Amit Shravan Patel
UNI: ap3567
COMS W4115 (CVN)
Prof. Stephen Edwards
Project Proposal

AP++

Introduction

The purpose of my project is to create a new language, let's give it the eponymous name of *AP++*, that consists of a subset of features and syntactical sugar of the various languages for which I have developed a predilection; from Python's intuitive list slicing syntax to the ++ operator and {} scope blocks in C++. *AP++* will be much smaller in scope than the gamut of features offered in these modern programming languages, but will still be significant enough to be able to implement a number of algorithms.

Language Features

Scope
{} blocks for defining scope
; line termination

Comments
// single-line comments

Conditionals

```
if (conditional expression1) {
} else if (conditional expression2) {
} else {
}
```

Variables
Variables will be strictly typed in *AP++*,
e.g. ***int*** *x = 4;* ***bool*** *y = true; void foo(****int*** *x);*

2 basic primitive types: *Integer, Boolean*

<u>Integer</u> (keyword *int)*
Declaration:
e.g. *int x = 1; int x = y; int x = y + 1;*

Operators:

Operator	Description	Examples
+	Arithmetic Addition	x + y : between 2 vars x + 1 : between var and literal 1 + 2 : between 2 literals
-	Arithmetic Subtraction	x - y : between 2 vars

		x - 1 : between var and literal 1 - 2 : between 2 literals
/	Arithmetic Division	x / y : between 2 vars x / 1 : between var and literal 1 / 2 : between 2 literals
*	Arithmetic Multiplication	x * y : between 2 vars x * 1 : between var and literal 1 * 2 : between 2 literals
%	Modulus	x % y: between 2 vars x % 2: between var and literal 1 % 2: between 2 literals
++x	Unary Pre-Increment Operator	++x
x++	Unary Post-Increment Operator	x++

Boolean (keyword: *bool, values: {true, false}*)
Declaration:
e.g. bool x = true; bool x = y; bool x = conditional expression;

Operators:

<table>
<tr><th>Operator</th><th>Description</th><th>Examples</th></tr>
<tr><td>&&</td><td>Boolean AND</td><td>x && y</td></tr>
<tr><td>||</td><td>Boolean ||</td><td>x || y</td></tr>
<tr><td>!</td><td>Boolean NOT</td><td>!x</td></tr>
</table>

Variables declared outside of a scoped block {} will be considered global variables that live on the heap. All other variables will be allocated on the stack.

Lists
Python-style mutable lists.
Declaration:
e.g. int x[] = []; int x[] = [1, 2, 4]; int x[] = y[:];

Function	Description
list.append(x)	appends element x to end of list
list.insert(i, x)	inserts element x at ith index
list.pop([i])	pops ith element of list of i specified, else from end
list.clear()	clears all elements from list
[:] splicing	returns sublists of specified range, e.g. l[:] - returns new list with all elements from l l[4:] - returns elements from index 4 to last l[:4] - returns elements from index 0 to 4 index inclusive l[2:4] - returns elements from index 2 to 4 inclusive

Loops

```
while (conditional expression) {
}
```

I opted not to implement the for loop since the same functionality can be achieved with a while loop and local variables.

Functions

with return types

```
int foo(int a, int b) {
    return 0;
}
```

no return types:

```
void foo() {
}
```

There will be no support for default arguments, variable arguments or function overloading. Every param and return will pass by value, not reference.

Example Programs

Euclidean Algorithm (GCD)

```
int gcd(int x, int y) {
    if (y == 0) {
        return x;
    }
    return gcd(y, x % y);
}
```

Merge Sort

```
// merges two sorted sublists of arr[] (arr[0..m], arr[m+1..r]) in-place.
void merge(int[] arr, int l, int m, int r) {
    // temp lists for l and r sides
    int[] L = arr[0:m];
    int[] R = arr[m+1:r];

    // merge the temp lists back into arr[l..r]
    int i = 0;      // init index of 1st sublist
    int j = 0;      // init index of 2nd sublist
    int k = l;      // init index of merged sublist

    while (k < r) {
        if (j >= r || (i < m && L[i] <= R[j])) {
            arr[k] = L[i];
            i++;
        } else if (i >= m || (j < r && L[i] > R[j])) {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
}

void mergeSort(int[] list, l, r) {
    if (l >= r) {
        return;
    }
    int m = (l + (r-1)) / 2;
    mergeSort(list, l, m);
    mergeSort(list, m+1, r);
    merge(list, l, m, r);
}
```