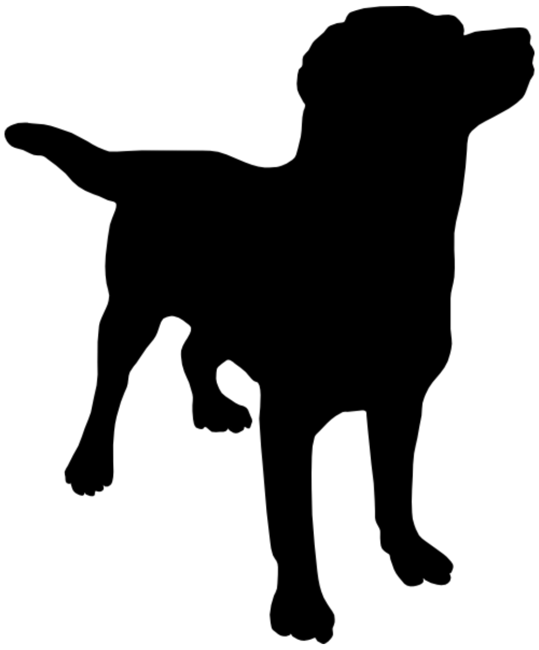


PAL : PDF Automation Language

Programming Languages and Translators - Spring 2016
Prof. Stephen Edwards

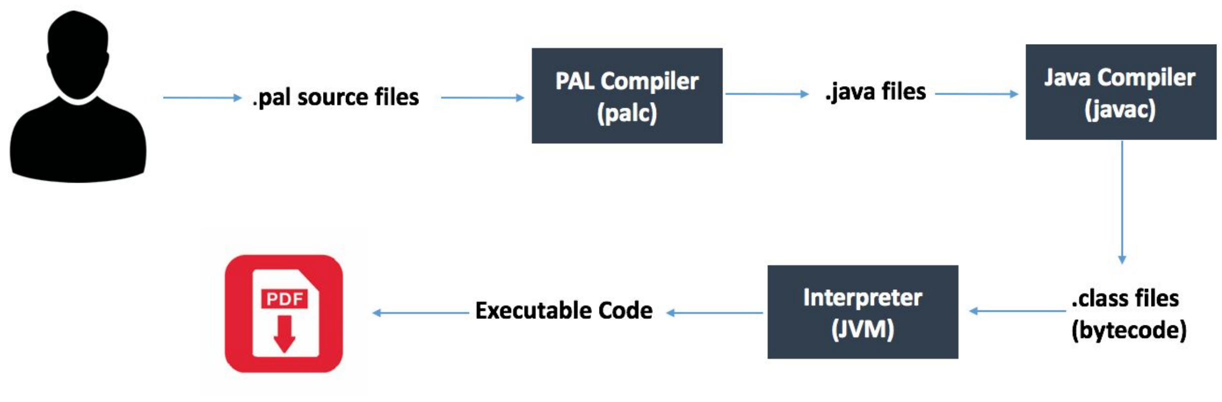
Anshuman Singh as4916@columbia.edu
Diksha Vanvari dhv2108@columbia.edu
Vinay Gaba vhg2105@columbia.edu
Viral Shah vrs2119@columbia.edu

I. Introduction:

Portable Document Format (PDF) is the file standard for the electronic exchange of documents. According to estimates by Adobe executives, there might be up to 2.5 trillion PDF documents existing in the world. The reason for its popularity is its platform-agnostic behaviour of passing and sending information that won't be skewed or altered. Our aim is to expand the range of operations performed on this popular data source through the means of PAL. There are many solutions available which are similar in nature to PAL but very often they do not fulfill the exact functionality as needed and are generally complicated, which requires a learning curve. We intend to simplify these interactions with PAL while at the same time also enable powerful operations which can fulfill operational needs.

II. Language

The PAL compiler will be written entirely in OCaml. The compiler will convert the PAL source code into equivalent JAVA code, that eventually will be compiled by the Java compiler and executed through the Java Virtual Machine. This will ensure a high portability on various platforms, even handheld and portable ones.



We will be using the following libraries to carry out PDF operations after the code has been compiled down to Java:

- Apache PDFBox

- iText

III. Proposed Use Cases

a. Create PDFs from text files.
b. Extract text from an existing PDF file.
c. Merge multiple PDFs into a single PDF.
d. Edit PDFs at a page level to generate PDFs with pages added, removed or swapped.
e. Split an existing PDF file into multiple PDF files, separated at specified page numbers or specified intervals.
f. Read a dataset from a data source (relational database) to be represented as tables in the resultant PDF.
g. Generate statistical representations from queried datasets to be represented as charts in the resultant PDF.
h. Advanced use cases can be supported via libraries written in the proposed language using the existing language features.

IV. Language Features

a. Comments

1. # must precede single line comments.

b. Variable Declaration

1. <identifiername> : type
2. <identifiername1>, <identifiername2> : type

c. Primitive Data Types

Data Type	Example
boolean	booleanVariable : boolean = true;
int	intVariable : int = 42;
char	charVariable : char = ‘c’;
string	stringVariable : string = “Hello World!”;

float	floatVariable : float = 42.0;
pdf	pdfVariable : pdf;
blob	blobVariable : blob;
dataset	datasetVariable : dataset;
page	pageVariable : page;
list	listVariable : list type

d. **Operators** : Operators will be overloaded and perform the following functions:

1. Operator '+'

   -
     ```
     pdf3 = pdf1 + pdf2;
     ```

     Merge the pages of pdf1 and pdf2 to form a new pdf pdf3.

   -
     ```
     pdf2 = pdf1 + pageN;
     ```

     Add a new page at the end of pdf1 to form a new pdf pdf2.

   -
     ```
     page3 = page1 + page2;
     ```

     Merge the blobs of page1 and page2 to form a new page page3.

2. Operator '-'

   -
     ```
     int n = 1;
     pdf2 = pdf1 - n;
     ```

     Removes page1 from pdf1 to form a new pdf pdf2.

3. Operator '/'

   -
     ```
     int n = 2;
     list1 : list = pdf1 / n;
     ```

     Splits a pdf at the specified integer page number. Returns a list of two pdfs, with pdf list1[1] consisting of the first 'n' pages and pdf list[2] consisting of the remaining pages.

4. Operator '%'

   -
     ```
     int n = 3;
     list: list1 = pdf % n;
     ```

     Splits a pdf into multiple pdfs of 'n' pages each. Returns a list of pdfs each of 'n' pages and the last pdf in the list consisting of the remainder pages.

5. Operator '<>'

   -
     ```
     int p1 = 1;
     int p2 = 2;
     pdf1(p1<>p2);
     ```

     Reorders the pdf by swapping the p1 and p2 pages.

e. I/O

1. `print s : string;`

   Prints the string to the output stream.

2. `scan s : string;`

   Reads as input a string from the input stream.

f. Functions : The entry point into the code is through the function start(). Other functions supported by the language are:

1. `split(pdf, int , int) : list;`

   Returns a pdf with pages between the two page numbers inclusive.

2. `save(pdf, string , string) : int;`
   Saves all the changes made to a given pdf. Writes the changes to the existing pdf or creates and writes a new pdf with the given

author name.

3. `createBlob(string, string) : blob;`

   Creates and returns a blob from the given location of a text file and assigns the font value of the particular blob.

4. `createPage(list) : page;`

   Creates and returns a page from a given list of blobs.

5. `createPDF(list) : pdf;`

   Creates and returns a pdf from a given list of pages.

6. `createDataset(string, string) : dataset;`

   Creates and returns a dataset by connecting to a data source using the connection string and the query supported by the data source.

7. `createChart(dataset, string, string) : blob;`
   Creates and returns a blob with the given chart type and chart title.

V. Programming Features

a. If Loop

```
if (<condition>) then {
      <statement1>
      <statement2>
} else {
      <statement3>
      <statement4>
}
```

If the condition is true, then it executes the statements in the first block as limited by the '{}' parentheses, else it executes the statements in the second block as limited by the '{}' parentheses.

b. For Loop

```
int n = 10;
for (int i = 1; i <= 10; i++) do {
      <statement1>
      <statement2>
}
```

While the condition mentioned by the second expression is true, the loop continues iterations, each time executing the statements in the block following 'do' limited by the '{}' parentheses.

c. While Loop

```
int 1 = 1;
int n = 10;
while (i != n) do {
      <statement1>
      i++;
}
```

While the condition mentioned by the expression is true, the loop continues iterations, each time executing the statements in the block following 'do' limited by the '{}' parentheses.

d. Main Function

```
main() : int {
      <statement1>
      <statement2>

      return 0;
}
```

Every program must have a main function. The program starts execution from the main function. On successful execution, the main function returns an integer value as specified in the return statement, else it returns -1.

e. User Defined Functions

```
function_name (parameter : parametertype) : returntype {
      <statement1>
```

```
        <statement2>

        return result;
}
```

A user defined function is represented by the above syntax. It consists of the function name followed by the function parameters and finally by the function return type.

VI. Example Program

a. Reading from a text file and writing to a pdf

```
main(){

#Creating blob from text file stored on local drive
textBlob : blob = createBlob("/path/test.txt","fontName.ttf");

#Creating a list variable and adding a blob to it
blobList : list blob;
blobList += textBlob;

#Creating a page and passing a list of blobs
page1 : page = createPage(blobList);

#Creating a list variable and adding a page to it
pageList : list page = {page1};

#Creating a pdf variable and passing a list of pages
pdf1 : pdf = createPDF(pageList);

#Saving pdf to given file path
save(pdf1,"path/filename.pdf");

}
```

b. Creating a pdf from the dataset returned by running a SQL query

```
main(){

#Creating a dataset by running a SQL query
resultDataset : dataset = createDataset("connectionString","SELECT * FROM
TABLE_NAME");

#Creating a blob of type table
tableBlob : blob = createChart(resultDataset,"TABLE","Table Title");

#Creating a blob of type bar chart
chartBlob : blob = createChart(resultDataset,"BAR_CHART","Chart
Title","X_Axis","Y-Axis");

#Creating a list variable and adding blobs to it
blobList1: list blob;
blobList1 += tableBlob;

blobList2: list blob;
blobList2 += chartBlob;


#Creating a page and passing a list of blobs to render the table
tablePage : page = createPage(blobList1);

chartPage : page = createPage(blobList2);

#Creating a list variable and adding a page to it
pageList: list page = {tablePage,chartPage};

#Creating a pdf variable and passing a list of pages
pdf1 : pdf = createPDF(pageList);

#Saving pdf to given filepath
save(pdf1,"path/filename.pdf");

}
```

c. Page Manipulation operations on a PDF

```
manipulate PDF(pdf1 : pdf, pdf2 : pdf){

#Appends pdf2 to pdf1 and creates a new pdf
pdf3 : pdf = pdf1 + pdf2;

#Saving pdf to given filepath
save(pdf3,"path/filename1.pdf");
#Get a list of all pages in pdf1
pageList1 : list page = getPages(pdf1);

#Appends only the first page from pdf1 to pdf2
pdf4 : pdf = pdf2 + pageList1[1];

#Saving pdf to given filepath
save(pdf4,"path/filename2.pdf");

#Get a list of all pages in pdf2
pageList2 : list page = getPages(pdf2);

#Appends only the first page from pdf1 to the first page from pdf2
pdf5 : pdf = pageList[2] + pageList1[1];

#Saving pdf to given filepath
save(pdf5,"path/filename3.pdf");

}
```