LRM

# BELL

Columbia University
COMS W4115 Programming Languages and Translators

Fall 2007
Professor Stephen Edwards


| | |
|---|---|
| Alicia Boyzk | amb2129@columbia.edu |
| Yousry ElMallah | yae2103@columbia.edu |
| Robert Lin | rcl2106@columbia.edu |
| Carlene Liriano | cl2294@columbia.edu |

**CONTENTS**

## 1. Introduction

BELL is an interpreted language that is designed to simplify the creation of web pages. It will include the following features:

- HTML Generation
- Math Functions
- Algorithms
- Graphics Management

The BELL language allows developers to easily create HTML with rich content. It can easily create HTML code that is tedious to produce by hand, such as generating tables and page layouts. BELL also includes practical functions that are commonly used, such as sorting algorithms and complex mathematical functions.

The utility of BELL can be found in the powerful packages that are standard to the language. The functionality BELL provides allows developers to produce high quality code with minimal effort.

## 2. Lexical Conventions

A program consists of tokens grouped together to form declarations and statements. In BELL, each instruction begins with an opening bracket followed by a colon "(:" and ends with a colon followed by a closing bracket ":)". Or, instructions may be grouped inside a single (: :) pair and separated by ; characters. Variables are indicated by preceding the variable name with a @. Similar to the C programming language, BELL consists of six different types of tokens: identifiers, keywords, constants, strings, operators, and other separators. White space, tabs, and comments are used to separate tokens and are ignored otherwise.

### a. Comments

Comments are supported in BELL. A comment is introduced by a ":(" and is terminated by a "):". They can be placed anywhere within the (: :) block.

### b. Identifiers (Names)

Identifiers are sequences of letters, digits, and/or underscores where the first character can only be a letter or an underscore. Identifiers are case sensitive.

### c. Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

| | | | |
|---|---|---|---|
| int | boolean | if | |
| char | string | else | |

| | | |
|---|---|---|
| float | array | while |
| double | NULL | for |

**d. Constants**

There are several kinds of constants, as follows:

**i. Integer constants**

An integer constant is a sequence of digits. An integer is always taken to be decimal. The associated keyword is *int.*

**ii. Floating constants**

A floating constant consists of an integer part, a decimal point, and a fraction part. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (not both) may be missing. Every floating constant is taken to be double precision. The associated keyword is *float.*

**e. Strings**

A string is a sequence of characters surrounded by double quotes “ **"** ”. Strings may consist of any characters except for the double quote, which must be escaped using a back slash (\"). Strings are associated with the keyword *string*.

**f. Other Tokens**

There are a set of single characters that must be used correctly according to BELL’s syntax. They include:

| | | | | | |
|---|---|---|---|---|---|
| { | } | ( | ) | + | - |
| * | / | $ | : | ; | _ |
| ! | % | \| | \ | , | . |

These are also a set of character pairs which must be used correctly according to BELL’s syntax. They include:

| | | | | | |
|---|---|---|---|---|---|
| && | \|\| | == | <= | >= | != |

## 3. Datatypes

**a. boolean** - A Boolean expresses a truth value. It can be either TRUE or FALSE.

**b. int -** An int is a number of the set Z = {…, -2, -1, 0, 1, 2, …}.

**c. float -** float numbers are 4 bytes long.

**d. string -** A string is a series of characters. In BELL, a character is the same as a byte, that is, there are exactly 256 different characters possible.

**e. array** - An array in BELL is like a Java array that is created given a fixed size.

**f. NULL** - NULL value represents that a variable has no value. A variable is considered to be NULL if it has been assigned the constant NULL or it has not been set to any value yet.

## 4. Functions

### a. Function Definition

A BELL function can be defined with the following syntax:

```
return type    function name    ( parameter type   parameter name ) {
        (:  Any BELL code can be inserted here :)
        return value;
}
```

### b. User-Defined Functions

BELL allows users to create their own functions to implement any functionally desired by the user. These functions can be defined using the syntax above. A function that is contained on one file can be used in another file by including it in defined in one file can be used in another file by simply including the file with the function definition in the file where it needs to be used.

```
use   filename;
```

### c. Internal (Built-In) Functions

BELL comes standard with many functions and constructs, providing a lot of flexibility to the developer. Every function will have a manual page that describes function parameters, behaviors, and return values. The following are some examples of some basic BELL functions:

- Print() ( prints a string )
- D.print() ( prints a string )
- Print_r() ( prints the contents of an array )
- Unique() ( determines if the values in a data structure are unique )
- Substr() ( returns a substring given a string)
- Count() ( returns the # of elements in any data structure )
- Include()
- Import() ( imports data from an external source )
- Popup()
- Isset() ( checks if a values is set )

## 5. Expressions

In BELL, expressions are essentially anything that gets assigned a value. Expressions in BELL may be anything from a simple assignment to function calls or relational expressions. Specific examples of those expressions that are supported by BELL follow.

### a. Function Call

A function call in BELL takes the form:

```
Function_identifier(arg1, arg2, …);
```

The function call contains the function name followed by parentheses that may or may not include function arguments, depending on the function definition. When arguments are passed, as in the above example, they are passed as a comma-separated list of expressions.

### b. Unary Expressions

Unary expressions in BELL are those that employ the following operators i: !, ++, --, and (type). These operators are described below.

*$variable= ++expression*

Prefix incrementation operators in BELL assign a value of the expression after the value has been incremented. In the above example, the value of $variable would be (expression + 1).

*$variable= --expression*

Prefix decrementation operators in BELL assign a value of the expression after the value has been incremented. In the above example, the value of $variable would be (expression - 1).

*$variable = expression++*

Post-fix incrementation operators in BELL assign a value to the variable and then increments the expression. In the above example, the value of $variable would be expression. After the assignment is made, expression is incremented by 1.

*$variable = expression--*

Post-fix decrementation operators in BELL assign a value to the variable and then decrements the expression. In the above example, the value of $variable would be expression. After the assignment is made, expression is incremented by 1.

*!expression*

The above example is a logical negation expression. Expression must either return a 0 or 1. If expression has a value of 0, the result is 1. If expression has a value of 1, the result is a 0.

**c. Relational/Comparison Expressions**

The following table summarizes the relational operators supported in BELL.

| Example Expression | Result |
|---|---|
| $a < $b | **TRUE** if $a is strictly less than $b. |
| $a > $b | **TRUE** if $a is strictly greater than $b. |
| $a <= $b | **TRUE** if $a is less than or equal to $b. |
| $a >= $b | **TRUE** if $a is greater than or equal to $b. |
| $a == $b | **TRUE** if $a is equal to $b. |
| $a != $b | **TRUE** if $a is not equal to $b. |

Relational operators group from left-to-right. Relational expressions that evaluate to true, return a result of type **int** with a value of 1. Similarly, those that evaluate to false, return a result also of type **int** with a value of 0.

The equality operators == and != follow the same rules as the relational operators. However, they have lower precedence than the other relational operators.

**d. Logical And/Or Expressions**

The table below summarizes the functionality of the logical operators in BELL.

| Example expression | Result |
|---|---|
| $a && $b | **TRUE, False** otherwise |
| $a \|\| $ | **TRUE** if either $a or $b is **TRUE**. |

In BELL, the logical-and operator takes precedence over the logical-or operator. These logical expressions are typically applied to other logical expressions or relational/comparison expressions. They return either true or false, where true is denoted by 1, and false by 0.

**e. Assignment Expressions**

Assignment expressions in BELL group right-to-left. Assignment expressions take the form of: Identifier –assignment operator- expression;

Assignment operators may be any of those listed in the Operators section. After this assignment expression executes, identifier will contain the value returned by expression.

For arithmetic assignment expressions, either both operands can be of the same type or the operand on the right side of the equation is converted to the type of the operand on the left.

## 6. Statements

Statements in BELL are executed in sequence. A statement in BELL is simply a line of code representing an instruction to the compiler. Much like some of the more popular programming languages that exist today, statements in BELL must be terminated with a ';'. Also, because BELL is geared at web-based functionality, a standard BELL program will have html code interspersed throughout. In order to tell the compiler when to start interpreting the code and when to skip over HTML, statements must be surrounded by opening and closing brackets similar to those found in PHP.

There are several types of statements, including assignment, function call, and return statements. BELL also features iterative and selection statements made up of the standard control-flow structures.

**a. Opening/Closing tags**

Opening and closing tags will delimit where BELL code begins and where BELL code ends and is followed by HTML code. A standard BELL enclosed statement resembles the following:

```
(: BELL code goes here; :)
<p> HTML stuff can follow </p>
(: More BELL stuff to come :)
```

**b. Assignment Statement**

Assignment statements are simply Assignment expressions terminated by a semi-colon and enclosed in BELL brackets. Please se the Assignment Expression example in the previous section.

**c. Selection Statements**

Selection statements are simply conditionals (if/else) that select whether or not a particular statement gets executed. The if-statement may take two forms:

```
(1) if (expr)
        statement
```

-or-

```
(2) if (expr)
        statement
    else
        statement
```

-or-

```
(3) if (expr)
        statement
    else if
        statement
    else
        statement
```

In example 1, if the expression evaluates to true then the statement will get executed, otherwise the compiler will ignore it and move on with the rest of the program.

If-statements may be followed by an else statement, as shown in example 2. Here if the expression evaluates to true the statement directly following the if is executed. Otherwise, the statement following the else is executing and then the program returns to a normal flow of control.

Finally, if-statements may also take the form shown in example 3. If the expression in the if-statement evaluates to true the statement following it is executed, otherwise, control flow is propagated downwards. If the else-if expression is true then that statement is executed and the program returns to its normal flow. Finally, the else will execute if none of the previous statements were executed.

It is important to note that an “else” will always be matched to the nearest if-statement to handle the “dangling else problem.”

**d. Iterative Statements**

The simplest iterative statement in BELL is the while loop. It functions pretty much the same as it does in other common programming languages such as C and Java. A while loop in BELL looks like the following:

```
While(Boolean expr)
      Statement
```

The while loop executes until the Boolean expression is no longer true. Multiple statements can be inside a while loop by enclosing in the scope of the loop (delimited by brackets).

Note also, like any other block of BELL code, iterative statements must be enclosed in the BELL tags.

BELL also makes use of the do-while control structure. The syntax is the following:

```
do {
   do something
} while (Boolean expr);
```

Unlike a while loop, the truth condition is check at the end of each iteration so a do-while loop will always execute at least once.

Another control structure is the for loop

```
for (expr1; expr2; expr3)
        statement
```

In this example, expr1 is an initialization for the loop. Expr2 is a testing condition for the loop to determine whether or not the loop should terminate. This condition is checked before each iteration. Finally, expr3 alters the value of expr1 to eventually cause the loop to reach a termination condition. A programmer may drop any or all of the expressions in the for loop, i.e. a programmer may use a for loop of the form:

```
for (;;)
        statement
```

This, however, is an infinite loop.

**e. Jump statement**

BELL only supports one kind of jump statement - the return statement. The return statement will unconditionally transfer control flow. Typically, return statements will appear at the end of a function definition, so that after a method has executed, the program will to return to its caller along with the value of the expression it is returning. A return-statement should be used when a function returns some value after its completion to its caller. Otherwise, it should be left out.

A return-statement will take the form:

```
return expr;
```

## 7. Bell-Specific Functions

The BELL language possesses unique functionalities that consolidate oft-called functions in typical web design. With simple, single-line functions, users are able to incorporate a rich variety of features and popular mechanics into the page they build.

### a. Math Functions

*convert_measurements( $value, $src_format, $des_format )*
Returns a float. Converts degrees Celsius and Fahrenheit; centimeters and inches; meters and feet; kilometers and miles; kilograms and pounds; milliliters to ounces.

*generate_prime( $lower_bound, $upper_bound )*
Returns an array of prime numbers between given range, inclusive.

*generate_fib( $lower_bound, $ upper_bound )*
Returns an array of Fibonacci numbers between given range, inclusive.

*convert_bases( $value, $src_base, $des_base )*
Returns a float of the new base from the old base.

*random_hour( $lower_bound, $ upper_bound )*
Returns a string of the hour between the lower and upper bound.

*calculate_gcd( $number1, $number2 )*
Returns an int of the greatest common denominator of number1 and number2.

### b. Algorithms

*sort( $array )*
Returns an int of the greatest common denominator of $number1 and $number2.

*md5_encode( $string )*
Returns an md5-encoded string of $string.

*validate_email( $email_address )*
Returns a Boolean of true if $email_address is a valid email address.

add_slashes( $string )
Returns a string with all control characters, single quotes, and double quotes escaped.

time_left( $datetime_string )

Returns a datetime string of tune time remaining to $datetime_string.

### c. HTML

*generate_quote()*
Returns a string of one of Robert's favorite quotes from a remote database.

*array_box( $array )*
Accepts an array and generates the html code for a select box.

*oo_table( [$title, $style, $xml_src] )*
Accepts an array and generates the html code for a table. The table has a CSS style of $style and is populated with the data from $xml_src which also determines the number of rows and columns in the table.

*url_link( $url, $link_name, $target )*
Returns a hyperlink pointing to $url. The title of the hyperlink is $link_name and the target is $target.

*play_mp3( $mp3_path )*
Embeds a Flash audio player in the outputted html file that plays the designated mp3 file at $mp3_path.

*insert_ad( )*
Inserts a banner advertisement from a remote source.

### d. Graphics

*img_gallery( $dir_path )*
Automatically generates an HTML page containing re-sized thumbnails of all of the image files in $dir_path. Clicking on the thumbnails will also open up the image in its original size.

*web_counter( )*
Creates a web counter in the HTML code that counts the number of unique visitors that visit the page.