**Output Statement** (p. 36)
```cpp
cout << expression1 << expression2 << ... << expression_n
```
(ex: `cout << pennies; AND/OR cout << "Total value = " << total << \n";`)
Purpose: Print the values of one or more expressions.

**Variable Definition** (p. 36)
```cpp
type_name variable_name;
type_name variable_name = initial_value;
```
(ex: `double total; AND/OR int pennies = 8;`)
Purpose: Define a new variable of a particular type, and optionally supply an initial value.

**Comment** (p. 41)
```cpp
/* comment text */
// comment text
```
(ex: `/* total value of the coins */ AND/OR // total value of the coins`)
Purpose: Add a comment to help a human reader understand the program.

**Input Statement** (p. 45)
```cpp
cin >> variable1 >> variable2 >> ... >> variable_n;
```
(ex: `cin >> pennies; AND/OR cin >> first >> middle >> last;`)
Purpose: Read the value for one more variables from the input.

**Assignment** (p. 47)
```cpp
variable = expression;
```
(ex: `total = pennies * 0.01;`)
Purpose: Store the value of an expression in a variable.

**Cast** (p. 50)
```cpp
static_cast<type_name>(expression)
```
(ex: `static_cast<int>(x + 0.5)`)
Purpose: Change an expression to a different type.

**Constant Definition** (p. 54)
```cpp
const type_name constant_name = initial_value;
```
(ex: `const double LITER_PER_OZ = 0.029586;`)
Purpose: Define a new constant of a particular type and supply its value.

**Function Call** (p. 58)
```cpp
function_name(expression1, expression2, ..., expression_n)
```
Purpose: The result of calling a function and supplying the values for the function parameters.

**Member Function Call** (p. 65)
expression.function_name(expression1, expression2, ..., expression n)
(ex: name.length()  AND/OR  name.substr(0, n – 1 ) )
Purpose: The result of calling a member function and supplying the values for the function parameters.

**Object Construction** (p. 81)
Class_name(construction parameters)
(ex: Time(19, 0, 0) )
Purpose: Construct a new object for use in an expression.

**Object Variable Definition** (p. 82)
Class_name variable_name(construction parameters);
(ex: Time homework_due(19, 0, 0); )
Purpose: Define a new object variable and supply parameter values for initialization.

**if Statement** (p. 116)
if (condition) statement
(ex: if (x >= 0) y = sqrt(x);
Purpose: Execute the statement if the condition is true.

**Block Statement** (p. 117)
{
    statement1;
    statement2;
    ...
    statementn;
}
(ex: {
    double length = sqrt(area);
    cout << area << “\n”;
}
Purpose: Group several statements into a block that can be controlled by another statement.

**if/else Statement** (p. 121)
if (condition) statement1 else statement2
(ex: if (x >= 0) y = sqrt(x); else cout << “Bad input
”;
Purpose: Execute the first statement if the condition is true, the second statement if the condition is false.

**while Statement** (p. 130)

while (condition) statement
(ex: while (x >= 10) x = sqrt(x);
Purpose: Execute the statement while the condition remains true.

**Function Definition** (p. 156)

return_type function_name(paramter1, parameter2, ..., parameter_n)
(ex: double abs(double x)
{
    if (x >= 0) return x;
    else return –x;
}
Purpose: Define a function and supply its implementation.

**return Statement** (p. 163)

return expression;
(ex: return pow(1 + p / 100, n);)
Purpose: Exit a function, returning the value of the expression as the function result.

**Function Declaration (or Prototype)** (p. 167)

return_type function_name(paramter1, parameter2, ..., parameter_n)
(ex: double abs(double x);)
Purpose: Declare a function so that it can be called before it is defined.

**Reference Parameter** (p. 172)

type_name& parameter_name
(ex: Employee& e AND/OR int& result)
Purpose: Define a parameter that is bound to a variable in the function call, to allow the function to modify that variable.

**Constant Reference Parameter**

const type_name& parameter_name
(ex: const Employee& e)
Purpose: Define a parameter that is bound to a variable in the function call, to avoid the cost of copying that variable into a parameter variable.
**Assertion (p. 190)**

```c
assert(expression);
```

(ex: `assert(x >= 0);`)  
**Purpose:** If the expression is true, do nothing. If it is false, terminate the program, displaying the file name, line number, and expression.

---

**Class Definition**

class *Class_name* {
   
   public:
   
   constructor declaration  
   member function declarations

   private:
   
   data fields

};

(ex: class *Point* {
   
   public:
   
   *Point* ( double xval, double yval );
   void move( double dx, double dy );
   double get_x() const;
   double get_y() const;

   private:
   
   double x;
   double y;

});

**Purpose:** Define the interface and data fields of a class.

---

**Member Function Definition (p. 217)**

```c
return_type *Class_name*::function_name(paramter1, parameter2, ..., parameter_n) [const]opt
 {
   
   statements

 }
```

(ex: `void *Point*::move(double dx, double dy)`

```
{x
   x = x + dx;
   y = y + dy;
}
```

```c
double *Point*::get_x() const
 {
   return x;
 }
```

**Purpose:** Supply the implementation of a member function.
Constructor Definition (p. 221)

Class_name::Class_name(parameter1, parameter2, ..., parameter_n)
{
    statements
}
(ex:   Point::Point( double xval, double yval )
    {
      x = xval; y = yval;
    }   )
Purpose: Supply the implementation of a constructor.

Constructor with Field Initializer List (p. 227)

Class_name::Class_name(parameters)
    : field1(expressions), . . ., field_n(expression)
{
    statements
}
(ex:  Point::Point( double xval, double yval )
    : x( xval ), y( yval )
    {
    }   )
Purpose: Supply the implementation of a constructor, initializing data fields before the body of the constructor.

for Statement (p.262)

for (initialization_statement; condition; update_expression) statement
(ex: for (int i = 1; i <= 10; i++) sum = sum + i;   )
Purpose: Execute the initialization statement. While the condition remains true, execute the statement and update expression.

do/while Statement (p. 268)

do statement while (condition);
(ex: do x = sqrt(x); while ( x >= 10);   )
Purpose: Execute the statement, then test the condition, and repeat the statement while the condition remains true.

Vector Variable Definition (p. 328)

vector<type_name> variable_name;
vector<type_name> variable_name(initial_size);
(ex: vector<int> scores;   AND/OR   vector<Employee> staff(20);   )
Purpose: Define a new variable of vector type, and optionally supply an initial size.
**Vector Subscript (p. 330)**

vector_expression[integer_expression]
(ex: salaries[ i + 1 ] )

Purpose: Access an element in a vector.

**Array Variable Definition (p. 349)**

type_name variable_name[size]
(ex: int scores [ 20 ] )

Purpose: Define a new variable of an array type.

**Two-Dimensional Array Definition (p. 356)**

type_name variable_name[size1][size2];
(ex: double monthly_sles [ NREGIONS ] [ 12 ]; )

Purpose: Define a new variable that is a two-dimensional array.