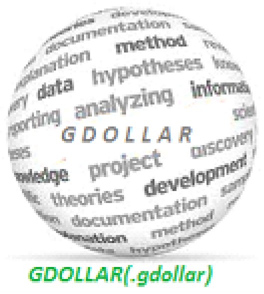


GDOLLAR Programming Language

BY

Jemin Information Technology

About the Author and Preface

This GDOLLAR is Designed by Analyzing many Research papers Using GDOLLAR one can build his own compiler, datastructures as Fast As could. I Thank God for this wisdom given to me…

----------Wilmix Jemin J,Jemin Information Technology

This EBOOK is Printed in Asia.

To Make Software Fast like Rabbit movement

and a global redistribution of prosperity

Acknowledgments

We'd like to acknowledge all of the people who played important roles in the creation

of this book. We'd also like to thank all of the developers who've spent time reading this manuscript

and pointing out all of the problems.

Finally, we'd like to extend a sincere thank you to the people who participated in the

GDOLLAR Program. In particular, those who've left feedback in the Author

Online forum have had a strong impact on the quality of the final printed product.

And for providing English translations of the text resources, we'd like to thank

Github and our supporters.

Thanks to all!

---------WILMIX JEMIN J

About this Book

Welcome to GDOLLAR! If you've picked up this book, we suspect you're a OOPS Professional concenterate only for research.

Perhaps you've worked with the Other Research Technologies in the past, perhaps this is your first step into GDollar P.L.

Whichever path has led you here, you're probably looking for a good introduction

to the new GDOLLAR Programming Language. This book intends to give you that introduction

and much more. If you've never heard of GDOLLAR, we cover the basics in enough

depth to keep you in tow. If you know what GDOLLAR does, but want a deeper understanding

of how it does it, we'll provide that too.

Roadmap

Book is focused on GDOLLAR Programming Language , if you have knowledge or experience about compiler design , building own datastructures can focus it.

But Minimum OOPS Technical Knowledge is required to focus on Studying, Designing GDOLLAR Modules .

GDOLLAR is an Advanced Technology focused on Research for

Compiler Design and Constructing Own DataStructures.

The Brief Contents

UNIT 1	Introduction to G$ Technology	8 -10
UNIT 2	G$ Datatypes,Looping, statements,operators	11- 34
UNIT 3	G$ Advanced Concepts	35- 38
UNIT 4	GDollar -MISC -PART1	39 - 56
UNIT 5	GDollar -MISC - PART2	57 -61

UNIT 6	GDollar Networking	62- 66
UNIT 7	G$ Coding standards ,Advantages over other P.L	67 - 70

UNIT 8	G$ MAIN Program Syntax And Advanced Concepts	71- 87
UNIT 9	G$ for Compiler Design , Datastructures ,Regular Expression	88 - 96
UNIT 10	GDollar Advanced Concepts	97 -100

UNIT 11	*G$ Compiler Design*	101 - 103
UNIT -12	*GDollar Mock Exercises*	104 - 106

Code conventions

The following typographical conventions are used throughout the book:

■ Courier typeface is used in all code listings.

■ Courier typeface is used within text for certain code words.

■ Italics are used for emphasis and to introduce new terms.

■ Code annotations are used in place of inline comments in the code. These highlight important concepts or areas of the code.

Code downloads

This will get you the GDOLLAR.zip file by purchasing it.

a couple of GDOLLAR archive files —as well as some documentation of the source. Instructions on how to install the application are contained in a README file in that download.

==

Unit 1 : Introduction to GDollar Technology

==

GDollar Technology is a modern Programming Language consists of JAVA OOPS, Behave like C/C++ ,GDollar Advanced OOPS Networking, RUN and compile at same time,it is used in Software Development , cloud computing ,Research, and ,Advanced OOPs.
It is used in case of constructing datastructures,etc.

SYNTAX FOR GDollar (.Gdollar) (beautiful syntax)

```
<GDollar>

<%
<! GDollar OOPS Logic !>

public void GDollar-Main()
{

}
?>

public void GDollar-Main()
```

note: This should be saved in filename.Gdollar

How GDollar Technology Works for GDollar-LIB?

At first .Gdollar is compiled by GDOLLARc compiler
GDollarc convert to intermediate code called as filename.C$. and it calls GDollarv.4 compiler.
After that GDOLLARv.4 compiles the intermediate code
to create .dll file immediately
so that user can directly use in GDollar-CWE Editor program.

How GDollar Technology works for GDollar-MAIN(CWE)?

When Gdollar compiler compilers a Gdollar program it
generates .dll and .exe file for futhure use.

Why you use Gdollar technology? what is the major advantage of Gdollar?
Since after compiling Gdollar code it generates
intermediate encode (.C$) files. That hacker or any body
cannot understood the code. ie) Hacker cannot take
the orginal source code.
GDollar technology prevents code stealing using this concept.

Why GDollar?

GDollar is a Programming Langauge and it used for constructing advanced datastruc
tures, complex datastructures, and focused on
compiler design and it is mostly by universities,colleges,companies,industries.
it is invented by wilmix jemin in JAVA ,C/C++ and editor
using JDollar(JWEB) P.L at 2016.

<u>What are Gdollar Modules?</u>

Gdollar has 5 modules they are....
a) GDollar -LIB
b) GDollar -Advanced OOPS (CWE -Editor)
c) GDollarv.4 (intermediate encoder)
d) JSLASH (autogenerated technology with in few seconds) (GDOLLAR COMPILERDESIGN)
e) GDollar - CJAVA

SYNTAX-1 (used only for creating libraries - .Gdollar)

```
<GDollar>
<IMPORT> Packagename;
<CLIB>

<%
```

<! OOPS Logic and datastructures !>

%>

How GDollar is formed ? What are its Advantages Over Native language JAVA Programming?

GDollar is formed in C++ OOPS concepts..
JAVA borrowed C++ OOPS concepts but
GDollar borrowed C++ OOPS concepts and JAVA oops and it has
Attractive syntax ; Plus in-build functions
for Program and it is responsible for creating
libraries (.dll). JAVA has attained the Programming
standards, But GDollar attains combination of C Technology
and JAVA Technology advantages.
GDollar Generates .dll files
but JAVA Generated .class files.
GDollar Has Advanced OOPS than JAVA 1.8.

UNIT -2 : GDollar DATATYPES,LOOPING,Statement,Operators

GDollarc Keywords

==================

<GDollar> <CUTIL> <IMPORT>

abstract add as ascending

async await base bool

break by byte case

catch char checked <CLASS>

const continue decimal default

delegate descending do double

dynamic else enum <EQUALS>

explicit extern false finally

fixed float for foreach

from get global goto

group if implicit in

int interface internal into

is	join	let	lock	
long	<PACK>		<NEW>	null
object	on	operator		orderby
out	override		params	partial
private	protected		public	readonly
ref	remove		return	sbyte
sealed	select	set	short	
sizeof	stackalloc		Shared	string
struct	switch	this	throw	
true	<TRY>	typeof	uint	
ulong	unchecked		unsafe	ushort
using	value	var	virtual	
void	volatile	where	while	
yield	<%	%>		

OTHER KEYWORDS IN GDOLLAR

AND -> AND operator

NOT -> NOT operator

# -> NOTEQUALS

RUN -> Runnable used in thread

TH-> Thread

<EXE> -> Exception

Friends -> Friend function

OTHER ATTRACTIVE SYMBOLS in GDOLLAR

--> => implements

<-- => extends

PRIMITIVE DATATYPES in GDOLLARC

The following table lists the available value types in CDollarcc (v.1)

bool	Boolean value	True or False	False
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit precise decimal values with 28-29 significant digits	(-7.9 x 1028 to 7.9 x 1028) / 100 to 28	0.0M
double	64-bit double-precision floating point type	(+/-)5.0 x 10-324 to (+/-)1.7 x 10308	0.0D
float	32-bit single-precision floating point type	-3.4 x 1038 to + 3.4 x 1038	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
long	64-bit signed integer type	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	8-bit signed integer type	-128 to 127	0
short	16-bit signed integer type	-32,768 to 32,767	0
uint	32-bit unsigned integer type	0 to 4,294,967,295	0
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0
ushort	16-bit unsigned integer type	0 to 65,535	0

OPERATORS in GDollarc

Operator Type	**Category**	**Precedence**
Unary	postfix	expr++ expr--
	prefix	++expr --expr +expr -expr ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== NOT=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	AND
	logical OR	OR
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

GDollarc has the following type of operators:

Arithmetic Operators

Relational Operators

Logical Operators

Bitwise Operators

Assignment Operators

Misc Operators

Arithmetic Operators

Example:

Assume variable A holds 1 and variable B holds 7 then:

Operator	Description	Example
+	Adds two operands	A + B = 8
-	Subtracts second operand from the first	A - B = -6
*	Multiplies both operands	A * B = 7
/	Divides numerator by de-numerator	B / A = 7
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 2
--	Decrement operator decreases integer value by one	A-- = 0

Relational Operators

Assume variable A holds 30 and variable B holds 10, then:

Show Examples

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.

!= Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. (A != B) is true.

> Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. (A > B) is true.

< Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. (A < B) is not true.

>= Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. (A >= B) is true.

<= Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. (A <= B) is not true.

Logical Operators

Assume variable A holds Boolean value true and variable B holds Boolean value false, then:

Operator	**Description**	**Example**
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit by bit operation. The truth tables for &, |, and ^ are as follows:

p	q	p & q	p | q	p ^ q
0	0	0	0	0

0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Operator Description Example

& Binary AND Operator copies a bit to the result if it exists in both operands.

| Binary OR Operator copies a bit if it exists in either operand.

^ Binary XOR Operator copies the bit if it is set in one operand but not both.

~ Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.

<< Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.

>> Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.

<u>Assignment Operators</u>

There are following assignment operators supported by CDollarcc:

Operator Description Example

= Simple assignment operator, Assigns values from right side operands to left side operand
C1 = A1 + B1 assigns value of A1 + B1 into C1

+= Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand C 1+= A1 is equivalent to C 1= C1 + A1

-= Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand C1 -= A1 is equivalent to C 1= C1 - A1

*= Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand C1 *= A1 is equivalent to C1 = C1 * A1

/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C 1/= A1 is equivalent to C1 = C1 / A1
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C1 %= A1 is equivalent to C1 = C1 % A1
<<=	Left shift AND assignment operator	C1 <<= 2 is same as C1 = C1 << 2
>>=	Right shift AND assignment operator	C 1>>= 2 is same as C 1= C1 >> 2
&=	Bitwise AND assignment operator	C1 &= 2 is same as C 1= C1 & 2
^=	bitwise exclusive OR and assignment operator	C1 ^= 2 is same as C1 = C1 ^ 2
|=	bitwise inclusive OR and assignment operator	C1 |= 2 is same as C1 = C1 | 2

<u>Miscellaneous Operators</u>

There are few other important operators including sizeof, typeof and ? : supported by Cdollarcc.

Operator	**Description**	**Example**
sizeof()	Returns the size of a data type.	sizeof(int), returns 4.
typeof()	Returns the type of a class.	typeof(StreamReader);
&	Returns the address of an variable.	ANDa; returns actual address of the variable.
*	Pointer to a variable.	*a creates pointer named 'a' to a variable.
? :	Conditional Expression	If Condition is true ? Then value A : Otherwise value B
is	Determines whether an object is of a certain type.	If(Girafee is animal) // checks if Girafee is an object of the Animal class.
as	Cast without raising an exception if the cast fails.	Object obj = new StringReader("Wilmix");

StringReader r = obj as StringReader

Operator Precedence in GDollarc

Operator precedence of the expression.some operators have higher precedence than others; for example, the multiplication or division operator has higher precedence than the addition operator.

For example x = 6 + 12 * 2; here, x is assigned 30, not 36 because operator * has higher precedence than +, so the first evaluation takes place for 12*2 and then 6 is added into it.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators are evaluated first.

Category	**Operator**	**Associativity**
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== NOT=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	AND	Left to right
Logical OR	OR	Left to right

Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma ,	Left to right	

Gdollar Statements consists of Print statements,

Read Statements , LOOPING Statements

Read Statements

Console.ReadKey();==> Read a vaue from console

Print Statements

SYNTAX:

```
GDOLLAR.WriteLine(String+datatype);
```

Types of Looping Statement

=============================

For Loop

========

For Loop operates when the condition met >= or <= or < or >.

At first counter is intialized to a value and it is followed by a condition

and it is followed by increment or decrement operator

A block inside the for loop to be executed if the condition met until false.

SYNTAX:

=======

```
for (index=intialialize value; index <> condition ; incrementor or decrementor)

{
```

```
<! BLOCK STATEMENTS !>




}
```

While Loop

==========

While Loop operates when the condition met >= or <= or < or > or == .which is tested at the TOP of the loop.

A block inside the WHILE loop to be executed if the condition met until false.

SYNTAX:

=======

```
while (index <> condition)

{
<! BLOCK STATEMENTS !>
```

}

Do – While Loop

================

Do - While Loop operates when the condition met >= or <= or < or > or == ; which is tested at the bottom of the loop.

A block inside the Do-WHILE loop to be executed if the condition met until false.

SYNTAX:

=======

```
do

{
<! BLOCK STATEMENTS !>

}

while (index <> condition)
```

foreach

========

The for-each loop introduced in CDollarc. It is mainly used to traverse array or collection elements.

The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.

Advantage of for-each loop:

==========================

It makes the clear consise of the code.

It eliminates the possibility of programming errors.

```
for  (index  in  collections)

{
<!  BLOCK  STATEMENTS  !>
```

}

Types of Conditional Statement

=================================

If Statement

==============

If Statement operates when the condition met it will

execute the block inside the if statement.

SYNTAX:

=======

```
if (condition1 ..... Condition.n)

{
```

<! BLOCK STATEMENTS>

}

If – Else statement

====================

If -Else Statement operates when the condition met it will

execute the block inside the if statement

or else execute the block inside else statement.

SYNTAX:

=======

if (condition1 ..... Condition.n)

{

```
<! BLOCK STATEMENTS>

}

else

{

<! BLOCK STATEMENTS>

}
```

If- Else-if statement

========================

If -Else Statement operates when the condition met it will

execute the block inside the if statement

or else execute the block inside if -else statement followed by a condition.

SYNTAX:

=======

```
if (condition1 ..... Condition.n)

{

<! BLOCK STATEMENTS>

}

else if (condition1 ..... Condition.n)

{

<! BLOCK STATEMENTS>

}
```

Switch Statement

================

Switch statement will test for the eqaulity when there is match with the value of expression.

The Statement inside the default statement is executed last when if none of the above case is satisfied.

if the statement is not followed by break then

another switch statement with equality is executed next.

or else it will skip that statement.

SYNTAX:

=======

```
switch (variable) {

  case v1:
       statements
       break;
  case v2:
       statements
       break;
  case v3:
  case v4:
       statements
  . . .
  default:
       statements
       break;
}
```

Types of Flow Control Statement

================================

Return Statement

================

Return Statement is used to return a value when a Function is a return type.

syntax: return value;

Continue Statement

===================

Continue statement is used to continue the loop.

SYNTAX:

=======

```
continue;
```

Break Statement

=================

Break statement is used to Skip from the loop.

SYNTAX:

=======

```
break;
```

Goto Statement

===============

Goto Statement is used as a climber to goto another block and execute

it.

SYNTAX:

=======

```
goto label;
```

Throw Statement

===============

Usually the throw statement is used with try-catch or try-finally statements.

A throw statement can be used in a catch block to re-throw the exception that the catch block caught.

SYNTAX:

=======

```
throw  exception;
```

ARRAYS

ARRAY is to store a value in a location

which uses stack datastructures..

===

UNIT-3 :GDollar ADVANCED CONCEPTS

ADVANCED OOPS CONCEPTS

Example -3:

Write a Program to print two String and add String 100 to
ArrayList.
and Technologies for year
2016 is C, GDollar ,GDollar,CHDOLLAR, JDOLLAR,JSTAR, JSAUCER.

Program :abc.Gdollar

```
<GDollar>
<IMPORT>  P
<%

class abc
{
public void GDollar-Main()
{
int i;
GDOLLAR.WriteLine("\nList of Technologies in year "+"2016 ");

string i1="weew";
GDOLLAR.WriteLine("wilmix"+i1);
GDOLLAR.WriteLine(" \njemin"+"is going");

}
}
%>
```

What will be the Output when you run using ?
>GDOLLARc abc.GDollar
Note: it will create .dll file for that.

Example-4:

Write a Program to add 1 lakh Natural integers using arraylist using GDollar LIB.
Program2:

```
<GDollar>
<CUTIL> //to load  GDollar  packages
<IMPORT>  P
<%
public class Program2
{
public  Shared void  LIB( )

{
<AList> ar= <NEW> <AList> ();
for (int i=0;i<=100000;i++)
ar.add(i,i);
}
}
%>
```

what is the intermediate code when Gdollar is compiled by GDollarc..
Program2.C$

```
<CDollar>
<CUTIL>
<IMPORT>  P
<%
public class Program2
{
public  *AB007 void  LIB( )

{
```

```
*AB043 ar= *AB091 *AB043 ();
for (int i=0;i<=100000;i++)
ar.add(i,i);


}
}
%>
```

What is the use?
Since we can reuse the code , and which means that it is the proof that this developer had coded it.
when GDollarc is used GDollarc compiler produces intermediate code and which again calls CDollarv.4 and which creates .dll file.

==

GDollar ADVANCED DATA STRUCTURES THEORY used in CWE EDITOR

GDollarArrays

SYNTAX for GDOLLARArrays:

CARRAYS list1 = new CARRAYS(string);
to add any collection objects to array use
add (String) functions and to Display those
objects use list1.Display();
Any class that use CARRAYS you should extends Array in class...

TREEOFARRAY

Write a Program to add 1 lakh Natural integers incremented by 10 using arraylist
and add the arraylist objects to TreeOfARRAY
what happens when you compile and execute the given the below program?

SYNTAX for TreeofArray:

TreeArray <name> = new TreeArray(String);
<name>.add(elements);
where elements may be string or collections....
It store other collection objects and stores huge amount of data
in tree format.
LISTOFARRAY

SYNTAX for List of Array:

LArray <name> = new LArray(string);
<name>.add(elements);
The elements may be string or collection elements.

Ans: It store other collection objects and stores huge amount of data
in tree format and sorts the elements in descending order and allow to insert the
element in to middle of list. This means act like combination of Set and Linked List , and Tree....

===

UNIT 4: MISC ,Fundametals of GDollar, Keywords,Operators,loops,Datatypes,Inner class, OOPS

concepts of GDollar ,and collections

===
GDollar OUTPUT STATEMENT

GDollar.WriteLine(" "+" ");
It is used for printing the output followed by line.
We had to add + operator to concatenate the outputs.....
String

String is represented by <Str> notation.

```
a) char[]  obj = <NEW> char[2];
        obj[0] = 'x';
        obj[1] = 'x';
        string eS = <NEW> string(obj);
```

This statement is used to create an object...
b) <Str> <strname> = value;
But this Statement will not create an object...
but it stores the value...
the differences between
a) if (s1==s2)
== means it is used to compare the values...
b) if s1.<EQUALS>(s2)
EQ means EQUALS is used to compare objects..
EXAMPLE

```
<GDollar>
<CUTIL>
<IMPORT>  P
<%
public class Program2
{
```

```
public  Shared void  LIB( )

{
<AList> ar= <NEW> <AList> ();
for (int i=0;i<=100000;i++)
ar.add(i,i);



 if (ar.<EQUALS>(ar)) // compare   two objects

GDOLLAR.WriteLine(""+ar);


}
}
%>
```

GDollar COLLECTIONS

Why we use collections in our software development?
Because for various projects we will use various kinds of datastructures that's why collections are focused.

Q: What are the Important concepts of Software Development?

ARRAYLIST

SYNTAX:

<AList> arraylistobjectname = <NEW> <AList>();
But type may be Object, int, Double,String,etc.
Why we focus Arraylist ?
Since ArrayList involves Powerful insertion and search mechanism when compared to array.
So we focus it.
Some built in functions available in ArrayList they are add and remove.
syntax : arraylistobjectname.add(loc,<datatype>);
loc means   location of the arraylist.
syntax: arraylistobjectname.remove(<datatype>);
How did you iterate the ArrayList?
by using While statement ...

LinkedList

<LList> arraylistobjectname <NEW> <LList>();
But type may be Object, int, Double,String,etc.
As according to collection concepts , built in functions are Designed for
LinkedList they are add and remove.

Actually when you study about Datastructures of LinkedList
and here we Designed the LinkedList using the LinkedList code
as mentioned in above that is LinkedList.c$. And add more functions...
and we use GDollar Generics...
What is the function of LinkedList? Why we use LinkedList?
In ArrayList You can't insert element in to the middle
or first or last so LinkedList is focused....
LinkedList is a Good example of Train....

VECTOR STACK

Here Vector implements Stack.
So we can mention in short notation as VList.
and Vector has push ,pop, and peek() apis.
push for push an element ,pop for POPing the last element.
peek for showing the firstelement in vector stack
Example:
=======

```
<GDollar>

<CUTIL>
<IMPORT>  P
{
public  class   abc

{

public  void  lib()
{
 VS s = <NEW> VS(3);
        s.<PUSH>(1);
        s.<PUSH>(2);
        s.<PUSH>(3);
        s.<PUSH>(4);
 while (NOTs.empty())
      {
            GDOLLAR.WriteLine("Top element is " + s.peek());
             GDOLLAR.WriteLine("Removed the element " + s.<POP>);
      }

}


}


}
```

OOPS in GDollar

A) INHERITANCE NOT USING EXTENDS METHOD...

C.GDollar

```
<GDollar>

<CUTIL>
<IMPORT>  P
{

class A
{
public A() { GDOLLAR.WriteLine("A's called"+"n"); }
}
class B
{
public B() { GDOLLAR.WriteLine("B's called"+"n"); }
}
class C
{
public C() { { GDOLLAR.WriteLine("C's called"+"n"); } }
public  void  lib()
{
<NEW> A();
<NEW> B();
<NEW> C();
}

}

}
```

Output:

B) POLYMORPHISM in GDollar

What is polymorphism?
It is Means action on method to do different things
based on the object that is action upon.
Example:

Write a Program to compute Rectangle Area and Triangle area
using Polymorphism.
Geometry.Gdollar

```
<GDollar>


<IMPORT>  P
{

public  class  EVEN
{

public void ISEVEN(int num1)
    {




for (int i=2;i<num1;i++)
{
```

```
if  (num1% 2==0)

 GDOLLAR.WriteLine("EVENNOS="+num1);



}

    }


    public void LIB()
    {

      int no = 100;

      int r;
      EVEN n = <NEW> EVEN();


      n.ISEVEN(no);



    }


}

}
```

C) Write a Progam to List Faculty , students using Diamond method in GDollar
:-
Note: Without Extends methods its calls methods and value when new ()
is intialized.
This is the Major Advantage of GDollar over native programming languages
like JAVA.
Program: TA.Gdollar

```
--------------

<GDollar>


<IMPORT>  P
{

class Person {


  Person(){}
public Person(int x)  { GDOLLAR.WriteLine("Person::Person(int ) called"+x);   }
}

class Faculty  {

public Faculty(int x)

{

<NEW> Person(x);
       GDOLLAR.WriteLine("Faculty::Faculty(int ) called"+x);
   }
}


class Student  {

public  Student(int x) {
<NEW> Person(x);
        GDOLLAR.WriteLine("Student::Student(int ) called"+ x);
   }
}



class TA   {
   TA(int x)  {
   <NEW> Faculty(x);
   <NEW>  Student(x);
  GDOLLAR.WriteLine("TA::TA(int ) called"+x);



   }


 public void LIB( )
{
   <NEW>  TA(30);


}

}



}
```

What will be the output ?

```
Tue Aug 18 07:59:57 GMT+00:00 2015*GDollar: Person::Person(int ) called
30Faculty::Faculty(int ) called
30Person::Person(int ) called 30Student::Student(int ) called 30TA::TA(int ) called 30
```

How to run this program?

```
GDOLLARc <Filename.Gdollar>
```

C) ABSTRACT CLASS

What did you meant by Abstract class?

Abstract class defines an Abstract concept which can't
be instanated using new Operator().
Where compare to multiple Inheritance it has an implementation
where multiple Inheritance cannot have.

```
<GDollar>


<IMPORT>  P
{


    class Programs1
    {
        public void LIB()
        {
           Subject subject = <NEW> Subject();
           GDOLLAR.WriteLine(subject.Describe());

        }
    }

    abstract class Topic
    {
        public virtual string Describe()
        {
            return "we  are  seeing  science  subject";
        }
    }

    class Subject : Topic
    {


public  string Describe()
        {
            return "We are seeing  maths  Geometry  subject";
        }
    }





}
```

===
===============

More about COLLECTIONS

Treeset

Treeset represent a collection that uses Tree datastructure for storage
Items in the collections are stored in Ascending or descending order.
<TS> objectname = new <TS>(<String>);
objectname.add(elements);
Write a GDollar Program about Treeset?
remaining things Developer should fill it.

```
<GDollar>

<CUTIL>
<IMPORT>  P
{

class  tree

{

public  void  lib()

{

   <TS> t = <NEW>  <TS>("wee");
t.add("13");
t.add("15");
t.add("12");
t.add("1");
GDOLLAR.WriteLine(""+t.ASCDisplay());
GDOLLAR.WriteLine(""+t.DESCDisplay());

}

}



}
```

```
Output:
------
Order=ASC[4][31][100][211][1123]
*******************************
Order=DESC[1123][211][100][31][4]
*******************************
Order=ASC[abraham][dion][priya][rahul][shiyam][wilmix]
*******************************
Order=DESC[wilmix][shiyam][rahul][priya][dion][abraham]
```

Operators conditions and loops

Operators

+ => ADD
++=> Increment
- => Substract
--=> Substract
* => Mulitply
/ = Division
~ => bitwise unary not operator
NOT (!) => flips true values to false and false values to true.
>>, >>>, and << => IT is used to shift all the bits of a number left
or right
a Specified number of places...

Other Operators

AND => And operator
OR => OR operator
?: => value =condition ? value1 : value2 (similar to if then else)
== => compare two values...
= => Assignment operators
EQ => Compare two objects

Relational Operators

> >= => Greater than , Greater than equals.
< <= => Less than , Less than equal
NOTEQ => Equals and not equals

NotEQ simillar to !=
CONDITIONS

IF Syntax:
----
if <condition> statements;
IF then else Syntax:
----
if <condition> statements else statements1
if <condition> statements1 else if condition1 statement2 .... and soon.
SWITCH Statements:
----
switch (expression)
{
case value1 :
statement1;
[break]
....
case valuen:
statementn;
[break]
-----
default:
default_statement;
}
Explanation:

If the expression is equals value1
statement1 will be executed.
if you use break it comes out of the loop
otherwise it continue to execute next statement.
The default value at the end is optional. It can be included if there are other values that can be held in
your variable but that you haven't checked for elsewhere in the switch statement.
THE WHILE LOOP
----
while (<condition>)
{
< Statements block>
}
Note: if the condition is true the block get executed.
otherwise the loop will be continued.

THE DO --- WHILE LOOP

```
do
{
< Statements block>
}
while( <conditon> )
```

Note: if the condition is true the block get executed.
and it is tested at the end of the loop, but not at the
beginning. The loop will be continued until it satisfies the condition.
biggest reason to be use the do – while loop is that
when you need the body of the loop to be run atleast once.

FOR LOOP

```
for ( identifier=value; cond1; iterator operators)
{
< Block statements >
}
```

For -EACH Statement

```
        //retrieving value using foreach loop
        foreach (string <VAR> in <OBJECT>)
        {
           statements;
        }
```

If you add integers (1 to 3) to arraylist
and if you wish to copy and store it in an integer variable
so that you can print the values that is copied from
arraylist.
Then follow this method of for each statements...

```
string[] hobies = { "twitter","cricket","footbal"};
  foreach (string hob in hobies)
{
GDOLLAR.WriteLine("value="+hob);
}
```

Output:

```
value=twitter
value =cricket
```

value =footbal
CONTINUE and Break

Break means it break out of loop
and continue means
it will continue to execute the statements;
for eg)
Program :WHILE LOOP with continue and break if statement...

```
<GDollar>

<CUTIL>
<IMPORT>  P
{


public class WHILE
{
public void LIB()
{
int a=0;
while (a <=10)
{
a++;
GDOLLAR.WriteLine("value="+a);
if ( a==9) continue;
else break;


}


}


}





}
```

```
Output:
-------
Wed Aug 19 10:09:23 GMT+00:00 2015*
GDollar: value=1
```

DATATYPES and OVERLOADING and OVERRIDING CONCEPTS, INNER CLASS

DATATYPES of GDollar are
----
int -> accept only int value
float -> accept float value=>eg) 1.5f
bool => true or false
char => accept character value
byte -> 1 byte
short -> 2 bytes
long-> 8 bytes
double-> for eg) 1.2121233232E9 => Accept double value
ARRAY => It is used to store values and had fixed size.
ARRAY

SYNTAX:

```
datatype[] object =  <NEW> datatype[10];

<GDollar>


<IMPORT>  P
{

public  class  abc
{
public  void  LIB()
{



int []a =  <NEW> int[10];


for (int i=1;i<=9;i++)
{
a[i]=i;



GDOLLAR.WriteLine(a[i]);
}

}
}


}
```

OTHER KEYWORDS IN GDollar

AND -> AND operator
NOT -> NOT operator
# -> NOTEQUALS
RUN -> Runnable used in thread
TH-> Thread
<EXE> -> Exception
Friends -> Frend function
INNER and OUTER CLASS

Inner class are nested inside outer class even if the fields declared as private members.

```
<GDollar>


<IMPORT>  P
{

class Outer {
private Shared int privInt = 10;

public void createInnerClass() {
Inner inClass = <NEW> Inner();

inClass.access();
}
class Inner {

public void access()

 {

GDOLLAR.WriteLine("The outer classs privInt is " + privInt);
}
}
}


}
```

OVERLOADING AND OVERRIDING functions

OVERLOADING

A functions with same name but different signature is called
as Overloading concept.
public void display(int i , String j) {}
=> If you pass int and string values from main program it will call
this function.
ABC a = <NEW> ABC(10,"ewew");
public void display(int i, int j) {}
ABC a = <NEW> ABC(10,20);

=> If you pass int and int values it will call this function.
OVERRIDING

A function with same name and same signature
will cause overriding....
Overriding can be avoided by using super() keyword.
in another class.

```
<GDollar>


<IMPORT>  P
{

public class section
 {
   public virtual string display()
   {
       return "CLASSA";
   }
 }
public class student:section
{

   public override string display()
   {
       return "CLASSA-100students-computerscience";
   }
}



}
```

Note: this will cause overriding
and it can be avoided by using super () keyword.

OTHER ATTRACTIVE SYMBOLS in GDollar

--> => implements
<-- => extends

==

UNIT: 5

FILE,Other collection concepts,Advanced Concepts of GDollar,MISC-2,Exception and Error,Garbage collection,Threads,Generics,GDollar Structures.

Advanced Topics in GDollar

OTHER COLLECTIONS CONCEPTS

<M> => map MEANS IT CONTAINS KEYS AND VALUE PAIRS...

HashSet

SYNTAX:

```
  <HASHSET> h <NEW> <HASHSET>(index);

h.add(data);
<PRINTLN>(""+h.GET());
```

HASHMAP

SYNTAX:

```
 <HASHMAP> h <NEW> <HASHMAP>(index);
 h.put(null, null); //you  can  also  put  null  key  and  null value
 h.put((data), null);

      h.display();
```

HASHTABLE

SYNTAX:

```
HashTable h  <NEW> HashTable(index);
      h.put(key,value);      h.display();
```

note: hash determines a order in which elements are
stored in the hash; SO it will display according
to hash stored order.

ADVANCED CONCEPTS of GDollar

ITERATOR

Iterator iterate about collection
in the forward direction .
and it will iterate record wise from the List or collection.

```
foreach( int a in  stringarray)
{
}
```

Here foreach statement is used for iterative purpose.

Exception and ERROR

Exception is a abnormal condition that arise during
the code sequence at run time.

<TRY> -> try in C/JAVA
<CATCH> -> catch in c/java
<Finally> -> final in c/java

SYNTAX:

```
<TRY>
{
< Executable good statements>
}
<CATCH> (<EXE> e)
{
GDollar.WriteLine(""+e);
}
<Finally>
{
<Final block statements>
}
```

Explanation:

When ever the Exception is true statements inside a try

block is executed; otherwise
statements inside a catch block is executed.
Exception occurs or not
final block get executed..
FINAL in GDollar

UnShared keyword means final in GDollar
eg)
UnShared int i=9;
// if a variable is declared as final
that value can't be changed.
eg)
UnShared class abc
{
....
}
if the class is declared as UnShared it can't
be overridden.
so if the method is declared as UnShared
such method can't be overriden by another class method.

Destructor:

Destructor means object is going to be destroyed.
~
where ~ is the Destructor operator.

Operator Overloading in GDollar

This means we can overload the operators
like + - = / > < >= <=

```
<GDollar>


<IMPORT>  P
{

UnShared class operatoroverloading
{
```

```
public Shared void operator *(int s1 ,int s2)
{
int s3=s1 * s2;
GDOLLAR.WriteLine(""+s3);
}
public  Shared void  LIB( )
{
operator *(10,10);

operator *(200,10000);
}




}


}
```

GENERICS

GENERICS means which is used to Pass Type as argument as class
for example if you want to pass String , int, float datatypes at the
same time and if you use display method to display the value of any
datatype
so Generic is most useful in that case.

```
<GDollar>


<IMPORT>  P
{
public class GEN<T>
{
T t;
T display(T t1)
{
t=t1;
return(t);
}
public  Shared void  LIB( )
{
```

```
GEN <int> i = <NEW> GEN<int> ();
GDOLLAR.WriteLine(""+ i.display(10));
}
}




}
```

===
=======
GDollar STRUCTURES

GDollar structure is another user defined data type available in GDollar programming, which
allows
you to combine data items of different kinds.
using the same memmory location. It also provide an efficient way of using the same
memory location for multi-purpose.
Thus GDollar Structures is Equivalent = C progamming Structures and union.
and it uses less memmory capacity than any Programming languages. IT is also
used to store collections, objecte ,etc.
IT is the most important datastructure implemented by wilmix jemin j.
He reduces the demerits of C Programming and
C child is GDollar. So GDollar has very beautiful and Advanced Concepts
than
any Programming Languages. And the Native Technology like JAVA
fails to do.
ADVANTAGES:

Billing programs, GUI, Record wise Search and Printing ,etc...

==

UNIT-6 :GDollar NETWORKING

==

GDollar Networking

N/w are essential to our life. Intenet is born due to networking and
A method of Client -server communications
gives like a house - to house interaction.

CLIENT SERVER PROGRAM

```
<GDollar>
<%
class CLIENTSERVER
{
public  Shared void  LIB( )
{
<CLIENT>("WILMIX","1099"); // Declare client and call client and pass hostname and port
<SERVER>("1099");//Declare server and server and pass portno
}
}
%>
?>
```

OUTPUT:

```
Sat Aug 22 08:52:19 GMT+00:00 2015*GDollar: Connecting to WILMIX on port 1099Waiting for
client on
port 1099...Socket timed out!
```

===
UNIT -6: MISC ,Advanced Concepts
===
Let us consider a Program to print using WHILE LOOP
Program -1

```
<GDollar>

<%
public class WHILE
{
public  Shared void  LIB( )
{
int a=0;
while (a <=10)
{
a++;
GDollar.WriteLine("value="+a);
}
}
}
%>
?>
```

Compilation:

```
GDOLLARc WHILE.Gdollar
```

Output:
(note: at One time compilation you will get this output in windows platform)

```
****************************************************************
Tue Mar 01 1
3:30:08 IST 2016*JAS: Error:
************************************************
****************************************
WHILE Tue Mar 01 13:30:09 IST 2016 GDollar:
Output: value=1value=2value=3value=4value=5value=6value=7value=8value=9value=10va
lue=11Error: value=1value=2value=3value=4value=5value=6value=7value=8value=9valu
e=10value=11
```

==
What is Pointers?

Variables that hold memory address are called pointers.

Why we mainly use Pointers?
Pointers reduces the length and complexity of the program,
They increase the execution speed.
It holds the memmory addres..
SYNTAX of GDollar Pointers:

{*} <pointer-name> Pointers(intialvalue);
for eg)
<Str> s ="Life is beautiful";
{*} l Pointers(s);
The given above statement will store the string "Life is beautiful"
in Pointer name l;

Example:

```
<Str>  s="dsdds";

 {*} l Pointers (s);

l.add(s);
for (int i = 0; i NOT= l.size(); i = i + 1)
 {

  <OBJECT> obj=l.GETKEY(i);
  <PRINTLN>(obj);



 }
```

what is the output?

```
dsdds
dsdds
```

BUCKET

Bucket are used to store key,value data, and Generated Random number
where datatype may be string ,object ,etc.
SYNTAX:

```
Bucket<DATATYPE> list = <NEW> Bucket<DATATYPE>(<DATATYPEVALUE>);
list.KeyAdd(<DATATYPEVALUE>);
```

list.add(<DATATYPEVALUE>);
list.RandomAdd();
list.Display(list);

Advantages

Using Bucket you can also Retrieve the values stored n position.
Searching and Insertion is fast than other DTS.
Random Indexing is possible.
eg) If you store a duplicate value such Random key will be different.
It also used to add many values.
EXTEND

Extend class is used in GDollar since to provide multiple inheritence
about 100000000 classes . Extends class also list values in methods and
constructor values.
Extend means a Bucket contains List of class and it is also
Behave like Bucket. So it is also one of the Advanced concepts in GDollar.
SYNTAX:

EXTEND <<DATATYPE>> list11 = <NEW> EXTEND <<DATATYPE>> (STRING);
list.KeyAdd(<DATATYPEVALUE>);
list.add(<DATATYPEVALUE>);
list.RandomAdd();
list.Display(list);

Advantages:

It is also used to add many values
Indexing is possible
Value can also be list by index and behave like bucket.
It list only the class value and object value.
It is stateless.
PIPE:

PIPE is used to maintain stateful state.
It is used for DataFlow in a Program. We can also add the values,
Constructor values of one class and other class and display it.
It also list the values from the Bucket.

SYNTAX:

```
Pipe <<DATATYPE>> list11 = <NEW> Pipe <<DATATYPE>> (STRING);
list.KeyAdd(<DATATYPEVALUE>);
list.add(<DATATYPEVALUE>);
list.RandomAdd();
list.Display(list);
```

Why we Prefer GDollar for software Field?

Used in BILLS, Forms ,Reports,Charts, any software project , GRAPHICS to web etc.

==

UNIT -7 :GDollar CODING STANDARDS AND ADVANTAGES OVER OTHER PROGRAMMING LANGUAGES

==

Coding Standards of GDollar

```
proper  syntax  fill  it...
<GDollar>
<USE> packages;
<CUTIL>
<IMPORT>  P
{
public class  <classname>
{
<! LOGIC !>
}
}
```

Note :

ALL Program should Start with <GDollar> means starting of a Program and scucceded by <IMPORT> package name
to load GDollar packages use <CUTIL>.

HIDE Unwanted code

```
<------ program code ------>
MAIN Program:
```

----
public Shared void LIB( ) indicates MAIN Program
otherwise it will not run the Program
STATIC BLOCK

Shared means Static keyword.
Shared will be executed first after that Main program will
be executed.
Shared
{
}
{} => This curly braces are mostly used.
SOME OPERATOR KEYWORDS

AND => && in java
NOT => !
=> !=
NEW Keyword

NEW is used to create an instance in memmory.
Always concenterate on important keyword not need to
memmorize at all.
DATATYPES:
----
int , char, double , float are ordinary keywords of oops
Programming language like C/JAVA/C#/GDollar.
STRUCTURES:

Always use Structure DATATYPE to store your data in objects form
so that it will reduce the storage allocation in memmory.
for one object it takes only 1 byte of memmory for structures.
RECYCLE:

Always use RECYCLE to make the unwanted object to be garbage collected.
//

If you want to describe something about your functions
use // . Don't use it unnessary at any way.
Special Characters

@,$%,^,[] are not allowed in the CDOLLLAR program
~

Use Destructor keyword to specify non GDollar resource deletion code to
be goes when you it..
Other Things
----
for loop , if statements, while loop, do while ,
for each, Switch statements, AutoBoxing , Generics , etc
are same.
Did GDollar support pointer?

YES.
private LinkedList nextNode =null;
consider this line ; This line creates a pointer to a class
LinkedList .
nextNode=new LinkedList(datum);
This statements are use to insert first data to Linkedlist
nextNode.add(datum) is used to insert many data....
Class Inheritance

If the Class is using another class variable in that case
you had to use <--- "extends" backward arrows
and front arrows --> for implements..
Implements is used when you use friend function.

GDollar ADvantages over JAVA and other Programming Languages
----

1) GDOLLAR PREVENTS CODE STEALING SO IT IS WIDELY USED.
GDOLLAR intermediate code tells that this programmer had coded it.
A) GDollar is the combination of JAVA , C/C++, and Advanced OOPS.
b) GDollar will only accept the shortest attractive syntax.
c) GDollar also used for construction of any datastructures.
d) GDollar helps the developers to provide inheritance by not using extends
keyword
and call the class in main program when use in linux.
e) GDollar Solves diamond Problem with multiple Inheritance when used in linux.
f) It also supports friendly function, pointers , and structures.
g) GDollar support Virtual memmory and garbage collection.
h) It is efficient, fast and easy to understand, and it is a OOPS Technology.
i) GDollar is a High level language.
j) GDollar is highly portable language
k) Using GDollar you can create any datastructures as libraries and

use it in your Application program.
l) GDollar language is a structured and object programming language.
m) GDollar has OOPS concepts like JAVA.
n) GDollar have the concept of Packages,etc.
o) GDollar have the concept of constructor or destructor and had magic oops concepts.
p) It Support functions with Default Arguments
q) It Supports Exception handling
r) It Support Generic Programming
s) It have pointer and Nodes..
t) GDollar is much simpler oops concepts, which leads to faster development and less mental overhead.
u) GDollar is almost always explicitly compiled
w) GDollar is easy to learn. GDollar was designed to be easy to use and is therefore easy to write,
compile, debug, and learn than other programming languages.
GDollar is object-oriented. This allows you to create modular programs and reusable code.
GDollar is platform-independent.
x) GDollar creates .exe or .dll files and it can
be used with GDollar main program (CWE EDitor) to create a complete software.
y) GDollar will compile and run at same time where other technology can't
do
z) GDollar is mainly used in complex programming , Billing the
goods,Graphics,etc
AA) GDollar is platform independant language
BB) GDollar is an interactive Technology.
CC) GDollar is used only in companies and industries.
DD) GDollar is used in compiler design and datastructures construction.

===

UNIT -8 :GDollar MAIN Program Syntax AND ADVANCED CONCEPTS PROGRAM.

===

<u>Syntax:</u>

```
<GDollar>
<INVOKE>
<PACK> <NAMESPACE>

   <CLASS> <CLASSNAME>
   {
      public FLOAT GDollar-MAIN()
      {

<! GDollar Logic !>




?>
```

BAG
=====
Bag is the extension of LinkedHashmap and it is the fastest
datastructures than
Dictionary.

SYNTAX:
=======

```
<GDollar>
<INVOKE>
<PACK>  bags

   <CLASS> bags
   {
      public FLOAT GDollar-MAIN()
      {

      Bag  b <NEW> Bag();

b.PUT(1,34); //  KEY  AS  1  AND  VALUE  AS  34
b.PUT(2,444);

<PRINTLN>(""+b);







?>
```

```
Bag object = new Bag();
object .put(key,value);
```

Functions

getValues(key) => it is used to get the values for a particular key

get(key,loc) => it is used to get the value stored at a loc (indexing purpose)

boolean containsValue(object Value) => To check the value present in bag or not.

put(key,value) => it is used to add key and value in Bag

remove(key ,value) => It is used to remove key and value.

TreeList
========

TreeList simillar to Bucket but store items in tree format.

TreeList list = new TreeList ("BUCKETS");
list.KeyAdd(KEY);
list.add(VALUE1);
list.RandomAdd(RANDOMNO);
list.DisplayO(list,0);

MASK
====
It is the extension of Tree Structure and it can store many values
using mask object and we can also retrieve the values stored in mask.
Mask m = new Mask(<DATATYPE>);
m.add(multiple values);
m.getR(Loc); => Get the values stored in right position
m.getL(LOC) => Get the values stored in left position

HEAP:
====

Creates a tree , puts the data into tree in a fairly balanced way and displays
the tree's size and data in a tree by performing an inorder traversal.
Heap hob = new Heap(<datatype>);
hob.add(datum);
hob = new Heap(key,value1,value2);

Bucktist
==========

Bucktist is simillar to Bucket but it is used to addd two values with one
key.
Bucktist l = null;
l= new Bucktist(key,value1,value2);

WICKET

=======

Wicket is used to store multiple values using same object with
4 values per key.

Syntax:

```
Wicket list12;
list12=new Wicket(key,v1,v2,v3,v4);
list12.Display();
list12.Display(list12,location);
```

EXAMPLE -1: BAG

```
<GDollar>
<INVOKE>
<PACK> MyP

    <CLASS> Programs
    {
        public FLOAT GDollar-MAIN()
        {

        Bag  b <NEW> Bag();

b.PUT(1,34);
b.PUT(2,444);

<PRINTLN>(""+b);






?>
```

EXAMPLE:2 : GDOLLARARRAYS

==========

```
<GDollar>



<PACK> MyP
{
   <CLASS> Programs
   {

      public FLOAT GDollar-MAIN()
      {






 <CDOLLARARRAYS> list1 <NEW>  <CDOLLARARRAYS>("ANIMALS ");
      list1.add("1 horse");
 list1.add("2 pig");
 list1.add("3 cow");
 list1.add("4 goat");
 list1.add("5 chicken");

 list1.add("6 ostrich");

 list1.Display();






?>
```

EXAMPLE-3: CREATE AN BOOTLOADER Using GDollar

```
 <GDollar>

<PACK>  MYOS
{
    <CLASS> MYOs
    {
public FLOAT GDollar-MAIN(){


<PRINTLN>("HelloWorld  for  booting  MYOS");




     </Statements>

?>
```

EXAMPE-4: POINTERS

```
<GDollar>
<PACK> MyP
{
    <CLASS> Programs
    {
        public FLOAT GDollar-MAIN()
        {


 <Str>  s="dsdds";

 {*} l Pointers (s);

l.add(s);

 for (int i = 0; i NOT= l.size(); i = i + 1)
 {

 <OBJECT> obj=l.GETKEY(i);
 <PRINTLN>(obj);
 }







?>
```

EXAMPLE-5: DICTIONARY

```
<GDollar>
<USE> CUTIL; //load  CUTIL  packages
<PACK> DTS
<%
   <CLASS> roots
   {
      public FLOAT GDollar-MAIN()
{
      <Dictionary> h <NEW> <Dictionary>(11);
     h.Add((80), (90));
     h.Add((40), (400));
     h.Add((65), (650));
     h.display();

     h.Add((58), (580));
     h.Add((24), (240));
     h.display();

     h.Add((2), (20));
     h.Add((13), (130));
     h.Add((46), (460));
     h.Add((16), (160);
     h.Add((7), (77));
     h.Add((21), (271));
     h.display();

     <TRY> {h.Add((99), (990));}
     <CATCH> (<EXE> e)
     {<PRINTLN>(" out of  memory");}
```

```
      // update element
      h.Add((7), (2977));
      h.display();

%>

?>
```

Example-6: EXTEND

```
<GDollar>

<INVOKE>


<PACK> MyP


    <CLASS> Programs

        public FLOAT GDollar-MAIN()

        {


 EXTEND list <NEW> EXTEND("BUCKETS");




      list.KeyAdd("1101");

                     list.add("jemin");

                     list.RandomAdd();
```

```
                    list.Display(list);

<PRINTLN>(""+list.DisplayO(list,1));





?>
```

EXAMPLE-7: HEAP

```
<GDollar>


<PACK> MyP
{
      <CLASS> Programs
    {
         public FLOAT GDollar-MAIN()
        {



 Heap root <NEW> Heap("wilmix");

 for (int i = 0; i <= 10; i = i + 1)
 {
 root.add("item " + i);
 }

 <PRINTLN>(root.size() );
 root.printTree();




?>
```

Example-8: LArray

```
<GDollar>

<USE> CUTIL;

<PACK> MyP
{
    <CLASS> Programs
    {
          public FLOAT GDollar-MAIN()
        {

 LArray root <NEW> LArray("root");


<ArrayList> ar <NEW> <ArrayList>();


 for  (int  i=0;i<=1000;i++)

 ar.add(i,i);

 root.add("wilmix");
 root.add("jemin");
 root.add("shalom");
 root.add("1010");
 root.add("101");
 root.add("201");
 root.add(ar.StringConvert());
 root.add("100000000");
 //print the tree's size and contents

 root.printTree();
```

```
?>


Example-9 : PIPE



<GDollar>

<PACK> MyP

{
   <CLASS> Programs
   {
public FLOAT GDollar-MAIN()
      {


Pipe list  <NEW> Pipe("BUCKETS");




    list.KeyAdd("1101");

               list.add("jemin");

               list.RandomAdd();

               list.Display(list);

<PRINTLN>(""+list.DisplayO(list,1));





?>
```

EXAMPLE-10: TREELIST

```
<GDollar>

<PACK> MyP

{
    <CLASS> Programs
    {
        public FLOAT GDollar-MAIN()
        {




TreeList list  <NEW> TreeList ("BUCKETS");







      list.KeyAdd("1101");

                     list.add("jemin");


                     list.RandomAdd("1111");

TreeList list2 <NEW> TreeList("BUCKETS");
 list2.KeyAdd("1102");

                     list2.add("rahul");


                     list2.RandomAdd("1112");
```

```
<PRINTLN>("DATA="+list.DisplayO(list,0));


<PRINTLN>("DATA="+list2.DisplayO(list2,0));




?>
```

Example-11 : MASK

```
<GDollar>


<PACK> My
{
    <CLASS> Programs
    {
           public FLOAT GDollar-MAIN()
        {


 MASK root <NEW> MASK("wilmix");

 for (int i = 0; i NOT= 10; i = i + 1)
 {
 root.add("item " + i);
 }




 root <NEW> MASK("root1",1211211,54441);

 root  <NEW> MASK("root2",121121,5444);
 root  <NEW> MASK("root5",99121888,"5");

 root  <NEW> MASK("root3",12112,544);

 root  <NEW> MASK("root4",1211,54);

 root  <NEW> MASK("root51",121,5);


 root.printTree();






?>
```

Example-12 : WICKET

```
<GDollar>


<PACK> MyPo
{
   <CLASS> Programs
   {
        public FLOAT GDollar-MAIN()
      {
 Wicket  list12;
 list12 <NEW> Wicket(1000,10002,43433,4343,5555451);
 list12 <NEW> Wicket(10001,100021,434331,4343,5555452);
 list12 <NEW> Wicket(10002,100022,434332,4343,5555453);
 list12 <NEW> Wicket(10003,100023,434333,4343,5555454);
 list12 <NEW> Wicket(10004,100024,434334,4343,5555455);
 list12 <NEW> Wicket(10005,100025,434335,4343,5555456);
 list12.Display(list12);

 <PRINTLN>("DATA="+list12.DisplayO(list12,0));




?>
```

Example-13 : STRUCTURE

```
<GDollar>
<PACK> MyPoi
{
    <CLASS> Programs
    {
          public FLOAT GDollar-MAIN()

        {


 <Str>  s="dsdds";

 {*} l Pointers(s);

l.add(s);

 for (int i = 0; i NOT= l.size(); i = i + 1)
 {

 <OBJECT> obj=l.GETKEY(i);
 <PRINTLN>(obj);



 }
 <STRUCTURE>  list <NEW> <STRUCTURE> (l.GETKEY(0));

 for (int i11 = 0; i11 NOT= list.size(); i11 = i11 + 1)
 {



 <OBJECT>  el=list.ret(i11);



 <PRINTLN>("SNO= "+el);

               }





?>
```

Example-14 : BUCKETIST

```
<GDollar>
<INVOKE>


<PACK> MyP
{
    <CLASS> Programs
    {
          public FLOAT GDollar-MAIN()
        {
```

```
Bucketist  bp    <NEW>  Bucketist("wilmix");

bp    <NEW>  Bucketist(1,222,434);
bp   <NEW>  Bucketist(1,222,434);


bp.Display(bp);


<PRINTLN>("DATA="+bp.DisplayO(bp,1));



?>
```

==

UNIT-9: G D O L L A R Technology focused for Compiler Design and datastructures design,Regular Expressions,Coding st andards.

==

COMPILER DESIGN USING GDollar

Introduction of Compiler Design

What is Compiler?

A Translator which transforms a highlevel language such as CDollar, JDOLLAR, C /C++,
Fortran or COBOL into a Particular computer machine or assembly language
Is called a compiler.

What is Interpreter?

It Process an internal form of the source program and data at a same time.
Ie, interpretation of the internal source form occurs at run time and
No object program is generated.

A compiler must perform two major tasks.
a) Analysis of a source program
b) Synthesis of a corresponding object program

A) Analysis of a Source program

a) Lexical Analyser

That we know Lexical Analyser is responsible for
Splitting the statements into tokens.
For eg) If A > B then is splited as
If 20
A 21
 >22
B 23
Then 24

Lexical Analyzer supplies tokens to syntax
Analyzer.

b) SYNTAX ANALYZER

IT's function is to take the source program from lexical

Analyzer and determine the manner in which it is

Decomposed into constituent parts.

Syntax Analyser outputs a syntax tree in
Which leaves are tokens and every nonleaf node represents
A syntactic class type.
SYNTACTIC TREE is factor, term , expression.

Semantic Analyser

It is main responsible of generation of intermediate form of source code.
Eg)
 (+ ,A,B,T1)
(+,C,D,t2)

The output of semantic Analyser is passed to code Generator.

At this point the intermediate form of the source program
is usually translated to either assembly language or machine language.

The Output of Code Generator is passed on to
a code Optimizer.

THE CODE OPTIMIZER

The Code Optimizer is responsible to produce a object or exe or class
Or wl files.

ERROR HANDLER

In compiler design Syntax error , invalid characters, out of memory exception,
Checked and unchecked exception are the exception occurs if a new
User do it. This error function f(X) is to determine the error in source code.
Without error handler no body can predict the errors..

SYMBOL TABLE

A Symbol Table will contains Variable name, Address, Type, dimensions, line declared or referenced, and pointer.
Mostly commonly performed on Symbol table are insertion and Lookup.

STORAGE ALLOCATION

In static storage allocation strategy it is necessary to be able to
decide at compile time
Exactly where each object will reside at run time.
But at dynamic storage allocation strategy the data area requirements for the progr
am
Are not known entirely at compilation time.

Note: Parser is divided in to topdown parsing and bottom up parsingÂ….
That you refered through many websites .
THE GDollar for compiler Design

GDollar and it's ADVANTAGES

What is GDollar? Who invented it? Who should focus it?
GDollar Technology is meant for Developing a compiler
By using GDollar Technology and it’s DataStructures…
which saves time and cost and years ;
And make you to develop a compiler with in a 6 month.

GDollar Technology is invented by wilmix jemin j in cdollar at first and
Fulfilled at year 2013 and modified at C# , JAVA , C/C++ P.L at year 2016.

System programmers, Technology inventors, GDollar Professionals,
And professionals who are interested in inventing compiler
Should be focused.

ABOUT GDollar

GDollar is an Opensource compiler focused on compiler Design. GDollar
belong to JAVA or C/C++ Group family.

GDollar is Invented in JAVA, C/C++, and Editor using Jdollar(JWEB)
GDollar virtual machine is used for
GDollar to run programs. G stands for Beta and Dollar stands for money.
So we called as Beta Technology.

GDollar is used by IT companies and industries in the world.

Syntax of GDollar:

```
<GDollar>

<USE> packages;

<%


<! OOPS statements !>


%>


 ?>
```

Merits of GDollar

> It is Good to create any compilers..

> It has simplified code

Demerits of GDollar:

It is not concenterated in creating Operating systems.
It is concenterated only in creating compilers.

How to compile and run GDollar?

```
GDollarc <filename>.Gdollar
```

Coding Standards of GDollar

```
<GDollar>
<USE> packages;

<%

 <! LOGIC OF GDOLLAR !>

%>

?>
```

Note : <% and %> is used to write class and it's logic.

ALL Program should Start with <GDollar> means starting of a Program and succeded by <USE>

to load GDollar packages and ?> Means End of the Program.

HIDE Unwanted code

<------ program code ------>

MAIN Program:
----

public void GDollar-Main( ) indicates MAIN Program

otherwise it will not run the Program

STATIC BLOCK

Shared means Static keyword.

Shared will be executed first after that Main program will

be executed.

```
Shared

{



}
```

{} => This curly braces are mostly used.

SOME OPERATOR KEYWORDS

AND => && in java

NOT => !

=> !=

<NEW> Keyword

<NEW> is used to create an instance in memmory.

Always concenterate on important keyword not need to
memorize at all.

DATATYPES:
----

int , char, double , float are ordinary keywords of oops

Programming language like C/JAVA/CDollar.

STRUCTURES:

Always use Structure DATATYPE to store your data in objects form

so that it will reduce the storage allocation in memmory.

for one object it takes only 1 byte of memmory for structures.

RECYCLE:

Always use RECYCLE to make the unwanted object to be garbage collected.

//

If you want to describe something about your functions

use // . Don't use it unnessary at any way.

Special Characters

@,$%,^,[] are not allowed in the GDollar program

~

Use Destructor keyword to specify non GDollar resource deletion code to

be goes when you it..

Other Things
----

for loop , if statements, while loop, do while ,

for each, Switch statements , Generics , etc

are same.

Did GDollar support pointer

YES.

Class Inheritance

If the Class is using another class variable in that case

you had to use <--- "extends" backward arrows
and front arrows --> for implements..

Implements is used when you use friend function.

===

UNIT-10: G D O L L A R ADVANCED CONCEPTS

UNION IN GDollar
================

SYNTAX:
==========
UNION u = new UNION(string);
UNION behave like structures but the only difference is
Union has build in functions like
a) Add(object)
b) Match(object,pos)
c) ret(Object)
d) Size() of object

Advantages:
structures is not used in the comparision of values so union is used.

USECASE in GDollar
==================

USECASE <Object> = new USECASE(datum);
datum may be datatype
eg) integer or string
<OBJECT>.Loc1(ob)
<OBJECT>.Loc2(ob)

USECASE behaves like UNION and is used for storing
objects at Loc1 ,Loc2 that has been used as a comparision with mathu1,matchu2
respectively.
and USECASE uses swithcase statement to asign values

ADVANTAGES:

i) Behave like struct
ii) compare the object
iii) Used for storing objects.
iV) it uses switch case statement

LOOP statement in GDollar:
=====================

SYNTAX:

```
LOOP  <Object> = new LOOP(datum);
datum may be datatype
 eg) integer or string
<OBJECT>.Loc1(ob)
<OBJECT>.Loc2(ob)
```

LOOP behaves like UNION and is used for storing
objects at Loc1 ,Loc2 that has been used as a comparision with mathu1,matchu2
respectively.

ADVANTAGES:

i) Behave like struct
ii) compare the object
iii) Used for storing objects.

BOOK in GDollar
=============

SYNTAX:

```
BOOK <OBJECT>= new BOOK(string);
<OBJECT>.STOREA(object,object);
<OBJECT>.STOREB(object,object);
```

It is used for storing object with key and value in STOREA and STOREB.
And it is used to compare with match statement

ADVANTAGES

i) Book is used for storing two block of pages.
ii) it is used in case of storing large amount of notes and it is used with database.

NOTE:

Examples for GDollar Advanced Datastructures is given in AdvancedDatastructures folders of GDollar Software.
GDollar is powerful than CDollar.

How to compile using GDollar and see the output?
GDollarc <filename>.Gdollar

CJAVA Program

```
<CJAVA>

<PACK> LArrays
{

    <CLASS> LArrays
   {



      public void main()
      {



LArray root <NEW> LArray("root");




 root.add("wilmix");
 root.add("jemin");
 root.add("shalom");
 root.add("1010");
 root.add("101");


 root.add("100000000");
 //print the tree's size and contents

 root.printTree();




     }
   }
}
```

Now compile using CDollacc compiler what it will happen?

```
F:\xxxxxx\GDOLLARSOFTWARE\ORGINAL\GDollar\outputs>LArrays.cjava.exe

 1010

 wilmix

 100000000

 root

 101

 jemin

 shalom
```

UNIT-11: G D O L L A R COMPILER DESIGN

Write a Gdollar Program to create a new Programming langauage?

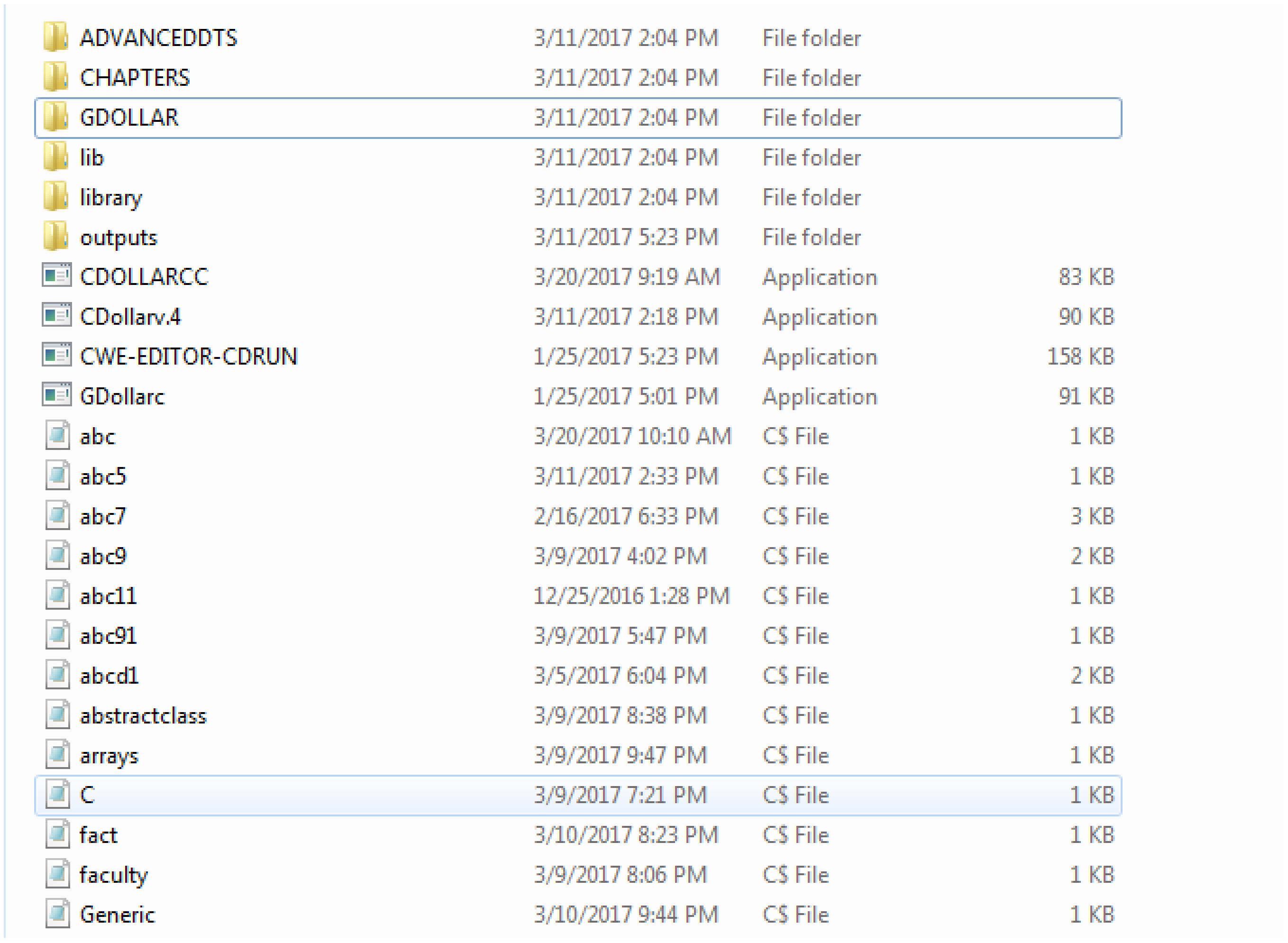
```
<JSLASH> //starting  Gdollar-jslash
<USE> compiler; //use  compiler   packages
<PACK> MyP
<%


 <CLASS> Jshell

<%


Shared String[] keywords1 = { "abstract", "boolean", "byte", "case",
            "catch", "char", "class", "continue", "default", "do", "double",
            "else", "extends", "final", "finally", "float", "for", "if",
            "implements", "import", "instanceof", "int", "interface", "long",
            "native", "new", "package", "private", "protected", "public",
            "return", "short", "static", "super", "switch", "synchronized",
            "this", "throw", "throws", "transient", "try", "void", "volatile","main",
            "while", "false", "true", "null","JSHELL","LOAD","JLOGIC;","CLOGIC","JEND" };
Shared String[] keytab1 ={"LOAD","JSHELL","JLOGIC","CLOGIC","JEND" };
//assign keywords

 public void main()
<%

String  t= args[0];
```

```
new Jcslash(keywords1); //assign keywords
String lext=".jshell"; // give  extension
String regx="\\s+|\\.+|\\;+|\\(+|\\)+|\\\"+|\\:+|\\[+|\\]+";
String regx1="\\<JSHELL>+|\\'LOAD'+|\\'JLOGIC'+|\\'CLOGIC'+|\\<JEND>+";
String regx2="^\\<JSHELL>+^\\'LOAD'+^\\'JLOGIC'+^\\'CLOGIC'+^\\<JEND>+";

//write   reqular  expression   to   parse   the  statements

      Jcslash.JParser(t,regx,53,regx1,regx2,lext,keytab1); //  call   jslash  parser


String path="out.txt";

Jcslash.signature(path,lext,t); //this  statement  will   call   usedefined   CDC compiler

// so  rename  your  compiler   as CDC.exe  create   using   C/C++  or  GDollar...




%>
```

<u>**Output:**</u>

Now you will get Jshell compiler which accepts attractive syntax...
(ie jshell.dll.exe in outputs folder)
with in 3 seconds...
Good bye to today olden methods...
So Gdollar is very exiting and powerful compiler.

GDollar Directory structure is given below?



ADVANCEDDTS	3/11/2017 2:04 PM	File folder	
CHAPTERS	3/11/2017 2:04 PM	File folder	
GDOLLAR	3/11/2017 2:04 PM	File folder	
lib	3/11/2017 2:04 PM	File folder	
library	3/11/2017 2:04 PM	File folder	
outputs	3/11/2017 5:23 PM	File folder	
CDOLLARCC	3/20/2017 9:19 AM	Application	83 KB
CDollarv.4	3/11/2017 2:18 PM	Application	90 KB
CWE-EDITOR-CDRUN	1/25/2017 5:23 PM	Application	158 KB
GDollarc	1/25/2017 5:01 PM	Application	91 KB
abc	3/20/2017 10:10 AM	C$ File	1 KB
abc5	3/11/2017 2:33 PM	C$ File	1 KB
abc7	2/16/2017 6:33 PM	C$ File	3 KB
abc9	3/9/2017 4:02 PM	C$ File	2 KB
abc11	12/25/2016 1:28 PM	C$ File	1 KB
abc91	3/9/2017 5:47 PM	C$ File	1 KB
abcd1	3/5/2017 6:04 PM	C$ File	2 KB
abstractclass	3/9/2017 8:38 PM	C$ File	1 KB
arrays	3/9/2017 9:47 PM	C$ File	1 KB
C	3/9/2017 7:21 PM	C$ File	1 KB
fact	3/10/2017 8:23 PM	C$ File	1 KB
faculty	3/9/2017 8:06 PM	C$ File	1 KB
Generic	3/10/2017 9:44 PM	C$ File	1 KB

UNIT -12: GDollar Mock Exercises

GDOLLAR MOCK EXERCISES

(1 100 = 100 marks)*

A) Develop a Accounting software with Credit/Debit /Discount on the goods sold and name of the Bank be ABC Ltd ,
use it in text format using GDollar.(10)

B) Develop a Telephone bill using CDollar

Advanced OOPS in Console. (10)

C) Briefly Describe G$ Program work flow with it's Advantages(5)

d) Write a Gdollar -JSLASH program to create a Programming language with

```
<NODE>
<IMPORT>
<%

<! Compile  only oops logic !>
%>
```

*(1 * 30 = 30 marks)*

f) ***GDollar Mini project (1 * 15 = 15 marks)***

In a Atm Transcation

i) Write a C$ program for Atm Transcation form using Cdollar Graphics.

ii) USE CDollar with WNOSQL DB

iii) Perform Transcations like Debit

5000rs or you wish.

If the A/c is low popup message is

displayed that a/c is low.

iv) Prepare a Report Based on C$ Transcations

v) USE GDollar with CDollar in this case for Software

Development.

g) Write A Gdollar Program to create a

datastructures like (3 * 10 =30 marks)

A) QUEUE BASED SYSTEM

B) LINKED LIST

C) Tree Structure System

BIBBLIOGRAPHY

GDollar Latest Tutorial will be updated in the given below Websites......

GDollar TUTORIALS

kindly go thru given tutorial url for more details....

GDollar Tutorial Url : https://sites.google.com/site/gdollarprogrammingtutorial
/gdollartutorial