Reference Manual – AB Programming Language

Ankit Pant – 2018201035
Bhavi Dhingra – 2018201058

Abstract

AB is a compiled programming language. It is a very simple programming language and supports only the basic data-types, conditionals, etc. This reference manual explains the syntax of the AB programming language.

Contents

1 Introduction

AB is a simple programming language that supports very basic constructs. It is a compiled programming language. The following sections provide a brief description of the language.

1.1 Notation

The various formal definitions of the programming language are defined as:

$$Nonterminal \rightarrow terminal \,|\, Nonterminal \,|\, \epsilon$$

Where:

- **Nonterminal** is the the part of the rule from which terminals or other rules can be derived.
- **terminal** is the part of the rule from which no more rules can be derived and is the fundamental unit of the programming language.

Some important points:

- Within the formal definition if something needs to be present literally, it is enclosed in quotation marks (i.e. “ ”).
- When more than one rule can be derived from a Nonterminal, each rule is separated by a pipe symbol '|'.
- The epsilon symbol (ϵ) in a rule represents a null-string.

2 Micro Syntax (Lexical Analysis)

This section describes the tokens which are the part of the input program. They can be further sub-divided as:

2.1 Identifiers

Identifiers are tokens that are used to name and hence identify the variables, functions, etc. in the program. They are described by the following lexical definitions:

$Letter \rightarrow [\_]^*[a-zA-Z][\_a-zA-Z0-9]^*$
$Digit \rightarrow [0-9]$

2.2 Keywords

Keywords are reserved words that cannot be used as identifiers. They hold a special meaning to the programming language. The following keywords have been defined in the programming language:

include, if, else, for, while, int, char, signed, unsigned, bool, return, void, string, print, println, input, true, false, main, and , or, not, define, break, continue

2.3 Literals

Literals are notations for constant values of some built-in types. They are described by the following lexical definitions:

$CharLiteral \rightarrow\ ' < any\,ASCII\,character\,except\,newline\,or\,the\,quote >'$
$StringLiteral \rightarrow\ `` < anynumberof\ ASCII\,characters\,except\,newline\,or\,the\,quote > "$
$IntegerLiteral \rightarrow DecimalInteger$
$DecimalInteger \rightarrow Digit^{+}$
$Fraction \rightarrow\ ``."\,Digit^{+}$
$FloatLiteral \rightarrow IntegerLiteral\,Fraction$

2.4 Operators

Operators are used to performed various mathematical, relational, logical, etc. operations. They are described by the following lexical definitions:

$LogicalOptr \rightarrow and \mid or \mid not$
$RelationalOptr \rightarrow < \mid > \mid == \mid <= \mid >= \mid ! =$
$ArithmeticOptr \rightarrow + \mid - \mid * \mid / \mid \%$
$BitwiseOptr \rightarrow << \mid >> \mid \& \mid [\mid] \mid$
$AssignmentOptr \rightarrow = \mid + = \mid * = \mid / = \mid \% = \mid \& = \mid \mid = \mid <<= \mid >>=$

2.5 Data Types

These are the data types built into the programming language. They are described by the following lexical definitions:

$PrimitiveType \rightarrow void \mid bool \mid char \mid int \mid unsigned \mid float \mid long \mid long\,long \mid unsigned\,long\,long$

2.6 Comments

Comments are statements in the input program that are ignored by the compiler. This programming language supports only single line comments as of now. They are described as:

$Comment\ \rightarrow\ // < any\,character\,except\,newline >$

3 Macro Syntax (Grammatical analysis)

This section describes the various Grammatical rules using which the programming language has been defined. The following subsections give a brief description of the various grammatical rules used in the language and describes the syntax of the language.

3.1 Variables

Variables are memory locations. They are described as:

$VarDec \rightarrow DataType\, IdentifierList$

Example:

```
char x;
int[10] arr;
bool[3][3] mat;
```

3.2 Expressions

Expressions form a part of the statements in in this programming language. They are described as:

$Atom \rightarrow True \,|\, False \,|\, Identifier\ MultiDim \,|\, Literal \,|\, FuncCall$
$ExprList \rightarrow \epsilon \,|\, \text{‘,’}\, ExprList \,|\, Expr\ ExprList$
$Expr \rightarrow Atom \,|\, UnaryExpr \,|\, BinaryExpr \,|\, TernaryExpr \,|\, \text{“(”}\, Expr\, \text{“)”}$
$UnaryExpr \rightarrow Atom \,|\, \text{“} - \text{”}\, Atom \,|\, \text{“} + \text{”}\, Atom \,|\, \text{“not”}\, Atom$
$BinaryExpr \rightarrow Expr\ Optr\ Expr$
$TernaryExpr \rightarrow \text{“(”}\, Expr\, \text{“?”}\, Expr\, \text{“ : ”}\, Expr\, \text{“)”}$

Example:

```
x = 123;
y = x+100;
z = (x+y)*100;
x < (y+z)
```

3.3 Block

A block encloses a list of statements. It starts with a left curly bracket and ends with a right curly bracket ({}) Any variable declared in a block has a scope within the block only. It is described as:

$Block \rightarrow \{StmtList\}$

Example:

```
{
        x = 123;
        y = x+100;
        z = (x+y)*100;
        print("Hello World!");
}
```

3.4 Functions

Functions are subroutines that are defined in the program and can be used to define operations that need to be performed repeatedly. The last statement inside a function should be a return statement. They can be defined as:

$FuncDef \rightarrow DataType\, FuncName\text{“(”}\, ParameterList\text{“)”}\ Block$

$FuncName \rightarrow Indentifier \,|\, \text{“}main\text{”}$

$ParameterList \rightarrow \epsilon \,|\, \text{‘,’}\ ParameterList \,|\, DataType\,\text{‘\&’}Identifier\ ParameterList$

$\qquad\qquad\qquad\quad |\, DataType\, Identifier\ ParameterList$

Example:

```
int Square(int x)
{
        return (x*x);
}
```

3.5 Function Calls

Function calls can be used to transfers the execution flow to a function which has been defined previously. They can be defined as:

$FuncCall \rightarrow Identifier\ \text{“(”}\, ExprList\text{“)”}$

Example:

```
int x = 2;
int y = Square(x);
```

3.6 Statements

Statements form the majority of the programs. They can be subdivided into simple statements and compound statements. Simple statements are comprised within a single logical line whereas compound statements contain a group of statements. They can be defined as:

$Delim \rightarrow ;$

$StmtList \rightarrow \epsilon \,|\, CompoundStmt\ StmtList \,|\, SimpleStmt\ StmtList$

$SimpleStmt \rightarrow Expr\ Delim \,|\, PrintStmt\ Delim \,|\, PrintlnStmt\ Delim$

$\qquad\qquad |\, ContinueStmt\ Delim \,|\, BreakStmt\ Delim \,|\, IncludeStmt \,|\, ReturnStmt\ Delim$

$\qquad\qquad |\, VarDec\ Delim \,|\, FuncCall\ Delim \,|\, DefineStmt \,|\, InputStmtDelim$

$PrintStmt \rightarrow \text{“}print(\text{”}\, ExprList\text{”)”}$

$PrintlnStmt \rightarrow \text{“}println(\text{”}\, ExprList\text{”)”}$

$InputStmt \rightarrow \text{“}input(\text{”}\, Identifier\ MultiDim\text{“)”}$

$ReturnStmt \rightarrow \text{“}return\text{”}\ Expr$

$BreakStmt \rightarrow \text{“}break\text{”}$

$ContinueStmt \rightarrow \text{“}continue\text{”}$

$IncludeStmt \rightarrow \text{“\#}include\text{”}\, Identifier$

$DefineStmt \rightarrow \text{“\#}define\text{”}\, Identifier\ Literal$

$CompoundStmt \rightarrow IfStmt \,|\, ForStmt \,|\, WhileStmt \,|\, FuncDef$

$IfStmt \rightarrow \text{“}if(\text{”}\, Expr\text{”)”}\ Block\ ElseStmt$

$ElseStmt \rightarrow \epsilon \,|\, \text{“}else\text{”}\ Block \,|\, \text{“}else\text{”}\ IfStmt$

$ForStmt \rightarrow \text{“}for(\text{”}\, Expr\ Delim\ Expr\ Delim\ IncrementStmt\text{“)”}\ Block$

$IncrementStmt \rightarrow \epsilon \,|\, Expr$

$WhileStmt \rightarrow \text{“}while(\text{”}\, Expr\text{“)”}\ Block$

3.7 Identifiers

Identifiers are tokens that are used to name and hence identify the variables, functions, etc. in the program. They can be defined on a macro level as:

$Identifier List \rightarrow Identifier,\, Identifier List \,|\, Identifier$

3.8 Literals

Literals are notations for constant values of some built-in types. They can be defined on a macro level as:

$\mathit{Literal} \rightarrow \mathit{CharLiteral} \,|\, \mathit{StringLiteral} \,|\, \mathit{IntegerLiteral} \,|\, \mathit{FloatLiteral}$

3.9 Operators

Operators are used to performed various mathematical, relational, logical, etc. operations. They can be defined on a macro level as:

$\mathit{Optr} \rightarrow \mathit{LogicalOptr} \,|\, \mathit{RelationalOptr} \,|\, \mathit{ArithmeticOptr} \,|\, \mathit{BitwiseOptr} \,|\, \mathit{AssignmentOptr}$

3.10 Data Types

These are the data types built into the programming language. They are sub-divided into primitive and compound data types. They can be defined on a macro level as:

$\mathit{DataType} \rightarrow \mathit{PrimitiveType}\; MultiDim$
$MultiDim \rightarrow \epsilon \,|\, \text{“[”}\, \mathit{Expr}\, \text{“]”}\; MultiDim$

Examples:

```
=> // Simple Statements (=> indicates different statements)
=> print("Hello World!");
=> println("This will add a newline character at the end");
=> int x;
=> input(x);
=> // Compound Statements (=> indicates different statements)
=>if(x<100)
{
       println("Less than Hundred!");
}
=>else
{
       println("More than hundred");
}
```

4 Few Semantic Checks

The five semantic checks that may be done on an input program are:

1. **Type Checking:** It can be done to ensure that all operands in any expression are of appropriate types.

2. **Scope Checking:** It can be done to constraint the visibility of an identifier to some subsection of the program.

3. **Undeclared Variable Check:** It can be done to ensure that the variable has been declared before use.

4. **Multiple Variable Declaration Check:** It can be done to ensure that the same variable has not been declared multiple times within a scope.

5. **Function Call Arguments Check**: It can be done to ensure that the number of arguments as well as the type of arguments in the function calls match the function signatures.

5 Sample Programs

5.1 A program which computes $g(N,k) = \Sigma_{i=1}^{N} i^k$ where N and k are given as inputs.

```
int main()
{
        int N;
        int k;
        int sum;
        sum = 0;
        print("Enter the value of N: ");
        input(N);
        print("Enter the value of k: ");
        input(k);
        int i;
        for(i=1;i<=N;i+=1)
        {
                int temp = i;
                int j;
                for(j=2;j<=k;j+=1)
                {
                        temp = temp * i;
                }
                sum = sum + temp;
        }
        print("The value of the expression = ");
        println(sum)
        return 0;
}
```

5.2 A program to check if a given input number N is prime.

```
bool is_prime(int n)
{
        int i;
        for (i = 2; i*i <= n; i += 1)
        {
                if (n % i == 0)
                {
                        return false;
                }
        }
        return true;
}

int main()
{
        int n;
        input(n);

        print (n);
        if (is_prime(n))
        {
                println (" is a prime number");
        }
        else
        {
                println (" is not a prime number");
        }
        return 0;
}
```

5.3 A program Find the sum of all prime numbers strictly less than N where N is provided as an input.

```
int main()
{
        int N;
        print("Enter the value of N: ");
        input(N);
        bool[N] parr;
        parr[0] = false;
        parr[0] = false;
        int i;
        for(i=2;i<N;i+=1)
        {
                parr[i] = true;
        }
        for(i=2;i<N;i+=1)
        {
                if(parr[i]==true){
                        int j;
                        for(j=2*i;j<N;j+=i)
                        {
                                parr[j] = false;
                        }
                }
        }
        int sum;
        sum = 0;
```

```
	for(i=2;i<N;i+=1)
	{
		if(parr[i]==true){
			sum = sum + i;
		}
	}
	print("The sum of all primes strictly less than N = ");
	println(sum);
	return 0;
}
```

5.4 A program Enumerate all the Pythagorean triplets (x, y, z) where x, y, z are integers and $x^2 + y^2 = z^2$ and $z < 10000000$. Output the count at the end.

```
int main()
{
	unsigned long long a, b, c, m, n;

	int d;
	for (d = 1; d < 100000000/2; d += 1)
	{
		for (n = 1; n < 100000000/2; n += 1)
		{
			m = n+d;
			c = m*m + n*n;
			if (c > 100000000)
			{
				break;
			}
			a = m*m - n*n;
			b = 2*m*n;
			println ("(", a, ",", b, ",", c, ")");
		}
	}
	return 0;
}
```

5.5 A program to print all combinations of $\{1, \ldots, n\}$ where n is given as an input.

```
void Combinations(int arr, int n, int l, int r)
{
	if(l==r){
		int i;
		for(i=0;i<n;i+=1)
		{
			print(arr[i]);
			print(",")
		}
		print(" ");
	}
	else{
		int i;
		for(i=l;i<=r;i+=1)
```

```
                {
                        int temp;
                        temp = arr[l];
                        arr[l] = arr[i];
                        arr[i] = temp;
                        Combinations(arr,n,l+1,r);
                        temp = arr[l];
                        arr[l] = arr[i];
                        arr[i] = temp;
                }
        }
}

int main()
{
        int N;
        print("Enter the value of N: ");
        input(N);
        int[N] arr;
        int i;
        for(i=0;i<N;i+=1)
        {
                arr[i] = (i+1);
        }
        println("The combinations are: ");
        Combinations(arr,N,0,N-1);
        println();
        return 0;
}
```

5.6 A program for insertion sort.

```
void swap(int& a, int& b)
{
        int temp = a;
        a = b;
        b = temp;
}

int main()
{
        int n;
        print ("Enter the size of array: ");
        input (n);

        int[n] arr;
        println ("Enter the values:");
        int i;
        for (i = 0; i < n; i += 1)
        {
                input (arr[i]);
        }

        int i, j;
        for (j = 1; j < n; j += 1)
        {
                for (i = j-1; ()i >= 0) && ()arr[i] > arr[i+1]); i -= 1)
                {
                        swap(arr[i], arr[i+1]);
                }
        }
        println ("After Insertion Sort:");
        for (i = 0; i < n; i += 1)
        {
                print (arr[i],);
        }
        println ();
        return 0;
}
```

5.7 A program for Radix sort.

```
int n;

void Count_Sort(int[] arr, int index)
{
        int[n] aux_arr;
        int[10] count;
        int i;
        for(i=0;i<10;i+=1)
        {
                count[i] = 0;
        }
        for(i=0;i<n;i+=1)
        {
                int temp = (arr[i]/index)%10;
                count[temp] = count[temp]+1;
        }
        for(int i=1;i<10;i+=1)
        {
                count[i] = count[i] + count[i-1];
        }
        for(i=n-1;i>=0;i-=1)
        {
                int temp = (arr[i]/index)%10;
                temp = temp - 1;
                int temp2 = count[temp];
                aux_arr[temp2] = arr[i];
                count[temp] = count[temp] - 1;
        }
        for(i=0;i<n;i+=1)
        {
                arr[i] = aux_arr[i];
        }
}

int main()
{
        print("Enter the number of elements in the array");
        input(n);
        int[n] arr;
        print("Enter the elements of the array")
        int i;
        for(i=0;i<n;i+=1)
        {
                input(arr[i]);
        }
        int maxe = arr[0];
        for(i=1;i<n;i+=1)
        {
                if(maxe>arr[i]){
                        maxe = arr[i];
                }
        }
        int digits = 0;
        while(maxe>0)
        {
                digits = digits + 1;
                maxe =maxe/10;
        }
        int exp = 1;
        for(i=1;i<=digits;i+=1)
        {
                Count_Sort(arr,exp);
                exp = exp * 10;
        }
        print("The sorted array after applying Radix Sort is");
        for(i=0;i<n;i+=1)
        {
                print(arr[i]);
                print(" ");
        }
        println();
        return 0;
}
```

5.8 A program for Merge sort.

```
#define INT_MAX 1000000007

void merge(int[]& arr, int i, int m, int j)
{
        int arr1[m-i+1 + 1], arr2[j-m + 1];
        arr1[m-i+1] = arr2[j-m] = INT_MAX;

        int k;
        for (k = 0; k < m-i+1; k += 1)
        {
                arr1[k] = arr[i+k];
        }
        for (k = 0; k < j-m; k += 1)
        {
                arr2[k] = arr[m+k+1];
        }
        int p = 0, q = 0;
        for (k = i; k <= j; k += 1)
        {
                if (arr1[p] < arr2[q])
                {
                        arr[k] = arr1[p];
                        p += 1;
                }
                else
                {
                        arr[k] = arr2[q];
                        q += 1;
                }
        }
}

void merge_sort(int[]& arr, int n, int i, int j)
{
        if (i < j)
        {
                int m = i + (j-i)/2;
                merge_sort(arr, n, i, m);
                merge_sort(arr, n, m+1, j);
                merge(arr, i, m, j);
        }
}

int main()
{
        int n;
        print ("Enter the size of array: ");
        input (n);

        int arr[n];
        println ("Enter the values:");
        int i;
        for (i = 0; i < n; i += 1)
        {
                input (arr[i]);
        }

        merge_sort(arr, n, 0, n-1);
        println ("After Merge Sort: ");
        for (i = 0; i < n; i += 1)
        {
                print (arr[i],);
        }
        println ();
        return 0;
}
```

5.9 A program to compute the sum of two input matrices.

```
int main()
{
        int n;
        int m;
        print("Enter the number of rows of the matrices: ");
        input(m)
        print("Enter the number of columns of the matrices: ");
        input(n)
        int[m][n] mat1;
        int[m][n] mat2;
        print("Enter the elements of first matrix");
        int i;
        int j;
        for(i=0;i<m;i+=1)
                {
```

```
                for(j=0;j<n;j+=1)
                {
                        input(mat1[i][j]);
                }
        }
        print("Enter the elements of second matrix");
        for(i=0;i<m;i+=1)
        {
                for(j=0;j<n;j+=1)
                {
                        input(mat2[i][j]);
                }
        }
        int[m][n] sum_mat;
        for(i=0;i<m;i+=1)
        {
                for(j=0;j<n;j+=1)
                {
                        sum_mat[i][j] = mat1[i][j] + mat2[i][j];
                }
        }
        print("The sum of the matrices is");
        for(i=0;i<m;i+=1)
        {
                for(j=0;j<n;j+=1)
                {
                        print(sum_mat[i][j])
                        print(" ");
        }
                println();
        }
        return 0;
}
```

5.10 A program compute the product of two input matrices.

```
int main ()
{
        int A_m, A_n;
        print ("Enter dimensions of matrix A: ");
        input (A_m)
        input (A_n);
        int[A_m][A_n] A;
        println ("Enter values of matrix A:");
        int i, j;
        for (i = 0; i < A_m; i += 1)
        {
                for (j = 0; j < A_n; j += 1)
                {
                        input (A[i][j]);
                }
        }

        int B_m, B_n;
        println ();
        print ("Enter dimensions of matrix B: ");
        input (B_m);
        input (B_n);
        int[B_m][B_n] B;
        println ("Enter values of matrix B:");
        for (i = 0; i < B_m; i += 1)
        {
                for (j = 0; j < B_n; j += 1)
                {
                        input (B[i][j]);
                }
        }

        if (A_n != B_m)
        {
                println ("The matrices are not compatible for multiplication!!");
                return 1;
        }

        int[A_m][B_n] C;
        for (i = 0; i < A_m; i += 1)
        {
                for (j = 0; j < B_n; j += 1)
                {
                        int k;
                        C[i][j] = 0;
                        for (int k = 0; k < A_n; k += 1)
                        {
                                C[i][j] += A[i][k] * B[k][j];
                        }
                }
        }

        println();
        println ("A x B: ");
        for (int i = 0; i < A_m; ++i)
        {
                for (int j = 0; j < B_n; ++j)
                {
                        print (C[i][j],);
                }
                println ();
        }
        return 0;
}
```