Verification Plan

Verification Plan
$Revision: 1.1 $ Edition
Published $Date: 1999/12/16 15:04:37 $

Verification of the OddBall compiler will be tightly integrated with the development of the product. As each component is finished a series of tests will be performed to unit test that specific component. At the conclusion of development all the unit tests will be repeated on the product as a whole and these tests will be presented as part of a final verification document.

Table of Contents

Chapter 1. Guidelines

This initial plan for verification we will identify and implement methods on how testing should be done. The following is a list of guidelines that the group has come up with:

- Methods for unit testing will be determined by the developer.
- Each developer will be required to supply code that tests various aspects of their solution.
- There will be a description at the top of each test case file that describes the test.
- After a test is run the conclusion will be included with the OddBall source.
- Final verification tests will be performed in a specific order to be described.

Chapter 2. Types of tests

Input/Output tests: The main aim of these is to test the various I/O functions e.g. the "read" statement, "write" statement and the "writeln" statement. These tests will see if they perform as defined and produce corresponding error results if incorrect code is given.

Expression tests: These tests are used to test the various aspects of expressions. These include the use of arithmetic and relational operators, the use of arrays, unary operators, expressions in assignment statements etc.

Conditionals tests: Testing of the various conditional statements e.g. the "cond....runs" statement, "cond...loops" statement, to see that the branching is occurring properly, each statement is reached, testing of nested conditions, check for matching "stops" for their corresponding "otherwise".

Functions: These tests will aim at testing the various features of functions, their definitions, proper passing of number and type of arguments, their usage in code and to see if they are returning the proper value or not.

Final complete Programs: These test cases include complete programs that test some/all of the above defined features to get the desired results. Since these test cases try to implement the various aspects of the language together, they are good means of testing the over all language features.

Chapter 3. Ordering of tests

Since these test the different functionality of the language, starting from the simple to the advanced features they have to be tested in a specific order. The order in which we execute the test cases are:

- I/O tests
- Expression tests
- Conditionals
- Functions
- Final complete Programs

I/O are the fundamental tests that are run first. Next we run the expression tests which test the various aspects of expressions described above. These are run after the I/O tests as we may have some variables in the expression whose values may be inputted or outputted. Conditional tests are run next as they may have expressions along with logical operators in the "cond" part. Then we go ahead with the functional tests as it may involve all of the above mentioned aspects of the language, and finally we execute full time programs.

Chapter 4. Error cases

The various error messages of the language are given below. They are classified based upon the type of the error.

4.1. I/O Errors

The various I/O errors that may occur are listed below:

i. expected a variable following READ keyword: This error occurs when a read statement is not followed by a variable in which the value has to be stored.

   Example:

   ```
   read
   ```

ii. expecting '[' or '.' following READ keyword: This error occurs when we are trying to read in more than one variable using single read statement as Oddball allows reading of only one variable using a read statement.

   Example: If we give a read statement as:

   ```
   read a,b,c.
   ```

iii. un-terminated string found: This error occurs when the writeln statement is not properly terminated or given incorrectely.

   Example: If we give a writeln statement as:

   ```
   writeln "Hello world.
   ```

   Similarly we get error messages if the read or write statement is not terminated by a period.

4.2. Expression Errors

Some of the errors that come under expression are listed below with a brief description of the error.

i. missing operand in expression: This error occurs when an expression has improper sequencing of operators and/or operands.

   Example:

   ```
   p+q*->r.
   ```

ii. was expecting "." at end of expression but found ")" instead: This error occurs when we have an unbalanced parenthesis in the expression.

   Example:

   ```
   (a+b)*(c-d)).
   ```

iii. invalid r-value "3". Values can only be assigned to scalar variables: This error occurs when we are trying to assign the value of an expression to a integer constant instead of a variable on the right hand side of the assignment operator.

   Example:

   ```
   x+y*z -> 3.
   ```

iv. operand must be a scalar instead of an array: This error occurs when we define a variable as scalar type and try to use that as an array type variable

   Example: If we declare:

   ```
   var a, b,c:integer.
   ```

   and give an expression as shown below then this error is generated:

   ```
   a[2]+b->c
   ```

4.3. Conditional Errors

The various conditional errors that may occur with regard to conditional statements are.

i. expecting a '.' following STOP keyword: This error occurs when the STOP keyword is not followed by either a period or a OTHERWISE keyword when we are using a COND statement.

ii. expecting RUNS following OTHERWISE keyword: This error occurs if the OTHERWISE keyword is not followed by the RUNS keyword in the COND statement.

4.4. Functional Errors

These errors occur when the functions are not defined or called properly with respect to their name, number of parameters, type of parameters etc. Some of the errors in this category are:

i. was expecting (to start parameter list to call of function: This error occurs if the function name is not followed by a left parenthesis to give the parameters that are to be passed to the calling function.

ii. invalid function parameter: This error occurs if the parameter passed while calling the function is not a valid one.

iii. type of parameter in call to function does not match the type of the receiving array parameter: This error occurs if the type of the parameters passed while calling a function does not match with the type of parameters declared in the function definition.

iv. was expecting) to end parameter list to call function : This error occurs if the parameter list in the calling function is not terminated by the right parenthesis.

Chapter 5. Test suites

The test suits that test the various aspects of the language are given at the end of the user manual. The tests are classified based on the category of the language they are testing. Each test has its own aim that tells what it tries to test, and a conclusion is included with each test file that summarizes the result of the test.

Chapter 6. Test Results

6.1. io1.odd

```
$<
Verification test.

Name: Input/Output test.

Description: This tests the input and output statemnets of the OddBall
language.

Result: The C source code was generated correctely and the executable are
both verified and found to be correct and working .

Tester(s): Prashanth and  Jidesh.
>$

define program() runs
   var a,b,c:integer.
stop.

   writeln "This is the first test program for oddball I/O.".
stop.
```

6.2. io2.odd

```
$<
Verification test.

Name: Input test.

Description: This test sees if we can read more than one input variable
using a single read statement.

Result: The compiler gave the appropriate error as expecetd as Oddball
does not allow reading of multiple variables using single read statemnat.
No C code was generated.
```

```
Tester(s): Prashanth and Jidesh.
>$

define program() runs
   var a,b,c:integer.
stop.

  writeln "Enter three numbers (one on each line):".
  read a.
  read b.
  read c.

  writeln a.
  writeln b.
  writeln c.

stop.
```

6.3. io3.odd

```
$<
Verification test.

Name: Input/Output test.

Description: This tests the input and output statements of the OddBall
language. Testing multiple entries on a writeln and write.

Result: The C source code was generated correctely and the executable are
both verified and found to be correct and working .

Tester(s): Prashanth and Jidesh.
>$

define program() runs
   var a,b,c,d,e:integer.
       k,l,m:character.
stop.

  5->a->b->c.
  10->d->e.
  writeln a,b,c.
```

```
  write d,e.

stop.
```

6.4. chararr1.odd

```
$<
Verification test.

Name:testing of character variables.

Description: This program tries to test the character data type of the
language. We assign a character to each element of the character array and
then prints outs them

Result: The character data types were handled properly and the requried
output was obtained.

Tester(s): Prashanth and Jidesh.
>$

define program() runs
  var   i:integer.
name of 20:character.
stop.

   'V'->name[1].
   'i'->name[2].
   'r'->name[3].
   'g'->name[4].
   'i'->name[5].
   'n'->name[6].
   'i'->name[7].
   'a'->name[8].
   'T'->name[9].
   'e'->name[10].
   'c'->name[11].
   'h'->name[12].

    1->i.
    cond(i < 13) loops
writeln  name[i].
```

```
i+1->i.
    stop.
stop.
```

6.5. exparrwarr.odd

```
$<
Verification test.

Name: Expression with arrays as array elements.

Description: This program tries have arrays within arrays and see if the
compiler handles them or not.

Result: The compiler handled the nested arrays properly and the
corrosponding C code and the executable was generated which ran correctly.

Tester(s): Prashanth and Jidesh.
>$

define program() runs
    var i,j,k,max,min:integer.
    a of 10, b of 25:integer.
stop.
  5->b[5].
  35->a[b[5]].

  writeln a[b[5]].
stop.
```

6.6. expression1.odd

```
$<
Verification test.

Name: Simple expression test test.

Description: This is a simple test for testing arithematic expressions in
oddball.
```

```
Result: The expression was validated by the compiler and the C code was
generated that compiled properly generating the executable.

Tester(s): Prashanth and Jidesh.
>$
define program() runs
  var i,j,k,l,m:integer.
stop.
   1->i.
   2->j.
   -3->k.
   -5->m.
   i + j * k / m -> l.
   writeln l.
stop.
```

6.7. parenthesis.odd

```
$<
Verification test.

Name: Parenthesis test.

Description: This program tests the use of parenthesis in an arithematic
expression.We test to see if the compiler handles the parenthesis matching
properly or not.

Result: The compiler handled the parenthesis properly and generated the C
code. The executable are both verified to be correct and working.
It also gave the corrosponding error message if the parenthesis are not
balanced.

Tester(s): Prashanth and Jidesh.
>$

define program() runs
   var a,b,c,d:integer.
stop.
  1->a->b->c->d.
  ((a+b)*(c-b))/((2*(a+b)+c)-d)->d.
  writeln d.
```

```
stop.
```

6.8. uniaryoperator.odd

```
$<
Verification test.

Name: Uniary operator test.

Description: This program tests the use and proper functioning of uniary
operators like uniary plus and uniary minus.

Result: The uniary operator was handled properly and  the C source code
was generated correctely and the executable are both verified and found to
be correct and working .

Tester(s): Prashanth and Jidesh.
>$

define program() runs
   var a,b,c,d:integer.
stop.
  1->a->b->c->d.
  (a+b)*(-(c+b))->d.
  writeln d.
stop.
```

6.9. cond1.odd

```
$<
Verification test.

Name:Testing of a simple conditional statement.

Description: This program tries to test a simple cond statement.

Result: The code was compiled properly by the compiler and the C
code and the executable was generated that was working correctly.

Tester(s): Prashanth and Jidesh.
```

```
>$
define program() runs
   var a,b,max:integer.
 stop.

  writeln "Please enter A:".
  read a.
  writeln "Please enter B:".
  read b.

   cond (a>b) runs
          writeln "A is greater than B".
   stop otherwise runs
          writeln "B is greater than or equal to A".
   stop.
stop.
```

6.10. condruns1.odd

```
$<
Verification test.

Name: Testing of single level cond...loops statement.

Description: This program tries to test the cond...loops statemant at
single level.

Result: The compiler handled  cond statements properly and the
corrosponding if statement in the C code was generated correctly.

Tester(s): Prashanth and Jidesh.
>$
define program() runs
   var a,b:integer.
stop.

   5->b.
   20->a.

   cond(a>b) loops
        b+1->b.
```

```
   stop.

   writeln b.

stop.
```

6.11. embedded_loops.odd

```
$<
Verification test.

Name: Testing of embedded level cond...loops statement.

Description: This program tries to test the cond...loops statemant at
an embedded level.

Result: The compiler handled  cond statements properly and the
corrosponding if statement in the C code was generated correctly.

Tester(s): Prashanth and Jidesh.
>$
define program() runs
   var i, j, k, l, m:integer.
   stop.

   0->i->j->k->l->m.

   cond (i < 5) loops
      cond (j < 5) loops
         cond (k < 5) loops
            cond (l < 5) loops
               m + 1 -> m.
               l + 1 -> l.
            stop.
            0 -> l.
            k + 1 -> k.
         stop.
         0 -> k.
         j + 1 -> j.
      stop.
      0 -> j.
      i + 1 -> i.
```

```
   stop.

   writeln m.

stop.
```

6.12. recursion.odd

```
$<
Verification test.

Name: Recursion test.

Description: This program generates the first 10 natural numbers using
the recursion function. This is a code for testing recursion on Oddball.

Result: The recursion function was handled properly by Oddball and the
natural numbers were generated using recursion.

Tester(s): Prashanth and Jidesh.
>$

define function(i:integer) runs
   var j:integer.
   stop.
   cond (i < 10) runs
      i + 1 -> i.
      writeln i.
      function(i)->function.
   stop.
stop.

define program() runs
   var i:integer.
   stop.

   0->i.

   function(i).

stop.
```

6.13. sum.odd

```
$<
Verification test.

Name: Function call test.

Description: This program tests the definition and calling of a function
from the main program. This program reads in two values and calls a function
that adds the two values and returns the sum.

Result: The function definition and calling was handled correctely and the
C code was generated. The executable are  verified to be correct and
working.

Tester(s): Prashanth and Jidesh.
>$

define add (i,j:integer) runs
  var a:integer.
stop.
 i+j->a.
 a->add.
stop.


define program () runs
  var i,j,k:integer.
stop.

  writeln "Enter the first value".
  read i.
  writeln "Enter the second value".
  read j.
  add(i,j)->k.
  writeln "The sum of the two numbers is ", k.
stop.
```

6.14. aveminmax.odd

```
$<
Verification test.

Name: Calculation of average, minimum and maximum.

Description: This program calculates the average of a given list of
numbers and also finds the minimum and maximum of the list of numbers.

Result: The program compiled correctly  and  the C source code was
generated and the corrsponding executable file was generated.

Tester(s): Prashanth and Jidesh.
>$

define average(list of 35,n:integer) runs
  var sum,ave,i:integer.
stop.

  1->i.
  0->sum.
  cond(i <= n) loops
     sum+list[i]->sum.
     i+1->i.
  stop.

  sum / n -> ave.

  writeln "The average of the given list of numbers is ",ave.
stop.



define minmax(list of 35,n:integer) runs
  var min,max,i:integer.
stop.
  1->i.
  0->max.
  65535->min.
  cond(i <= n) loops
     cond(list[i] > max) runs
          list[i]->max.
     stop.
     cond(list[i] < min) runs
```

```
          list[i]->min.
     stop.
     i+1->i.
  stop.
  writeln "The minimum of the given numbers is ", min.
  writeln "The maximum of  the numbers is ", max.
stop.

define program() runs
  var i,n,num,list of 35:integer.
stop.

  writeln "Enter the number of elements in the search list:".
  read n.
  1->i.
  cond(i < n+1) loops
     writeln "Enter element ",i," :".
     read list[i].
     i+1->i.
  stop.

 average(list,n).
 minmax(list,n).
stop.
```

6.15. fact1.odd

```
$<
Verification test.

Name: Search test.

Description: This program calculates the factorial of a given number.

Result: The program compiled correctely  and  the C source code was
generated that calculated the factorial of the number correctely.

Tester(s): Prashanth and Jidesh.
>$

define program() runs
```

```
  var i,j,n,fact,prod:integer.
stop.

  writeln "Enter the number whose factorial is to be calculated:".
  read n.
  1->j.
  2->i.
  n->prod.
  cond(n<0) runs
  writeln "Entered value is a negative number.Enter a positive number".
  stop
  otherwise runs
       cond(n=0) runs
       writeln "The factorial of 0 is 1".
       stop
       otherwise runs
           cond(n=1) runs
           writeln "The factorial of 1 is 1".
           stop
           otherwise runs
              cond(i<n) loops
                ((n-j) * prod) -> prod.
                j+1->j.
                i+1->i.
              stop.
           writeln "The factorial of the given number is ",prod.
           stop.
       stop.
  stop.
stop.
```

6.16. fib1.odd

```
$<
Verification test.

Name: Fibonacci series generation.

Description: This program generates first elements of the fibonacci
series.This is another test for testing the various language features.
```

```
Result: The program compiled correctely and the C source code was
generated and the executable generated the requried elements of the
series.

Tester(s): Prashanth and Jidesh.
>$
define program() runs
  var i,j,k,n,count:integer.
stop.

   0->i.
   1->j.
   writeln " Enter the number of elements of the series to be generated".
   read n.
   writeln "The first", n, "elements of the fibnocci series are".
   writeln i.
   writeln j.
   2->count.
   cond(count<n) loops
      i+j->k.
      j->i.
      k->j.
      count+1->count.
      writeln k.
   stop.
stop.
```

6.17. genprime.odd

```
$<
Verification test.

Name: Prime number generation.

Description: This program generates the first n prime numbers where n is
specified by the user. This basically tests several aspects of the
language like the cond statements, arithematic expressions.

Result: The program compiled correctely  and  the C source code was
generated. The executable are both verified and found to be correct
and generating the requried number of prime numbers.
```

```
Tester(s): Prashanth and Jidesh.
>$
define program() runs
   var i,k,rem,flag,n,count,num:integer.

stop.

  writeln "Enter the number of primes to be generated:".
  read n.
  writeln "The first ", n, " prime numbers are:".
  write "2, 3".
  2->count.
  4->num.
  cond(count<n) loops
     0->flag.
     num / 2 -> k.
     2->i.
     cond(i <= k) loops
         cond(flag=0) runs
              num % i->rem.
              cond(rem=0) runs
                  1->flag.
              stop.
          stop.
          i+1->i.
      stop.
      cond(flag=0) runs
           write ", ",num.
           count+1->count.
      stop.
      num+1->num.
  stop.
  writeln.
stop.
```

6.18. globalwfunc.odd

```
$<
Verification test.

Name: Global variables with functios.
```

```
Description: This program is used to test the scope of local and global
variables. Here we try to access a variable that is local to a function in
another function.

Result: The compiler compiled properly giving the corrosponding error
indicating invalid access of variables. No C code was generated as a
result of compilation error.

Tester(s): Prashanth and Jidesh.
>$
var a,b:integer.
stop.

define program() runs
  var i,j,k:integer.
stop.

   i+(a-b)*j->k.
   writeln k.
stop.

define func1() runs
  var x,y,z:integer.
stop.
  a+x->y.
  y+i->z.
stop.
```

6.19. prime.odd

```
$<
Verification test.

Name: Prime number testing.

Description: This program tests if a given number is a prime number or
not. This uses the cond statements of the language.

Result: The program compiled correctely  and  the C source code was
generated. The executable are both verified and found to be correct
and working .
```

```
Tester(s): Prashanth and Jidesh.
>$

define program() runs
   var i,k,rem,flag,n:integer.

stop.

  writeln "Enter the number which has to be checked :".
  read n.
  0->flag.
  n / 2 -> k.
  2->i.
  cond(i<k) loops
     cond(flag=0) runs
         n % i->rem.
         cond(rem=0) runs
             1->flag.
         stop.
     stop.
     i+1->i.
  stop.

  cond(flag=1) runs
    writeln "The given number is not a prime number".
  stop
  otherwise runs
    writeln "The given number is a prime number".
  stop.
stop.
```

6.20. search.odd

```
$<
Verification test.

Name: Search test.

Description: This program searches for a given number in list of numbers.
This basically tests the use of cond statements of the language.

Result: The program compiled correctely  and  the C source code was
```

```
generated  and the executable are both verified and found to be correct
and working .

Tester(s): Prashanth and Jidesh.
>$

define program() runs
  var n,i,num,list of 35:integer.
      flag,pos:integer.
stop.

  writeln "Enter the number of elements in the search list:".
  read n.
  1->i.
  cond(i < n+1) loops
     writeln "Enter element ",i," :".
     read list[i].
     i+1->i.
  stop.

  writeln "Enter the number to be searched:".
  read num.
  0->flag.
  1->i.
  cond(i <= n) loops
     cond(list[i]=num) runs
          1->flag.
          i->pos.
     stop.
     i+1->i.
  stop.
  cond(flag=1) runs
        writeln "Search successful".
   writeln "The number is present in ",pos,"th position in the list".
  stop
  otherwise runs
        writeln "Search failed".
    writeln "The number is not present in the list".
  stop.
stop.
```