Crayon (.cry)

Language Reference Manual

Naman Agrawal (na2603)
Vaidehi Dalmia (vd2302)
Ganesh Ravichandran (gr2483)
David Smart (ds3361)

1 Lexical Elements

1.1 Identifiers

Identifiers are strings used to name variables and functions. The first character of an identifier must be a letter of the alphabet. They are case sensitive.

1.2 Keywords

The following keywords are reserved for use in the language and cannot be used as identifiers. These keywords are case sensitive.

Array
Canvas
Pixel
for
if
else
else if
void
return
function
import
neg
print
true
false

1.3 Comments

The characters **:'(** introduce a comment, which terminates with the character **)**

1.4 Literals

Literals are constants, either string, numeric, or boolean values. Type casting is not supported, so changing the type of a variable from its initial type will throw an error.

1.4.1 String Literals
String literals are sequences of zero or more non-double-quote characters and/or escaped characters. They are enclosed in double quotes. Escaped characters are characters that come after a backslash "\", which signals to the compiler that the escaped character should be treated one of the following ways:

\”	Insert a double quote at this point.
\\	Insert a backslash at this point.
\n	Insert a newline at this point.
\t	Insert a tab at this point.
\r	Insert a carriage return at this point.
\b	Insert a backspace at this point.

(Note: credit for table above goes to Julie Chien’s T.B.A.G Language Manual, found at the following link: http://www.cs.columbia.edu/~sedwards/classes/2015/4115-fall/reports/tbag.pdf)

1.4.2 Integer Literals
Integer literals are whole numbers in base-10 represented by a sequence of one or more digits Integers preceded by the “neg” keyword are negative numbers, separate from the subtraction operator mentioned below.

Examples: 42; neg 6

1.4.3 Boolean Literals
A boolean literal represents a truth value and can have the values true or false.

Examples: true; false

1.5 Operators
Operators are characters that perform actions on different elements, such as addition, subtraction, and other mathematical processes. Boolean operators such as “and” and “or” can also be used on boolean literals.

Examples: +,*, -, ||, &&

1.6 Delimiters
Delimiters are characters that separate other elements and tell the compiler how to interpret associated delimiters.

1.6.1 Parentheses and Braces

Parentheses force evaluation of parts of an algorithm in a specific order and enclose arguments of a function.

1.6.2 Commas
Commas separate arguments of a function.

1.6.3 Brackets
Brackets are used for array initialization, assignment, and access.

1.6.4 Semicolon
A semicolon indicates the end of a sequence of code.

1.6.5 Curly Braces
Curly braces enclose function definitions and blocks of code. Blocks enclosed within curly braces do not need to be terminated by semicolons.

1.7 Whitespace
Whitespace separates different elements and can be used within string literals.

List of whitespace characters: spaces, tabs, newlines, and vertical tabs.

2 Data Types
Crayon(.cry) is statically typed. The types of all variables are known at compile time and cannot be changed.

2.1 Primitive Data Types

2.1.1 int
These are 32-bit signed integers that can range from −2,147,483,648 to 2,147,483,647.

2.1.2 string
All text values will be of this type, including hex codes for colors.

2.1.3 boolean
A truth value that can be either true or false.

2.1.4 void
Used only as a return type in a function definition; specifying void as the return type of a function means that the function does not return anything.

2.2 Non-Primitive Data Types

2.2.1 Arrays
Arrays are ordered, fixed-size lists that can be used to hold both primitive and non-primitive

data-types. All elements of an array must be of the same type. An array must be initialized with its size.

2.2.1.1 Declaring Arrays

You can declare an array by indicating the type of the elements that the array will contain, followed by brackets enclosing the number of elements an array will hold, followed by an identifier for the array. For example:

int[5] myArray; declares an array named myArray that can hold 5 integers.

2.2.1.2 Accessing and setting array elements

Array elements can be accessed by providing the desired index of the element in the array you wish to access enclosed within brackets next to the identifier of the array. For example:

myArray[1]; returns the element in myArray at index 1. Array elements can be set by accessing the element via the desired index in which to place the item and then assigning the desired value to the entry. For example:

myArray[1]=4; sets the element in myArray at index 1 to 4.

2.2.2 Canvas

Canvases are finite-sized two-dimensional Arrays that holds Pixel values at each coordinate. Pixel values are modified by the user to create the image. A canvas can be rendered by the corresponding Crayon graphics library to produce the raster image.

2.2.2.1 Declaring Canvas

You can declare a canvas by indicating the name and x and y dimension sizes in brackets. For example:

Canvas banana[20,20];

2.2.3 Pixel

The Canvas is made up of many Pixels. Each Pixel has a name, x and y coordinates, and color, and can be referenced after creation to change its color and/or coordinates.

Pixel(banana,x,y,hex)

3 Expressions and Operators

3.1 Expressions

Expressions are made up of one or more operands and zero or more operators. Innermost expressions are evaluated first, as determined by grouping into parentheses, and operator precedence helps determine order of evaluation. Expressions are otherwise evaluated left to right.

3.2 Operators

The table below presents the language operators (including assignment operators, mathematical operators, logical operators, and comparison operators), descriptions, and associativity rules.

Operator	**Meaning**
+	addition
-	subtraction
*	multiplication
/	division
=	assignment
%	modulus
++	increment
--	decrement
==	equality
!=	not equal
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
&&	and
||	or
!	not

3.3 The if Statement

The if statement is used to execute a statement if a specified condition is met. If the specified condition is not met, the statement is skipped over. The general form of an if statement is as follows:

```
if( condition ) {
        action1;
}
else {
	Default action;
}
```

“if” must be followed with “else,” although the else may have no statement associated with it:

```
if( condition ) {
	action1;
	} else{ }
```

3.4 Functions

3.4.1 Function Definitions

Function definitions consist of an initial keyword “function,” a return type, a function identifier, a set of parameters and their types, and then a block of code to execute when that function is called with the specified parameters. An example of an addition function definition is as follows:

```
func int sum(int a,int b) {
	return a + b;
}
```

3.4.2 Calling Functions

A function can be called its identifier followed by its params in parentheses.

for example: sum(1,2);

3.4.3 Built-in Functions

3.4.3.1 The print function

The print function can be used to print out strings, integers, and booleans to the command line. The general structure for calling the print function is as follows:

```
print("welcometothejungle");
print(666);
```

Anything within the parentheses will be printed; it must be of type string, integer, or boolean. Note that if a user wishes to print on a new line, a new line must be explicitly specified.

3.4.3.2 Canvas functions

3.4.3.2.1 Canvas.set

The Canvas.set function can be used to set the cursor at a particular (x,y) value on the canvas. A cursor is simply a reference to a specific Pixel, which will be returned by the function at that particular coordinate.The general structure for calling the Canvas.set function is as follows:

Canvas.set(mycanvas, 0,0); *:’(returns the Pixel for the corresponding Canvas mycanvas at 0,0)*

This sets the cursor at the top left of the canvas.

3.4.3.2.2 Canvas.fill

The Canvas.fill function can be used to fill colour in specific pixels. The general structure for calling the Canvas.fill function is as follows:

```
Canvas.fill(mycanvas, 700, 500, ”blue”);
```

This creates a box of horizontal length 700 pixels and vertical length of 500 pixels with the colour blue (which is set in the standard library to an appropriate hex code).

3.4.3.2.3 Canvas.move

The Canvas.move function can be used to move the cursor x pixels in the horizontal direction and y pixels in the vertical direction from its current location. The general structure for calling the Canvas.move function is as follows:

```
Canvas.move(mycanvas, 0,100);
```

This moves the cursor 0 pixels in the x direction and 100 pixels in the y direction from the current position. This will return the new Pixel type at the new location.

3.5 Context Free Grammar

Note: Taken from Simplified C version from lecture, but will be expanded upon more as the Crayon project is underway and we learn how this works!

program → ϵ | program vdecl | program fdecl

fdecl → id (formals) { vdecls stmts }

formals → id |formals , id

vdecls → vdecl | vdecls vdecl

vdecl → int id ;

stmts → ϵ |stmts stmt

stmt → expr ; | return expr ; | { stmts } | if (expr) stmt| if (expr) stmt else stmt| for (expr ; expr ; expr) stmt| while (expr) stmt

expr → lit | id | id (actuals) | (expr) | expr + expr| expr - expr| expr * expr| expr / expr| expr == expr| expr != expr| expr < expr| expr <= expr| expr > expr| expr >= expr| expr = expr

actuals → expr| actuals,expr