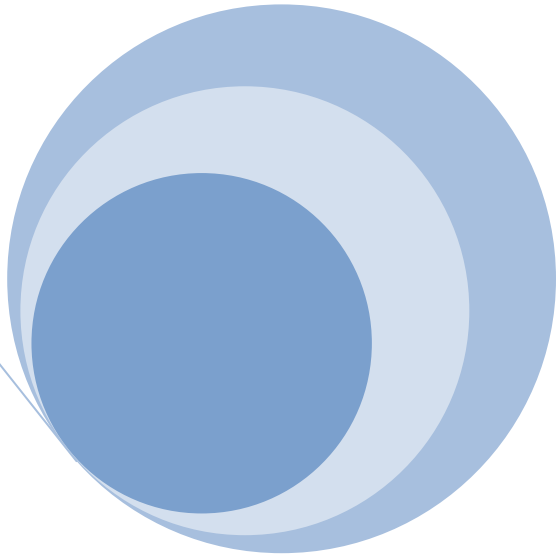
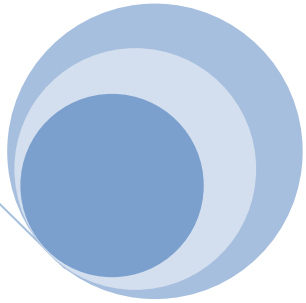


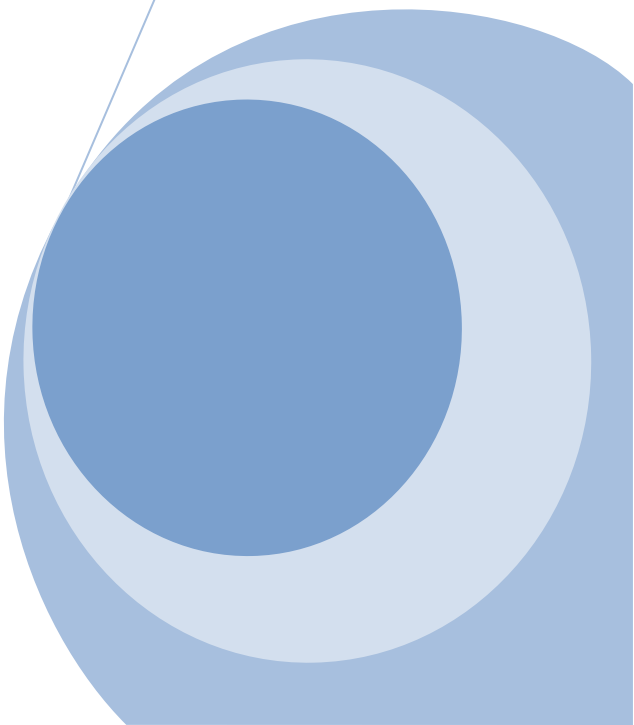


“AIC” All In the Cards Language Reference Manual

COMS 4115 Programming Languages and Translators

AIC is a language that allows the easy creation of card games ranging from the well known to brand new user created.

Andrew Willouer asw2126
6/22/2009



Contents

1 Introduction

There are thousands of card games that use a standard 52 card deck. AIC allows the quick creation of card games by only specifying certain parameters. AIC language describes the rules, players, dealer, number of decks, and card values and the AIC compiler takes this input and creates a console based card game. This allows the user to concentrate on the rules of the game and not worrying about the coding details.

2 Lexical Conventions

There are six kinds of tokens in the AIC language: identifiers, keywords, constants, strings, expression operators, and other separators. Blanks, tabs, newlines, spaces and comments (whitespace) are ignored except to separate tokens. Some whitespace is required to separate adjacent tokens.

2.1 Comments

Comments start with // and end with newline. Everything from // to the end of the line is considered part of the comment. Comments do not nest.

2.2 Identifiers (Names)

An identifier is a sequence of letters (a-z), digits (0-9), symbols and the first character must be a letter. Upper and lower case letters are distinct and there is no limit on the length of an identifier. The underscore character and the period character counts as letters also. Keywords are reserved and cannot be redefined.

2.3 Keywords

The following identifiers are reserved for keywords:

if	else	while	do	for	return
Ndecks	Players	Dealer	CName	Print	Shuffle
Deal	Input	Decks	Cards	Hands	setvalue
value	true	false	var	main	

NDecks (To specify the number of decks, 1-10)

Players (To specify the number of players, 1-10)

Dealer (To specify whether to have a dealer as a separate player or not, true or false)

CName (The name of the card game)

2.4 Integer Constants

An integer constant is a sequence of digits, 0-9 and always considered to be decimal.

2.5 Strings

A string is a sequence of one or more characters which may include letters, digits, symbols and whitespace surrounded by double quotes (eq. "This is a String!").

2.5 Operators

An operator performs an operation on tokens. The operators are listed below in order of precedence highest to lowest:

&&	Logical and	Left to Right Precedence
||	Logical or	Left to Right Precedence
=	Assignment	Right to Left Precedence
==	Equality test	Left to Right Precedence
!=	Inequality test	Left to Right Precedence
<	Less than	Left to Right Precedence
<=	Less than or equal	Left to Right Precedence
>	Greater than	Left to Right Precedence
>=	Greater than or equal	Left to Right Precedence
+	Add	Left to Right Precedence
-	Subtract	Left to Right Precedence
/	Division	Left to Right Precedence
*	Multiplication	Left to Right Precedence

2.6 Conditional Statements

A conditional statement consists of either an if(expression) { true block } else {false block} or just if(expression) { true block} without an else block. Conditional statements can be nested.

2.7 Iteration Statements

An iteration statement repeats executing statements as long as the condition is true for a while loop and a certain number of times for a for loop.

The while iteration statement consists of while(expression) {statements} The expression of a while loop must resolve to either true or false and the statements are run as long as the expression is met.

The for iteration statement consists of for(start expression; test expression; count expression) {statements}

3 Data Types

There are five distinct data types:

Integers: A list of digits (0-9)

Strings: A list of characters surrounded by double quotes.

Cards: A list that contains the card symbol and an integer value.

Hands: A list that contains all the cards that belong to someone.

Decks: A list that contains all the cards not in play or someone's hand. Cards can be removed from the deck or put back into the deck depending on the rules of the game.

3.1 Arrays

Arrays are declared with square brackets [ ]. The language supports arrays of integers and strings.

4 Scope

There are two kinds of scope: global scope and local scope. Identifiers declared in a local function can only be accessed in that function. Global identifiers declared outside functions may be used throughout the entire program.

5 Declarations

Identifiers must be identified as variables or functions. Declarations have the following form:

Variables: var identifier

Functions: identifier (parameters list) {block of function}

5.1 Variables

Variables are uninitialized when created. They must be assigned a value to be used. Variables values can change after being initialized.

5.2 Functions

A function must return a valid type. A function does not need a parameter list. The parameter list can contain multiple identifiers separated by a comma. A function can call another function within its function block, but a function cannot declare a new function within its function block. When calling a function the parameter list must be the same size as the parameter list declared when the function was declared.

5.2.1 Predefined Functions

There are several built in functions included in the language. They include:

Print ( ) Prints everything contained in between the parenthesis. Text must be surrounded by double quotes. Variables do not need to be in double quotes and their values are printed.

Shuffle(Decks) Takes the Decks type and randomly distributes the cards throughout the deck

Deal (Player, number) Takes a certain number of cards from the deck, specified by number and gives them to the player specified by Player.

Input (string) Takes one string variable and waits for text from the console and places the text into the string variable

main() Has no parameters but is the specialized function where code begins to be executed from.

6 References

Ritchie, Dennis M, "C Reference Manual" http://cm.bell-labs.com/cm/cs/who/dmr/cman.pdf

7 Example Code

Below is example code for Blackjack played with 2 decks and a dealer.

```
Decks = 2

Players = 2

Dealer = true

CName=Blackjack

Print(CName)  //Print the name of the game to the console window

Shuffle //This will shuffle 2 decks of cards and randomly distribute the cards in the deck.

//Now need to set the value of the face cards.

Cards.J*.setvalue = 10 //Using matches all 4 suites.

Cards.Q*.setvalue = 10

Cards.K*.setvaule =10

Cards.A*.setvalue =11 //The value of 1 will be programmed into the rules

//Card game initialized at this point.

Deal(Dealer, 2)  This will take the top 2 cards from the deck and add them to the dealer's hand.

Print(Hands.Dealer.Cards[1]) //Only prints the dealers second card and the first is hidden in blackjack

Print(Hands.Dealer.Cards.value[1]) //Only prints the dealers second card's value

//Now need to deal to all the players

For(J = 1 to Players.value)

{

Deal(J, 2)

Print(Hands.J.Cards) //Prints the cards the player has  (For example 2H   9S)
```

```
Print(Hands.J.Cards.value) //Prints the player hand's value which would be 11 for the previous cards
}
For(J = 1 to Players.value)
{
Print("Player" J "Do you want another card? Yes or No?"
Input  //wait for text from console
        If(Input = "Yes")
        {
        Deal(J,1)  //removes top card from deck and adds to players hand
        Print(Hands.J.Cards) //Prints the cards the player has  (For example 2H   9S)
        Print(Hands.J.Cards.value) //Prints the player hand's value which would be 11 for the previous cards
        If(Hands.J.Cards.value > 21)
        {
        Print("Player" J "You went over 21, you lose")
        }
}
//Player's turns over, time for the dealer to go
Print(Hands.Dealer.Cards) //Print both of the dealers cards
Print(Hands.Dealer.Cards.value]) //And their values
For(J = 1 to Players.value)
{
        //Deal the dealer cards until he goes bust or ties or beats the players hand and make sure the player hasn't already gone bust
        While(Hands.Dealer.Cards.value <= 21) && (Hands.Dealer.Cards.value <= Hands.J.Cards.value)&&(Hands.J.Cards.value<=21)
        {
```

```
            Deal(Dealer, 1)

            Print(Hands.Dealer.Cards) //Prints the cards the dealer has

            Print(Hands.J.Dealer.value)

        }

}
```

//At this point the dealer has compared his cards to all the players and either gone bust or has a higher score or equal score then all of them. Dealer wins if the scores are equal in this version.

```
        If(Hands.Dealer.Cards.value > 21)

        {

        Print("Dealer went bust everyone wins");

        }

        Else

        {

        Print("Dealer wins")

        }
```