Analysis on the History of Programming Languages

David Muckle ‘18 – (Sponsors: Professor Ken Basye and Professor John Magee)

CLARK UNIVERSITY

Project Summary

Computers and the languages used to create their programs have been around for less than a century, but we have already seen a stunning amount of change and growth in how we use computers, especially languages. Languages have evolved with not just computers, but how we use them. In order to better understand how languages have evolved over time, I have picked five languages to study, and demonstrated five programs in each language, to exemplify how the languages differ in construction. BASIC, for example, was created to be easy to use both for beginners and for businesses, and thus makes user input easy to perform, while more complicated things like function definitions and complex data types fall by the wayside. Compared to Lisp, with its focus on lists and functions as first class citizens, BASIC is simpler because Lisp was made for mathematicians, not businesses. Included here is a table of language features of the five languages, a family tree of languages, and example programs in each language.

Language Features

	BASIC	Lisp	C	Java	Go
Type	Interpreted	Interpreted	Compiled	Compiled	Compiled
Imperative	Yes	Yes	Yes	Yes	Yes
Object Oriented	No	No	No	Yes	No
Functional	No	Yes	No	No	Yes
Supports Recursion	No	Yes	Yes	Yes	Yes

Language Family Tree

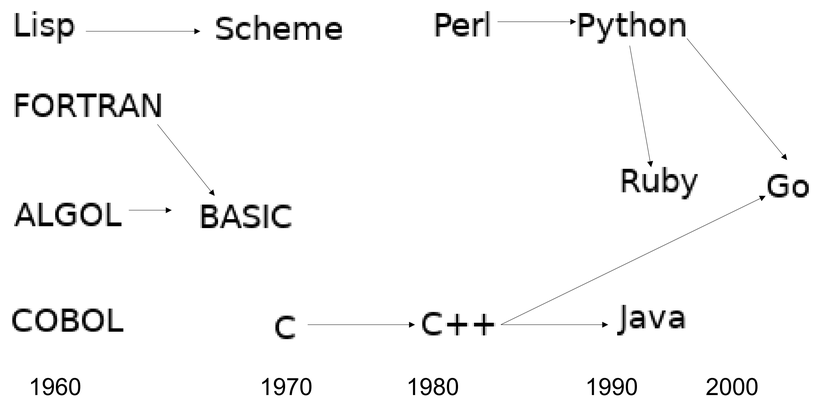

Fig. 1: The language family tree, with older languages on the left, newer on the right.

Example Programs

Echo User Input

BASIC (Built using FreeBASIC with -lang qb option.)

```
10 INPUT "", a$
20 PRINT "You said "; a$
30 END
```

Lisp (Scheme) (Built using Racket v6.11)

```
(define (echo)
  (define in (read-
line))
  (display "You said ")
  (displayln in))
(echo)
```

C (Built using gcc version 7.3.1)

```
#include <stdlib.h>
#include <stdio.h>
void main() {
    char in[256];
    fgets(in, 256, stdin);
    printf("You said %s\n", in);
}
```

Java (Built using Java 1.8.0.)

```
import java.util.Scanner;
public class echo {
    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        System.out.println("You said " + s.next());
    }
}
```

Go (Built using Go version 1.10.0/1.9.5)

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strings"
)

func main() {
    reader := bufio.NewReader(os.Stdin)
    text, _ := reader.ReadString('\n')
    text = strings.Replace(text, "\n", "", 1)
    fmt.Printf("You said %s\n", text)
}
```

Add Two Numbers

BASIC

```
10 INPUT "Enter a number ", x
20 INPUT "Enter another number ", y
30 LET sum = x + y
40 PRINT sum
50 END
```

Lisp (Scheme)

```
(define (addition x y)
  (define sum (+ x y))
  (displayln sum))
(define inX (read-line))
(define inY (read-line))
(define x (string-
>number (if (eof-object?
inX) (error "Requires
input") inX)))
(define y (string-
>number (if (eof-object?
inY) (error "Requires
input") inY)))
(if (and (number? x)
(number? y)) (addition x
y) (error "Input not a
number"))
```

C

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void main() {
    char inX[256];
    char inY[256];
    fgets(inX, 256, stdin);
    fgets(inY, 256, stdin);
    int x = atoi(inX);
    int y = atoi(inY);
    printf("%d\n", (x + y));
}
```

Java

```
import java.util.Scanner;
public class sum {
    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        int x = s.nextInt();
        int y = s.nextInt();
        int sum = x + y;
        System.out.println(sum);
    }
}
```

Go

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
    "strings"
)

func main() {
    reader := bufio.NewReader(os.Stdin)
    xIn, _ := reader.ReadString('\n')
    xIn = strings.Replace(xIn, "\n", "", 1)
    yIn, _ := reader.ReadString('\n')
    yIn = strings.Replace(yIn, "\n", "", 1)
    x, _ := strconv.Atoi(xIn)
    y, _ := strconv.Atoi(yIn)
    sum := x + y
    fmt.Println(sum)
}
```

Compare Two Inputs

BASIC

```
10 INPUT "Enter an input ", x$
20 INPUT "Enter another input ", y$
30 IF x$ = y$ THEN GOTO 80
50 ELSE GOTO 100
70 END IF
80 PRINT "Inputs are the same"
90 END
100 PRINT "Inputs are not the same"
110 END
```

Lisp (Scheme)

```
(define (compare str1
str2)
  (equal? str1 str2))
(define x (read-line))
(define y (read-line))
(if (compare x y)
(displayln "Inputs are
the same") (displayln
"Inputs are not the
same"))
```

C

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void main() {
    char x[256];
    char y[256];
    fgets(x, 256, stdin);
    fgets(y, 256, stdin);
    if(strcmp(x, y) == 0) {
        printf("Same input\n");
    } else {
        printf("Not the same input\n");
    }
}
```

Java

```
import java.util.Scanner;
public class compare {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        String x = s.next();
        String y = s.next();
        if(x == y){
            System.out.println("Inputs are the
same");
        } else {
            System.out.println("Inputs are not the
same");
        }
    }
}
```

Go

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strings"
)

func main() {
    reader := bufio.NewReader(os.Stdin)
    xIn, _ := reader.ReadString('\n')
    xIn = strings.Replace(xIn, "\n", "", 1)
    yIn, _ := reader.ReadString('\n')
    yIn = strings.Replace(yIn, "\n", "", 1)
    if xIn == yIn {
        fmt.Println("Inputs are the same")
    } else {
        fmt.Println("Inputs are not the same")
    }
}
```

Linked List

Note: Because the version of BASIC being used does not support recursive type definitions, the example here does not work. However, if recursive definitions were allowed, this is what it would look like.

Note that Lisp is built around the evaluation of lists, thus it by default uses linked lists. The example here uses the car and cdr functions, which return the first element and the rest of the list respectively. The example here would then return the second element of a list.

BASIC

```
10 TYPE list
 nextlist AS list
 value AS STRING
END TYPE
50 DIM firstList AS list
60 firstList.value = "List "
70 DIM secondList AS list
80 firstList.nextList = secondList
90 secondList.value = "test"
100 PRINT firstList.value
103 IF fistList.data == "" THEN
GOTO 110
105 firstList = firstList.nextList
110 END
```

Lisp (Scheme)

```
(define (list-example l)
  (car (cdr l)))
(define x "List ")
(define y "test\n")
(define z "not printed")
(define l (cons x (cons
y z)))
(display (car l))
(display (list-example
l))
```

C

```
#include <stdlib.h>
#include <stdio.h>
struct list {
    char* data;
    struct list* next;
};
typedef struct list* List;

void main() {
    List newlist = malloc(sizeof(struct
list));
    newlist->data = "List ";
    newlist->next = malloc(sizeof(struct
list));
    newlist->next->data = "test\n";
    while (newlist != NULL){
        printf(newlist->data);
        newlist=newlist->next;
    }
}
```

Java

```
public class List {
        private Object data;
        private List next;

        public List(Object data) {
                this.data = data;
        }
        public void append(Object data) {
                List newnode = new List(data);
                this.next = newnode;
        }
        public List next() {
                return this.next;
        }
        public Object get() {
                return this.data;
        }
        public static void main(String[] args) {
                String firstData = "Test ";
                List newList = new List(firstData);
                String secondData = "list";
                newList.append(secondData);
                while(newList!=null) {
                   System.out.print(newList.get());
                        newList = newList.next();
                }
        }
}
```

Go

```
package main

import "fmt"

type list struct {
    next *list
    data string
}

func main() {
    newList := new(list)
    newList.data = "List "
    newList.next = new(list)
    newList.next.data = "test\n"
    for ; newList != nil; newList = newList.next {
        fmt.Printf(newList.data)
    }
}
```

Fibonacci Sequence

BASIC

```
10 INPUT "Enter a number ", n
20 LET a = 0
30 LET b = 1
35 IF n = a THEN GOTO 110 END IF
38 IF n = b THEN GOTO 130 END IF
41 FOR i = 1 to n
50   x = a + b
60   a = b
70   b = x
80 NEXT i
90 PRINT a
100 END
110 PRINT a
120 END
130 PRINT b
140 END
```

Lisp (Scheme)

```
(define (fib n)
  (define (fof n a b)
    (cond ((= n 0) a)
      ((= n 1) b)
      (#t (fof (- n 1) b
(+ a b)))))
  (fof n 0 1))
(define in (read-line))
(define num (string-
>number (if (eof-object?
in) (error "Requires
input") in)))
(if (number? num)
(displayln (fib num))
(error "Input not a
number"))
```

C

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int fibonacci(int n, int a, int b) {
    if (n == 0)
        return a;
    else if (n == 1)
        return b;
    else
        return fibonacci((n-1), b, (a +
b));
}

void main() {
    char in[256];
    fgets(in, 256, stdin);
    int inInt = atoi(in);
    printf("%d\n", fibonacci(inInt, 0,
1));

}
```

Java

```
import java.util.Scanner;
public class fib {
    public static int fibonacci(int n, int a, int b)
{
        if (n == 0)
            return a;
        else if (n == 1)
            return b;
        else
            return fibonacci((n-1), b, (a + b));
    }
    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        int in = s.nextInt();
        System.out.println(fibonacci(in, 0, 1));
    }
}
```

Go

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
    "strings"
)

func fibonacci(n int, a int, b int) int {
    if n == 0 {
        return a
    } else if n == 1 {
        return b
    } else {
        return fibonacci((n - 1), b, (a + b))
    }
}

func fib(n int) int {
    return fibonacci(n, 0, 1)
}

func main() {
    reader := bufio.NewReader(os.Stdin)
    in, _ := reader.ReadString('\n')
    in = strings.Replace(in, "\n", "", 1)
    inInt, _ := strconv.Atoi(in)
    fmt.Println(fib(inInt))
}
```

Resources Used

Louden, K. C., & Lambert, K. A. (2012). Programming languages: Principles and practice (3rd ed.). Boston, MA: Course technology.

Levy, S. (2010). Hackers: Heroes of the computer revolution. Sebastopol, CA: OReilly Media.