COMPUTER SCIENCE & ENGINEERING
IIT KANPUR

CS335: Compiler Design Project

JAVA to x86_64 Compilation Toolchain

Kajal Deep (200489)

Kuldeep Singh Chouhan (200530)

Sandeep Kumar Bijarnia (200856)

Instructor: Dr. Swarnendu Biswas

TA: Abhishek Revskar

Support Provided

01

PRIMITIVE TYPES

02

MULTIDIMENSIONAL ARRAYS (ANY DIMENSION)

03

BASIC OPERATORS

04

METHOD DECLARATION AND INVOCATION

05

FUNCTIONS, RECURSION AND CONTROL FLOW (IF-ELSE, FOR, WHILE)

06

CLASSES AND OBJECTS

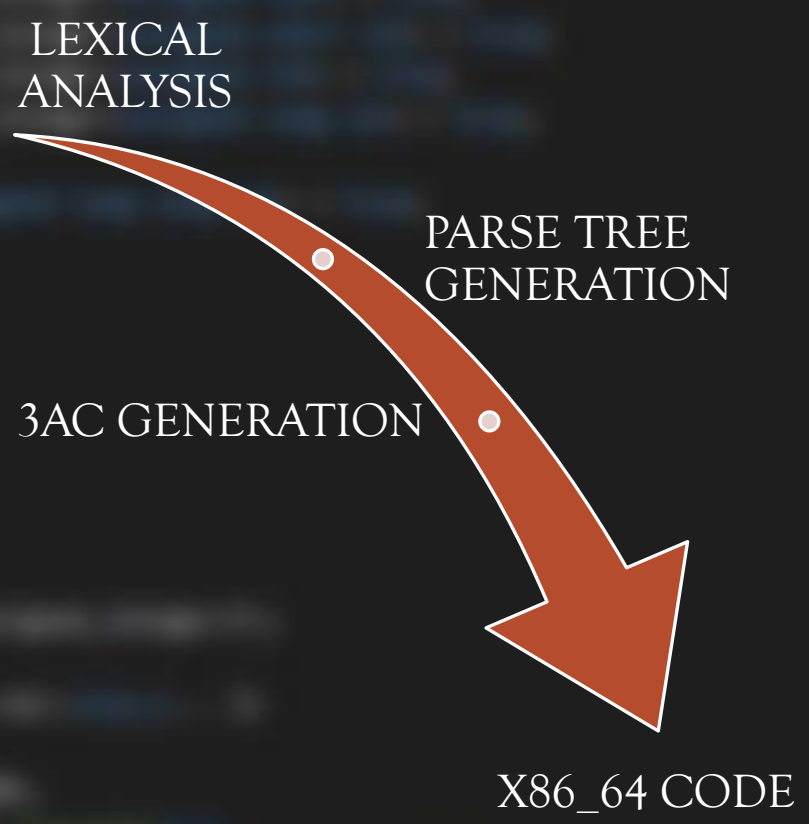
07

SUPPORT FOR PRINTLN

Assumptions & Limitations

- println_x() function is used to print variable x's value, instead of default way
- For printing an array value or value obtained from method invocation, create a temporary variable, say x, then store that value in x and then use println x()
- Temporary variables are not stored, so before calling function expression or while accessing/ storing array at an index, store that function call or expression in a variable and then provide it later
- We are not creating any class object by default, So to use variables of class(i.e global variables) a object needs to be created by constructor. Also to access these variables "this" keyword is must. E.g this.x for global variable x. Whereas these points are optional in actual java compiler

Procedure


LEXICAL ANALYSIS
PARSE TREE GENERATION
3AC GENERATION
X86_64 CODE

Example Test Case

cs335_project_200530 > testcases > test_1.java

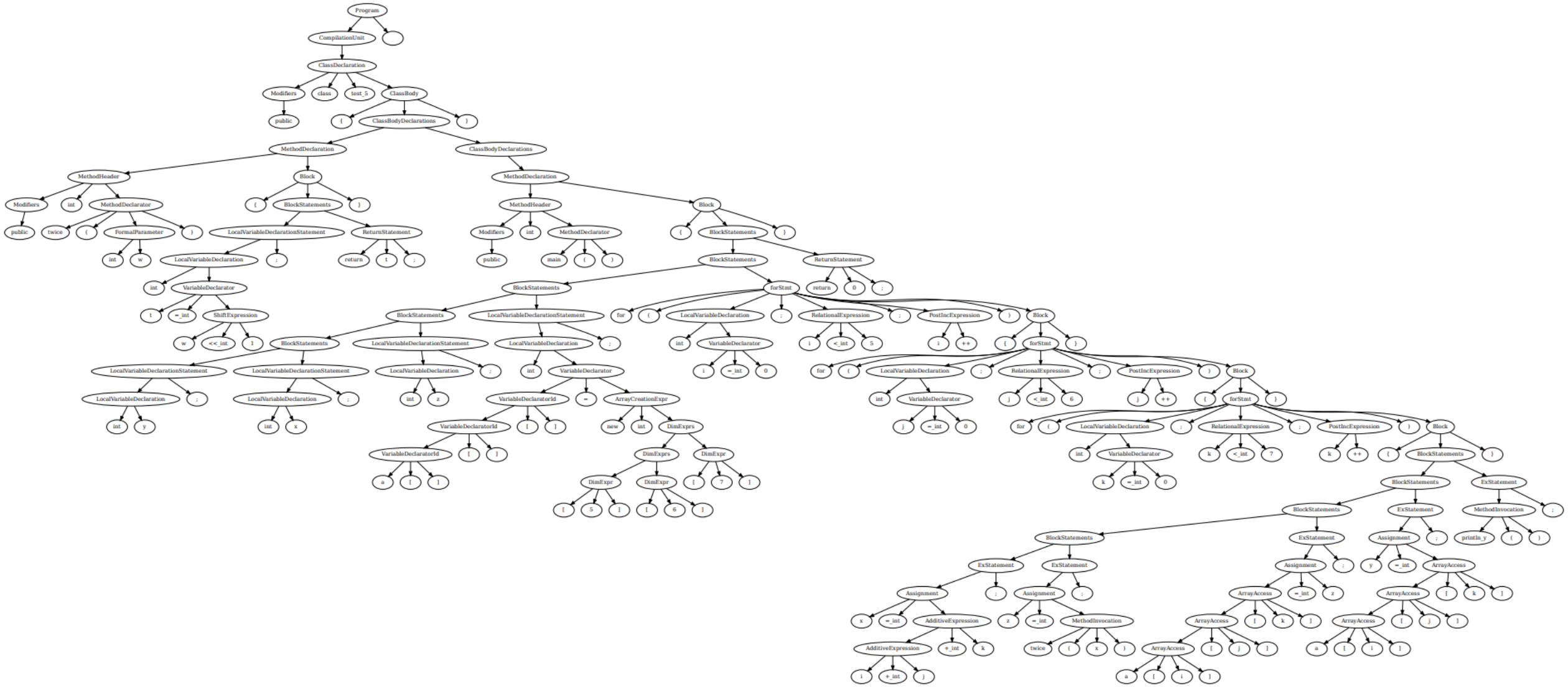
```java
public class test_5 {
    public int twice(int w){
        int t = w<<1;
        return t;
    }
    public int main(){

        int y;
        int x;
        int z;
        int a[][][] = new int[5][6][7];
        for(int i=0; i<5;i++){
            for(int j=0; j<6;j++){
                for(int k=0; k<7;k++){
                    x = i+j+k;
                    z = twice(x);
                    a[i][j][k] = z;
                    y = a[i][j][k];
                    println_y();
                }
            }
        }

        return 0;
    }
}
```

Array
Function
Loops
Printing

Parse Tree


Program
CompilationUnit
ClassDeclaration
Modifiers
class
test_5
ClassBody
public
{
ClassBodyDeclarations
}
MethodDeclaration
ClassBodyDeclarations
MethodHeader
Block
Modifiers
int
MethodDeclarator
{
BlockStatements
}
public
twice
(
FormalParameter
)
LocalVariableDeclarationStatement
ReturnStatement
int
w
LocalVariableDeclaration
;
return
t
;
int
VariableDeclarator
t
=_int
ShiftExpression
w
<<_int
1
MethodDeclaration
MethodHeader
Block
Modifiers
int
MethodDeclarator
{
BlockStatements
}
public
main
(
)
BlockStatements
ReturnStatement
return
0
;
BlockStatements
forStmt
BlockStatements
LocalVariableDeclarationStatement
for
(
LocalVariableDeclaration
;
RelationalExpression
;
PostIncExpression
)
Block
BlockStatements
LocalVariableDeclarationStatement
LocalVariableDeclaration
;
int
VariableDeclarator
i
<_int
5
i
++
{
forStmt
}
LocalVariableDeclarationStatement
LocalVariableDeclarationStatement
LocalVariableDeclaration
;
int
VariableDeclarator
i
=_int
0
for
(
LocalVariableDeclaration
;
RelationalExpression
;
PostIncExpression
)
Block
LocalVariableDeclaration
;
LocalVariableDeclaration
;
int
z
int
VariableDeclarator
int
VariableDeclarator
j
<_int
6
j
++
{
forStmt
}
int
y
int
x
VariableDeclaratorId
=
ArrayCreationExpr
j
=_int
0
for
(
LocalVariableDeclaration
;
RelationalExpression
;
PostIncExpression
)
Block
VariableDeclaratorId
[
]
new
int
DimExprs
int
VariableDeclarator
k
<_int
7
k
++
{
BlockStatements
}
VariableDeclaratorId
[
]
DimExprs
DimExpr
k
=_int
0
BlockStatements
ExStatement
a
[
]
DimExpr
DimExpr
[
7
]
BlockStatements
ExStatement
MethodInvocation
;
[
5
]
[
6
]
BlockStatements
ExStatement
Assignment
;
println_y
(
)
BlockStatements
ExStatement
Assignment
;
y
=_int
ArrayAccess
ExStatement
ExStatement
Assignment
;
Assignment
;
ArrayAccess
=_int
z
ArrayAccess
[
k
]
x
=_int
AdditiveExpression
z
=_int
MethodInvocation
ArrayAccess
[
k
]
ArrayAccess
[
j
]
AdditiveExpression
+_int
k
twice
(
x
)
ArrayAccess
[
j
]
a
[
i
]
i
+_int
j
a
[
i
]

3AC Generation

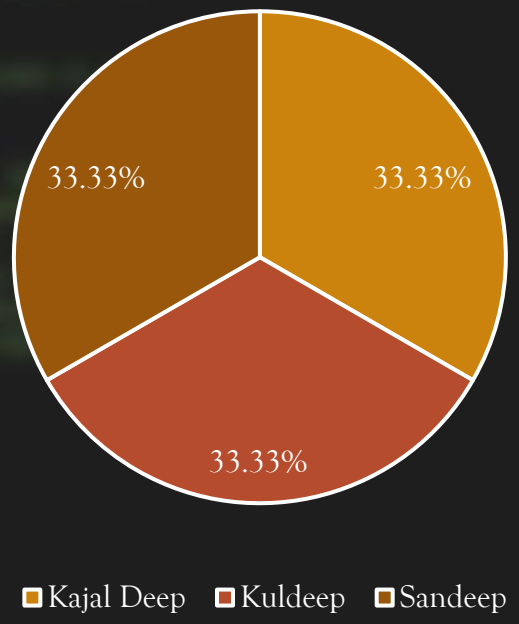
cs335_project_200530 > src > 3ac.txt

```
test_5.main:
    t2_0: BeginFunc
    t2_1: stackPointer-=  32  // Manipulating stack (equal to size of function)
    t2_2: =  1680    _t1
    t2_3: param  _t1
    t2_4: call  allocmem  1  _v91
    t2_5: =  _v91    a
    t2_6: =  0    i
    t2_7: <_int  i  5  _v109
    t2_8: ifFalse  _v109  goto  t2_51
    t2_9: =  0    j
    t2_10: <_int  j  6  _v128
    t2_11: ifFalse  _v128  goto  t2_48
    t2_12: =  0    k
    t2_13: <_int  k  7  _v147
    t2_14: ifFalse  _v147  goto  t2_45
    t2_15: +_int  i  j  _v159
    t2_16: +_int  _v159  k  _v162
    t2_17: =_int  _v162    x
    t2_18: PushParam  x
    t2_19: LCall  twice
    t2_20: returnRegister  _v172
    t2_21: RestoreMachineState  //Adjust Base Pointer to previous base pointer and reload registers
    t2_22: stackPointer+=  8  // Remove parameters passed into stack
    t2_23: =_int  _v172    z
    t2_24: =  i    _v179
    t2_25: *  _v179  7  _v179
    t2_26: +  _v179  j  _v179
    t2_27: *  _v179  6  _v179
    t2_28: +  _v179  k  _v179
    t2_29: ArrayAddress  a[_v179]    _v189
    t2_30: ArrayAssign  z    _v189
    t2_31: =  i    _v200
    t2_32: *  _v200  7  _v200
    t2_33: +  _v200  j  _v200
    t2_34: *  _v200  6  _v200
    t2_35: +  _v200  k  _v200
    t2_36: ArrayAddress  a[_v200]    _v210
    t2_37: ArrayAccess  _v210    y
    t2_38: LCall  println_y
    t2_39: returnRegister  _v218
    t2_40: RestoreMachineState  //Adjust Base Pointer to previous base pointer and reload registers
    t2_41: stackPointer+=  0  // Remove parameters passed into stack
    t2_42: =  k    _v151
    t2_43: ++  k    k
    t2_44:     goto  t2_13
    t2_45: =  j    _v132
    t2_46: ++  j    j
    t2_47:     goto  t2_10
    t2_48: =  i    _v113
    t2_49: ++  i    i
    t2_50:     goto  t2_7
    t2_51: Return  0  // load return value to return register...
    t2_52: EndFunc
test_5.twice:
    t3_0: BeginFunc
    t3_1: stackPointer-=  16  // Manipulating stack (equal to size of function)
    t3_2: getparam      w
    t3_3: <<_int  w  1  _v24
    t3_4: =  _v24    t
    t3_5: Return  t  // load return value to return register...
    t3_6: EndFunc
```

X86_64 assembly code

```
    .global main
    .data
    .text
main:
    pushq %rbp
    mov %rsp, %rbp      # beginFunc
    sub $48, %rsp       # stackPointer-= 32
    mov $1680, %r8     # _t1=1680
    mov %r8, %rdi     # Load Param _t1
    call malloc
    mov %rax, %r9     # Get Refernce from return reg in  _v91
    mov %r9, %r12     # a=_v91
    mov %r12, -32(%rbp)        # Set a in stack
    mov $0, %r13     # i=0
    mov %r13, -40(%rbp)        # Set i in stack
t2_7:
    mov -40(%rbp) ,%r14       # Load i from stack
    cmp $5, %r14
    jl t2_7t
    mov $0, %r10
    jmp t2_7f
t2_7t:
    mov $1, %r10
t2_7f:
    cmp $1, %r10
    jne t2_51
    mov $0, %r12     # j=0
    mov %r12, -48(%rbp)        # Set j in stack
t2_10:
    mov -48(%rbp) ,%r13       # Load j from stack
    cmp $6, %r13
    jl t2_10t
    mov $0, %r11
    jmp t2_10f
t2_10t:
    mov $1, %r11
t2_10f:
    cmp $1, %r11
    jne t2_48
    mov $0, %r14     # k=0
    mov %r14, -56(%rbp)        # Set k in stack
t2_13:
    mov -56(%rbp) ,%r12       # Load k from stack
    cmp $7, %r12
    jl t2_13t
    mov $0, %r15
    jmp t2_13f
t2_13t:
    mov $1, %r15
t2_13f:
    cmp $1, %r15
    jne t2_45
    mov -40(%rbp) ,%r13       # Load i from stack
    mov -48(%rbp) ,%r14       # Load j from stack
    mov %r13, %rcx
    add %r14, %rcx      #   _v159 = i + j
    mov -56(%rbp) ,%r12       # Load k from stack
    mov %rcx, %r8
    add %r12, %r8      #   _v162 = _v159 + k
    mov %r8, %r13     # x=_v162
    mov %r13, -16(%rbp)        # Set x in stack
    mov -16(%rbp) ,%r14       # Load x from stack
    sub $8, %rsp       # PushParam x
    pushq %r14       # PushParam x
    call twice
    mov %rax, %r9     # ReturnValue = _v172
    add $16, %rsp       # stackPointer+= 8
    mov %r9, %r12     # z=_v172
    mov %r12, -24(%rbp)        # Set z in stack
    mov -40(%rbp) ,%r13       # Load i from stack
    mov %r13, %r10     # _v179=i
    mov %r10, %r10
    imul $7, %r10      #   _v179 = _v179 * 7
    mov -48(%rbp) ,%r14       # Load j from stack
    mov %r10, %r10
    add %r14, %r10      #   _v179 = _v179 + j
    mov %r10, %r10
    imul $6, %r10      #   _v179 = _v179 * 6
    mov -56(%rbp) ,%r12       # Load k from stack
    mov %r10, %r10
    add %r12, %r10      #   _v179 = _v179 + k
    imul $8, %r10        # addr * 8
    mov -32(%rbp) ,%r13       # Load a from stack
    add %r10, %r13     # addr + base
    mov %r13, %r11
    mov -24(%rbp) ,%r14       # Load z from stack
    mov %r14, (%r11)     # Array assign
    mov -40(%rbp) ,%r12       # Load i from stack
    mov %r12, %r15     # _v200=i
    mov %r15, %r15
    imul $7, %r15      #   _v200 = _v200 * 7
    mov -48(%rbp) ,%r13       # Load j from stack
    mov %r15, %r15
    add %r13, %r15      #   _v200 = _v200 + j
    mov %r15, %r15
    imul $6, %r15      #   _v200 = _v200 * 6
    mov -56(%rbp) ,%r14       # Load k from stack
    mov %r15, %r15
    add %r14, %r15      #   _v200 = _v200 + k
    imul $8, %r15        # addr * 8
    mov -32(%rbp) ,%r12       # Load a from stack
    add %r15, %r12     # addr + base
    mov %r12, %rcx
    mov (%rcx), %r13     # Array access
    mov %r13, -8(%rbp)        # Set y in stack
    sub $32, %rsp       # For print statement
    mov $format, %rdi
    mov -8(%rbp) ,%r14       # Load y from stack
    mov %r14, %rsi
    call printf
    add $32, %rsp       #For print stmt
    add $0, %rsp       # stackPointer+= 0
    mov -56(%rbp) ,%r12       # Load k from stack
    mov %r12, %r8     # _v151=k
    mov -56(%rbp) ,%r13       # Load k from stack
    add $1, %r12      #   k = k + 1
    mov %r12, -56(%rbp)        # Set k in stack
    jmp t2_13
t2_45:
    mov -48(%rbp) ,%r14       # Load j from stack
    mov %r14, %r9     # _v132=j
    mov -48(%rbp) ,%r12       # Load j from stack
    add $1, %r12      #   j = j + 1
    mov %r12, -48(%rbp)        # Set j in stack
    jmp t2_10
t2_48:
    mov -40(%rbp) ,%r13       # Load i from stack
    mov %r13, %r10     # _v113=i
    mov -40(%rbp) ,%r14       # Load i from stack
    add $1, %r13      #   i = i + 1
    mov %r13, -40(%rbp)        # Set i in stack
    jmp t2_7
t2_51:
    mov $0, %rax     # Return 0
    jmp return_main
return_main:
    mov %rbp, %rsp
    popq %rbp
    ret       # EndFunc
twice:
    pushq %rbp
    mov %rsp, %rbp      # beginFunc
    sub $32, %rsp       # stackPointer-= 16
    mov 16(%rbp) , %r11     #   Get Argument w
    mov %r11, -8(%rbp)     #   Store Argument w in stack
    mov -8(%rbp) ,%r12       # Load w from stack
    mov %r12, %rax
    sal $1, %rax
    mov %rax, %r15
    mov %r15, %r13     # t=_v24
    mov %r13, -16(%rbp)        # Set t in stack
    mov -16(%rbp) ,%r14       # Load t from stack
    mov %r14, %rax     # Return t
    jmp return_twice
return_twice:
    mov %rbp, %rsp
    popq %rbp
    ret       # EndFunc
format:
    .asciz  "%d\n"
```

Effort Chart



Thank You!

Project compiled successfully.....