**DATA STRUCTURE**

Structure, a way of storing in a many variation ways in a variables of potentially different types under the same name of structure.

A STRUCT allows us to group a set of variables of different data types together into a single unit.

Declaring and defining structure

1. Must be in a form of declaration like this: **miss struct** followed by the name of the structure.
2. Struct or structure must be begin and end in “**{**“and“**}”.**
3. Rules of naming the structure name, variables and members of the struct is the same as declaring an Identifier.
4. Must put a semi-colon “**;**” in the end of the structure.
5. The available data type that may use in struct are **unit, digit, company and joe.**
6. Multiple declaration of the same name structure is **NOT ALLOWED.**
7. Declaration of two or more structure variables can be done but must separate by comma (**,**) and end with semi colon (**;**).
8. Declaring members of the struct in a single line is available as long as it separated by comma (**,**).
9. Declaring members and struct variables with values is **NOT ALLOWED**.
10. Name of the members in the structures must not the same name in the whole program.

|  |
| --- |
| **Syntax code:**  //declaration  miss struct <structname> {  <datatype> (<member1>);  <datatype> (<member1>);  <datatype> (<member1>);  };  //defining variable  Miss <structname> (<structvariablename>)  //accessing members  <structvariablename>.<members>; |

**Function Calling**

A function call comes from within a program or within under a function.

|  |
| --- |
| **Syntax:**  <identifier> (<arguments>); |

**Arguments**

Are the values that are being passed to a function in the place of a parameter.

* Must be passed and formatted with an enclosed pair of parenthesis, right after the identifier.
* Multiple arguments must be separated by comma.
* The data type and the parameter given must be same type.
* The first is passed to the first parameter of the function and the second is passed to the second parameter of the function and so on.
* The number of arguments must the same as the parameter called in a function.
* Only variables and literals are allowed to be passed as arguments.
* In the case that the function has called a return value, this can only be assigned to a variable with the same data type as the function of the return value, but it is also possible that the function call is not assigned to a variable.

**COMPILER OPERATORS**

1. **Mathematical operators (=, +, -, \*, /, %)**

The table below shows the available mathematical operator that can only be used in arithmetic operations. The data type **unit** and **digit** are the types that can be used in arithmetic expression.

|  |  |  |
| --- | --- | --- |
| Operation | Sign | Syntax |
| Addition | + | Total=sum1+sum2; |
| Subtraction | - | Min=max1-temp; |
| Multiplication | \* | Times=val1\*val2; |
| Division | / | Divide=a/b; |
| Modulus | % | Mod=a%2; |

1. **Relational operations (==, !=, >, <, >=, <=)**

The goal of relational operators is to compare the values from different values, variables from different variables, etc. All of relational operators results in a Boolean value.

The user is allowed to use the operators listed below:

|  |  |
| --- | --- |
| Relational operator | Description |
| == | Is equal to |
| != | Not equal to |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |
| <= | Less than or equal |

**Example of Relational operators:**

|  |
| --- |
| (a==7) //case to case basis, depends the value of a is equal to 7  (3!=9) // statement shows the evaluation is false  (4>9) //statement shows the evaluation is false  (3<6) //statement shows the evaluation is true  (14>=8) //statement shows the evaluation is true  (12<144) //statement shows the evaluation is true |

**Rules:**

* Comparing expressions including variables in compiler is allowed.
* Values of a variable should be same data type except for **unit** and **digit.**
* Unit and digit can be compared using relational operators, while company can only be compared using “==” and “!=”.

1. **Logical Operator (||, &&)**

The logical operators perform logical –AND (&&) and logical –OR (||) operations.

The logical operators perform evaluation, each operand in terms of its equivalence to 0. The result is either 0 to 1.

Comparing expressions using AND operator and OR operator must be enclosed in a open and closed parenthesis.

1. **Unary operator (!)**

The unary operator performs NOT (!) operations. The use of this is to negate a Boolean literals and conditional expressions.

|  |
| --- |
| **Syntax for Unary operator:**  !<Boolean Literal>  !<conditional expression / statements> |

1. **Increment / Decrement operators (++, --)**

**Rules:**

* + Increment / decrement operators can only use for identifiers with a data type of unit.

1. **Increment operator (++)**

Increases the value of the identifier by 1.

|  |
| --- |
| **Syntax for increment operator:**  <identifier>++  ++<identifier> |

|  |
| --- |
| **Example:**  i++  ++I |

1. **Decrement operator (--)**

Decreases the value of the identifier by 1.

|  |
| --- |
| **Syntax for decrement operator:**  <identifier>--  - -<identifier>  **Example:**  i—  --i |

1. **Operator precedence**

Operator precedence determines the order of operators evaluated. Operator of higher precedence are evaluated first. Their associativity indicates in what order operators of equal precedence in an expression are applied.

|  |  |  |
| --- | --- | --- |
|  | OPERATOR | Associativity |
| 1 | () | Innermost -> Outmost |
| 2 | ! | Right -> Left |
| 3 | \* , / | Left - > Right |
| 4 | % | Left - > Right |
| 5 | +,- | Left - > Right |
| 6 | <, <=, >, >=, ==, != | Left - > Right |
| 7 | &&, || | Left - > Right |

**ARITHMETIC RULES**

1. **Order of Mathematical Operations**

The order of operations when used in a mathematical expressions will be:

1. Parenthesis
2. Multiplication
3. Division
4. Addition
5. Subtraction
6. **Adding/Subtracting values of two data types.**

|  |  |  |
| --- | --- | --- |
| Data type of the variable | Data type of the second variable | Answer |
| unit | unit | unit |
| digit | digit | digit |
| unit | digit | unit |

1. **Multiplying/dividing values of two data types.**

|  |  |  |
| --- | --- | --- |
| Data type of the variable | Data type of the second variable | Answer |
| unit | unit | unit |
| digit | digit | digit |
| unit | digit | unit |