(define (make-double f)
```

```
  (define (double-fn x) (f x x))
```

```
  (double-fn))
```

4. This function assumes that f is a function with two parameters and creates the function that repeats the parameter x in a call to f . The function value double f returned by `make-double` is created by a local define and then written at the end to make it the returned value of `make-double`.

5. Note that x is not a parameter to `make-double` itself but to the function created by `make-double`.

6. we can use `make-double` to get both the square function and the double function

(define square (make-double *))

(define double (make-double +))

7. The symbol "*" and "+" are the names for the multiplication and addition functions. Their values - which are function values - are passed to the `make-double` procedure, which in turn returns function values that are assigned to the names square and double.