APL
Application Programmer's Language
Specification
Version 1.0

Dr. K. Muralikrishnan
kmurali@nitc.ac.in
NIT Calicut

April 8, 2013

Contents

1 Introduction

APL or *Application Programmers Language* is a simple and statically typed programming language. The features and constructs of this language are minimal and mainly intended for testing an experimental operating system. The compiler of APL runs on XSM (*eXperimental String Machine*) architecture.

This document describes the programming constructs, syntax and semantics of APL. The structure of APL is similar in some aspects to programming languages like C and Java. A typical APL program is orgnaized in the following way.

```
Global Declarations
...
Function Definitions
...
Main Function
```

2 Lexical Elements

2.1 Comments and White Spaces

APL allows only line comments. Line comments start with the character sequence **//** and stop at the end of the line. White spaces in the program including tabs, newline and horizontal spaces are ignored.

2.2 Keywords

The following are the reserved words in APL and it cannot be used as identifiers.

```
read   print     if       then       else     endif
while    do    endwhile    break    continue  integer
string  main   return      decl     enddecl   Create
 Open   Write    Seek       Read      Close    Delete
 Fork    Exec    Exit    breakpoint
```

2.3 Operators and Delimiters

The following are the operators and delimiters in APL

```
(   )   {    }   [   ]   /   *   +   -   %
>   <   >=   <=  !=  ==  ;   =   &&  ||  !
```

2.4 Identifiers

Identifiers are names of variables and user-defined functions. Identifiers should start with an alphabet, and may contain both alphabets and digits. Special characters are not allowed in identifiers.

```
identifier -> (alphabet)(alphabet | digit)*
```

2.5 Literals

There are integer literals and string literals in APL. An integer literal is a sequence of digits representing an integer. Negative integers are represented with a negative sign preceding the sequence of digits. Any sequence of characters enclosed within double quotes (") are considered as string literals. However APL restricts string literals to size of atmost 16 characters including the '\0' character which is implicitly appended at the end of a string value.

Examples of literals are : `19, -35, "Hello World"`

3 Data Types

3.1 Primitive Types

There are two primitive datatypes in APL.

1. **Integer** : An integer value can range from -32767 to +32768. An integer type variable is declared using the keyword `integer`
2. **String** A string type represents the set of string values. A string value can be atmost 16 characters long. String type variables is declared using the keyword `string`.

3.2 Arrays

Arrays are sequence of elements of a single type. Arrays can be of **integer** or **string** data types. APL allows the use of single-dimensional arrays only, i.e. linear arrays. Array elements are accessed by the array name followed an index value within square brackets (e.g. arr[10]).

4 Declarations and Scope

Declarations should be made for variables and functions defined in the APL program.

4.1 Global Variables

Global variables are declared in the first section of the program within a **decl** ... **enddecl** block. Global variables can be accessed from any function in the program. Global variables can be of integer, string, integer array or string array datatypes. Global variables are declared with its datatype followed by the variable name. If the variable refers to an array the size of the array must be given in square brackets. The general form of declarations is as follows

type variable_name;
type variable_name[size];

4.2 Function Declaration

For every function except the **main()** function defined in a APL program, there must be a declaration. All functions have global scope and is declared in the first section within **decl ... enddecl** block, along with the global variables.

A function declaration should specify the name of the function, the name and type of each of its arguments and the return type of the function. A function can have integer/string arguments. Parameters may be passed by value or reference. Arrays cannot be passed as arguments. If a global variable name appears as an argument, then within the scope of the function, the new declaration will be valid and global variable declaration is suppressed. Different functions may have arguments of the same name. For arguments that are passed by reference, the argument name is preceded by an ampersand(&) in the function declaration. The return type of a function must be either integer or string.

The general form of declarations is as follows

type function_name (type1 argument1,argument2,...; type2 argument1,argument2,...;...);

Examples for global declarations

```
decl
        integer x,y,a[10],b[20];
        integer f1(integer a1,a2; string b1; integer &c1), f2();
        string t, q[10], f3(integer x);
        integer swap(integer &x, &y);
enddecl
```

4.3 Local Variables

Local variables can be declared anywhere inside a function definition except in the body of **if** and **while**. Local variables will have a function scope, i.e. it can only be accessed in the function in which it is declared. Arguments of a function are treated as local variables. Local variables can be integer or string. Arrays cannot be declared locally. All globally declared variables are visible inside a function, unless suppressed by a re-declaration. The general form of declarations is as follows

type variable_name;

5 Function Definition and Main Function

Every APL program must have a **main()** function and zero or more user-defined functions. Every function other than the **main()** function must be declared within the **decl ... enddecl** block. The general form of a function definition is given below

type function_name(ArgumentList)
{

Function Body
}

The function body must contain a return statement and the return value must be of the return type of the function. The arguments and return type of each function definition should match exactly with the corresponding declaration. Every declared function must have a definition. The signature of the function in the delcaration should match the definition of the function which includes the return type, and the names, passing method and datatypes of the arguments. The language supports recursion and static scope rules apply.

5.1 main()

The **main()** function must be a zero argument function of type integer. Program execution begins from the body of the **main()** function. The **main()** function need not be declared. The **main()** function definition follows all user-defined function definitions. The definition part of **main()** should be given in the same format as any other function.

6 Expressions

An expression specifies the computation of a value by applying operators and functions to operands. Function call in APL are treated as expressions, and the value of the expression is its return value. APL supports arithmetic and logical expressions

6.1 Arithmetic Expressions

Any integer value, variable, function returning an integer or 2 or more arithmetic expressions connected by arithmetic operators termed as arithmetic expressions. APL provides five arithmetic operators, viz., +, -, *, / (Integer Division) and % (Modulo operator) through which arithmetic expressions may be combined. Expression syntax and semantics are similar to standard practice in programming languages and normal rules of precedence, associativity and paranthesization hold. APL is strongly typed, and hence the types of the oprands must match the operation.

6.2 Logical Expressions

Logical expressions may be formed by combining arithmetic expressions using relational operators. The relational operators supported by APL are

```
<, >, <=, >=, ==, !=
```

Standard meanings apply to these operators. The operators take two arithmetic expressions as operands and the result will be a boolean value, either of 1(true) or 0(false). Only realational operator that can be applied to two strings is == (to check equality). This also considered as a Logical expression. Logical expressions themselves may be combined using logical operators, && (logical and) , || (logical or) and ! (not).

6.3 Function Call

All functions except the **main()** function can be invoked from any other function including itself. The general form of a function call is

function_name(value1,value1...);

Function calls are treated as expressions. The function takes in the values of its arguments and returns a value of type equal to the return type of the function. This value is treated as the evaluated result of the function call.

7 Statements

Statements control the execution of the program. All statements in APL are terminated with a semicolon ;

7.1 Assignment Statement

The APL assignment statement assigns the value of an expression to a variable, or an indexed array of the same type or a string value to a string variable. **=** is known as the assignment operator. Initialization during declaration is not allowed in APL. The general syntax is as follows
variable_name = string_value / array_variable / expression

7.2 If Statement

If statements specify the conditional execution of two branches according to the value of a boolean expression. If the expression evaluates to true, the **if** branch is executed, otherwise, if present, the **else** branch is executed. The **else** part is optional. The general syntax is as follows

if *(logical expression)* ***then***
    statements;
else
    statements;
endif;

7.3 While Statement

While statement iteratively executes a set of statements based on a condition which is a logical expression. The statements are iteratively executed as long as the logical expression evaluates to true.

while *(logical expression)* ***do***
    statements;
endwhile;

7.4 Break statement

Break statement is a statement which is used in a **while** loop block. This statement stops the execution of the loop in which it is used and passes the control of execution to the next statement after the loop. This statement cannot be used anywhere else other than **while** loop. The syntax is as follows

break ;

7.5 Continue statement

Continue statement is a statement which is also used only in a **while** loop block. This statement skips the current iteration of the loop and passes the control to the next iteration after checking the loop condition. The syntax is as follows

continue ;

7.6 Return statement

Return statement in a function passes the control from the callee to the caller function and returns a value to the caller function. All functions including the **main()** must have exactly one **return** statement and it should be the last statement in the funtion body. The return type of the function should match the type of the expression. The return type of main is integer. The syntax is as follows

return *expression;*

7.7 Read/Print statements

The standard input and output statements in APL are **read** and **print** respectively. The read statement reads an integer value from the standard input device into an integer variable or an indexed array variable or a string value into a string variable. The print statement outputs a string literal or the value of string variable or an arithmetic expression into the standard output.

read *variable_name;*
print *expression / string;*

7.8 Breakpoint statement

A **Breakpoint** statement is used to debug the program. The program when run in debug mode pauses the execution at this instruction.

breakpoint;

8 System Calls

System Calls allow the programs written in APL to interact with the operating system running on the XSM architecture. 10 System Calls are supported by APL.

8.1 Create

Create s a file with the specified filename in the filesystem. The return value of this system call is 0 in case of success and the appropriate error code in case of failure.

integer ***Create****(string filename);*

NOTE: filename must not exceed 10 characters

8.2 Open

Returns a file descriptor of the file in the filesystem with the specified filename. The file descriptor is an integer value. If the **Open** fails, an appropriate error code is returned .

integer ***Open****(string fileName);*

8.3 Read

Reads one word from a file which has the specified file descriptor, into a string/integer variable. The return value of this system call is 0, if it is a success. If the **Read** fails, an appropriate error code is returned.

integer ***Read****(integer fileDescriptor, string/integer variable);*

8.4 Write

Writes one word from a string/integer variable, to a file in the filesystem with the specified file descriptor. The return value of this system call is 0, if it is a success. If the **Write** fails, an appropriate error code is returned.

integer ***Write****(integer fileDescriptor, string/integer variable);*

8.5 Seek

Seek is used to change the read/write head position in a file. It moves the head to the specified number of words from the beginning of the file. The return value of this system call is 0 in case of success and the appropriate error code in case of failure.

integer ***Seek****(integer fileDescriptor, integer numWords);*

8.6 Close

This system call is used to close an open file. The return value of this system call is 0 in case of success and the appropriate error code in case of failure.

integer ***Close****(integer fileDescriptor);*

8.7 Delete

This system call is used to delete the file from the file system whose name is specified in the argument. The return value of this system call is 0 in case of success and the appropriate error code in case of failure.

integer ***Delete****(string fileName);*

8.8 Fork

This system call is used to create a copy of the current process in the system. The return value of this system call is the PID of the child process for the parent, and 0 for the child.

integer ***Fork****();*

8.9 Exec

This system call is used to load the program, whose name is specified in the argument, in the memory space of the current process (overwriting existing contents) and start its execution. The return value of this system call is 1 in case of failure.

integer ***Exec****(string fileName);*

8.10 Exit

This system call is used to terminate the execution of the process which invoked it and remove it from the memory.

void ***Exit****();*