**BARANI INSTITUE OF INFORMATION TECHNOLOGY**

****

**SUBMITTED BY**

ZOHAIB AHMAD

15-Arid-2427

**SUBMITTED TO**

Mr. Omer

**DISCIPLINE**

BSCS-6C

**LANGUAGE NAME**

SCAR LANGUAGE

**What is Regular Expression?**

A regular expression is a special text string for describing a search pattern.

**What is the Use of Regex?**

A regular expression (sometimes called a rational expression) is a sequence of characters that define a search pattern. Usually this pattern is used by string searching algorithms for "find" or "find and replace" operations on strings, or for **input validation**.

**Common Challenging**

* Validating Email
* Validating CNIC Number
* Validating Mobile Number

**Lab 1st Assignment of Variable Declaration in Scar Language Project.**

* Declaration of Integer Variable

**Regex For Int Variables.**

[ |\t]\*INT[ |\t]\*@{1}[A-Z]+[ |\t]\*;

* **[ | \t]\*** means that their must be a single space or space equal to TAB
* **INT** Variable Datatype must be Start With INT
* **@{1}** Variable must start With and only contain 1 @ at the start of variable.
* **[A-Z]+** Variable length greater or equal than 1.

**Regex For Float Variables.**

[ |\t]\*FLOAT[ |\t]\*@{1}[A-Z]+[ |\t]\*;

* **[ | \t]\*** means that their must be a single space or space equal to TAB
* **FLOAT** Variable Datatype must be Start With FLOAT
* **@{1}** Variable must start With and only contain 1 @ at the start of variable.
* **[A-Z]+** Variable length greater or equal than 1.

**Regex For Char Variables.**

[ |\t]\*CHAR[ |\t]\*@{1}[A-Z]+[ |\t]\*;

* **[ | \t]\*** means that their must be a single space or space equal to TAB
* **CHAR** Variable Datatype must be Start With CHAR
* **@{1}** Variable must start With and only contain 1 @ at the start of variable.
* **[A-Z]+** Variable length greater or equal than 1.

public void VarDeclaration(string data, int i)

{

//variable Add in hastable

string[] variables = data.Trim().Split(' ');

//string[] variables = collonChecking.Trim().Split(' ');

foreach (var item in variables)

{

if (item != "")

{

if (item.Equals("INT") || item.Equals("FLOAT") || item.Equals("CHAR"))

{

datatype = item;

}

else

{

if (varType.ContainsKey(item.ToString()) == false)//does not contain hashtable key then add

{

if (datatype.Equals("INT"))

{

varType.Add(item, datatype);

varData.Add(item, 0);

}

else if (datatype.Equals("FLOAT"))

{

varType.Add(item, datatype);

varData.Add(item, 0 + "f");

}

else if (datatype.Equals("CHAR"))

{

varType.Add(item, datatype);

varData.Add(item, "");

}

else

{

richtxtResult.Text += "Data Type Are Not Used In This Language " + item + " At Line " + i + "\n";

}

}

else

{

richtxtResult.Text += "Already varaible Declared " + item + " At Line " + i + "\n";

}

}

}

}

**Symbol Table** is an important data structure created and maintained by the compiler in order to keep track of semantics of variable i.e. it stores information about scope and binding information about names, information about instances of various entities such as variable and function names, classes, objects, etc.

* It is built in lexical and syntax analysis phases.
* The information is collected by the analysis phases of compiler and is used by synthesis phases of compiler to generate code.
* It is used by compiler to achieve compile time efficiency.
* It is used by various phases of compiler as follows :-
  1. **Lexical Analysis:** Creates new table entries in the table, example like entries about token.
  2. **Syntax Analysis:** Adds information regarding attribute type, scope, dimension, line of reference, use, etc in the table.
  3. **Semantic Analysis:** Uses available information in the table to check for semantics i.e. to verify that expressions and assignments are semantically correct(type checking) and update it accordingly.
  4. **Intermediate Code generation:** Refers symbol table for knowing how much and what type of run-time is allocated and table helps in adding temporary variable information.
  5. **Code Optimization:** Uses information present in symbol table for machine dependent optimization.
  6. **Target Code generation:** Generates code by using address information of identifier present in the table.

**Symbol Table entries –** Each entry in symbol table is associated with attributes that support compiler in different phases.  
**Items stored in Symbol table:**

* Variable names and constants
* Procedure and function names
* Literal constants and strings
* Compiler generated temporaries
* Labels in source languages

**Information used by compiler from Symbol table:**

* Data type and name
* Declaring procedures
* Offset in storage
* If structure or record then, pointer to structure table.
* For parameters, whether parameter passing by value or by reference
* Number and type of arguments passed to function
* Base Address

**Operations of Symbol table –** The basic operations defined on a symbol table include:

https://cdncontribute.geeksforgeeks.org/wp-content/uploads/asd1-1.png

**Implementation of Symbol table –**  
Following are commonly used data structure for implementing symbol table :-

1. **List –**
   * In this method, an array is used to store names and associated information.
   * A pointer **“available”** is maintained at end of all stored records and new names are added in the order as they arrive
   * To search for a name we start from beginning of list till available pointer and if not found we get an error **“use of undeclared name”**
   * While inserting a new name we must ensure that it is not already present otherwise error occurs i.e. **“Multiple defined name”**
   * Insertion is fast O(1), but lookup is slow for large tables – O(n) on average
   * Advantage is that it takes minimum amount of space.
2. [**Linked List**](https://www.geeksforgeeks.org/data-structures/linked-list/) **–**
   * This implementation is using linked list. A link field is added to each record.
   * Searching of names is done in order pointed by link of link field.
   * A pointer **“First”** is maintained to point to first record of symbol table.
   * Insertion is fast O(1), but lookup is slow for large tables – O(n) on average
3. [**Hash Table**](https://www.geeksforgeeks.org/hashing-data-structure/) **–**
   * In hashing scheme two tables are maintained – a hash table and symbol table and is the most commonly used method to implement symbol tables..
   * A hash table is an array with index range: 0 to tablesize – 1.These entries are pointer pointing to names of symbol table.
   * To search for a name we use hash function that will result in any integer between 0 to tablesize – 1.
   * Insertion and lookup can be made very fast – O(1).
   * Advantage is wuick search is possible and disadvantage is that hashing is complicated to implement.
4. [**Binary Search Tree**](https://www.geeksforgeeks.org/binary-search-tree-data-structure/) **–**
   * Another approach to oimplement symbol table is to use binary search tree i.e. we add two link fields i.e. left and right child.
   * All names are created as child of root node that always follow the property of binary search tree.
   * Insertion and lookup are O(log2 n) on average.

**Example**

int value = 10;

void pro\_one()

{

int one\_1;

int one\_2;

{ \

int one\_3; | \_ inner scope 1

int one\_4; |

} /

int one\_5;

{ \

int one\_6; | \_ inner scope 2

int one\_7; |

} /

}

void pro\_two()

{

int two\_1;

int two\_2;

{ \

int two\_3; | \_ inner scope 3

int two\_4; |

} /

int two\_5;

}