Espresso

Somdeep Dey
Rohit Gurunath
Jianfeng Qian
Oliver Willens
Overview

- Introduction & Background
- Planning & Schedule
- Development Environment
- Syntax
- Architecture
- Testing
- Demonstration
Introduction

What is the Idea behind Espresso?

- A Object-Oriented programming language inspired by Java, stripped down and augmented.
Goals

- **Intuition.**
  
  Easy to just start coding for experienced programmers. A great platform to learn for beginners.

- **Transparency.**
  
  The LLVM IR code allows the user to understand the nuts and bolts of their program.

- **Flexibility.**
  
  Espresso allows for broad purpose use, rather than single-domain application. The language is portable and robust.
Development Environment

- Version Control
- Text Editing
- Operating System
- Virtualization
Project Timeline

Proposal: Sep 28 - Oct 26

“Hello World!”: Oct 27 - Nov 20


Comprehensive Pipeline: Dec 12 - Dec 20
Git History

129 Commits
Guidelines

- **Time Management**
  - Start the project early

- **Cooperation**
  - Teamwork and integration

- **Communication**
  - Avoid doing the same work

- **Software Tools**
  - Efficiency improvement
Syntax

Comments

//This is an Espresso Comment
/

So is this

*/

Operators

+  //add
-  //sub
*  //mult
/  //div
=  //assign
== //eq
!-  //neq
<  //lt
<=  //leq
>  //gt
>=  //geq
&&  //and
||  //or
!  //not

Arrays

int[10] arr;
Arr = {1,2,3,4,5,6,7,8,9,10};
Float[1] precise_arr;
precise_arr[0] = 0.0002;
int i;
for (i=1; i<10; i++){
  print_int(i);
}

int x = 0;
while(i<10){
  print_int(x);
  x++;
}

while (i < 2){
  print_int(item);
  break;
}

for (int i=0; i<4; i++){
  if (arr[i] > 0)
    print_int(data[i]);
}

int first_positive(int[] arr){
  for(int i=0; i<4; i++){
    if(arr[i] > 0)
      return arr[i];
  }
  return -1;
}

Class BankCount{
  int saving;
  String name;
  BankCount(class BankCount self, String n, int a){
    self.name = n;
    self.Saving = a;
  }

  bool withdraw(class BankCount self, int a){
    if (a < 0){
      return false;
    } else if(self.saveing > a){
      self.saveing -= a;
      return true;
    }
  }
}
class work
{
    int a;
    void main()
    {
        int b;
        int c;
        int d;
        int[10] arr;
        this.a = 100;
        class animal an;
        lambda : char lfunc(char a) {
            return a;
        }
        print_char (an.getChar(lfunc));
    }
}

class animal
{
    char b;
    bool x;

    char getChar(lambda lfunc) {
        return #lfunc('a');
    }

    int perform()
    {
        int i;
        i = 5;
        i = 1;
        return i*2;
    }
}
Architecture
# Testing

## Our MO: Test-Driven Development

<table>
<thead>
<tr>
<th>Unit Testing</th>
<th>Integration Testing</th>
<th>Automation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small test programs were written throughout the development process, designed to test the most recently added feature.</td>
<td>We created a large and comprehensive test suite, built to test features we didn’t think of during the development process, and to make sure the newest feature doesn’t negatively affect any of the previous ones.</td>
<td>/testall.sh</td>
</tr>
</tbody>
</table>