text++

Final Report

Joi Anderson - jna2123 // Manager + Tester
Klarizsa Padilla - ksp2127 // Language Guru + Tester
Maria Javier - mj2729 // System Architect + Tester

Contents

1. Introduction

---

text++ is a markup language designed for the production of technical documentation in an intuitive programming form. Unlike other templating languages like LaTeX, text++ is a markup language with algorithmic computing capabilities, allowing programmers to write documents as efficiently as they would write code.

2. Language Tutorial

To begin writing a document in text++, your document must include the def void start() {}. The function named start is a special function in all text++ programs; it is the function called when the program is run. The start function does not need to be called explicitly. The execution of all text++ programs begins with the start function, regardless of where the function is actually located within the code. The document will begin with a first page so you may begin calling write, or textout functions without first calling the function addPage. When you would like to begin writing on a new page, you simply call addPage. If you write def void start, a document will be created with a new page. Outside of start you may declare functions and call them inside of start. You may also create a function and call that function inside of start. Note: you can declare a variable outside of start but you cannot initialize it to a value.

2.1 Environment Setup

text++ was developed in OCaml. Before using text++ to program, make sure that Ocaml is installed properly. To do this, follow these steps:

Step 1: Install Homebrew

```
$ /usr/bin/ruby -e "$(curl -fsSL  https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Step 2: Install Opam and Configure OPAM

Opam is the OCaml Package Manager to install Ocaml packages and libraries. For installation instructions, see: https://opam.ocaml.org/doc/1.1/Advanced_Install.html

Step 3: Install libharu

Additionally, text++ utilizes the libharu library. Haru is a free, cross platform, open-source software library for generating PDF.

Installing libHaru on Linux/Unix is as easy as this:

```
./configure && make && make install
```

If you're using a Git checkout or a Github tarball, don't forget to run

```
./buildconf.sh
```

in order to create `./configure` script.

For more detailed instructions: https://github.com/libharu/libharu/wiki/Installation

2.3 Environment Setup

See more detailed instructions here:
https://github.com/libharu/libharu/wiki/Installation

3. Language Manual

---

1. Introduction

text++ is a markup language designed for the production of technical documentation in an intuitive programming form. Unlike other templating languages like LaTeX, text++ is a markup language with algorithmic computing capabilities, allowing programmers to write documents as efficiently as they would write code.

1.1 Software Used

The bulk of our code is written in OCaml and compiles to LLVM. We also have a file written in C to make use of the external library LibHaru, which is used in our display/PDF functions. The open source library for generating PDF files (libHaru) is run through a shell script that calls our code and the LLVM interpreter. (We also used the LLVM interpreter frequently when testing individual cases.) To compile to rendering mode, we use the run.sh shell script, which in turn calls a tpp source program. We used github for version control and for the repository. Each team member pulled code to their own computer or virtual machine and used their own choice of IDE.

1.2 LibHaru Overview

libHaru is a free, cross platform, open source library for generating PDF files. text++ utilizes the following features of LibHaru:

- Generating PDF files with lines and text.
- Text placement via a coordinate system.
- Embedding Type1 font and TrueType font.

libHaru is written in ANSI C, so theoretically it supports most of the modern operating systems.

2. Lexical Conventions

This section covers the text++ lexical convention for comments and tokens. There are six kinds of tokens: identifiers, keywords, constants, strings, operators, and separators. Blanks, tabs, newlines, and comments separate tokens, but they otherwise have no syntactic significance.

2.1 Comments

Single or multi-line comments start with the /* characters and terminate with the */ characters, ignoring all other characters encapsulated between the start and terminating characters.

```
/* This is a comment.
    It can have multiple lines. */
```

2.2 Identifiers

An identifier is a sequence of letters and digits, and the first character must be alphabetic. Identifiers must start with an alphabetic character, including the '_' character. Identifiers are case-sensitive.

2.3 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise.

- `if`
- `else`
- `for`
- `while`
- `return`
- `int`
- `bool`
- `float`
- `string`
- `void`
- `true`
- `false`
- `def`

2.3.1 Type-specifiers

Type-specifiers include bool, int, float, and string.

2.4 Data Types

There are three data types each with their own type, form and value :

2.4.1 Integer

An integer literal is a sequence of digits, represented by characters [0-9]. Integer constants have type `int`.

2.4.2 Float

A floating constant consists of an integer part, a decimal point, and a fraction part. Float constants have type `float`.

2.4.3 String

Strings are marked with double quotes.

2.5 Operators (Overview)

An operator character signifies that an operation should be performed. The operators `[]`, `()`, and `{}` are used to encapsulate expressions and must occur in pairs.

Operator can be one of the following:

`+ - * / % = == < <= > >= !|| -- ++`

For more on operators, please reference section 4 of this guide.

2.6 Separators

A separator is a symbol between each element. There is a single separator token in text++. The separator token in this language is a ‘,’ and whitespace is ignored. Separators are allowed in the following syntax:

Argument Separation: `myFunction( x, y)`

3. Declarations

3.1 Function Declarations

Functions are declared as:

```
def returnType functionName(type parameter, type parameter2){
      // local variables
      // statements
}
```

3.2 Variable Declarations

Variables are declared as:

```
type variableName;
variableName = expression;
```

A variable may have its value updated, as long as its type remains consistent.

```
int a;
a = 23;
a;                  // 23
a = “word”;       // Compiler error.
```

4. Expressions

Precedence of operators follows the following order of operations: Grouping symbols, Multiplication, Division, Addition, Subtraction. Text++ is a left-associative language (evaluated left to right, after the application of order of operations).

4.1 Primary Expressions

4.1.1 Identifier

An identifier (like a variable) is a primary expression whose type is required to be defined in its declaration.

4.1.2 String Literal

A string literal is a sequence of zero or more characters enclosed within quotation marks. A string literal is a primary expression.

4.2 Unary Operators

Not

```
! expression
```

Logical negation operator. Applicable for type boolean.

Increment

```
expression ++
```

The left-value expression is incremented. Applicable to type int.

Decrement

```
expression --
```

The left-value expression is decremented. Applicable to type int.

4.3 Multiplicative Operators

Multiplication

```
expression * expression
```

The binary * operator indicates multiplication. Applicable to type int and float.

Division

```
expression / expression
```

The binary / operator indicates division. Applicable to type int and float.

Modulus

```
expression % expression
```

The binary % operator yields the remainder from the division of the first expression by the second. Both operands must be int. The remainder keeps the sign of the dividend.

Concatenation

```
expression ^ expression
```

To concatenate two strings on a single line, use the concatenation operator (a single '^'). Applicable to type string.

4.4 Additive Operators

Addition

```
expression + expression
```

The result is the sum of the expressions. Applicable to type int and float.

Subtraction

```
expression - expression
```

The result is the difference of the operands. Applicable to type int and float.

4.5 Relational Operators

```
expression < expression
expression > expression
expression <= expression
expression >= expression
```

The operators < , > , <= , and >= all yield false if the relation is false and true if the relation is true.

4.6 Equality Operators

Equal and Not Equal

```
expression == expression
expression != expression
```

The == and the != operators are analogous to the relational operators except for their lower precedence. Thus `a < b == c < d` is true whenever `a < b` and also `c < d`.

4.7 Boolean Operators

And

```
expression && expression
```

The && operator returns true if both its operands are true, false otherwise. The second operand is not evaluated if the first operand is false.

Or

```
expression || expression
```

The || operator returns true if at least one of its operands is true, false otherwise.

5. Statements

Statements are executed in sequence.

5.1 End of Statement

The end of each statement is marked by a single ';'.

5.2 Expression Statements

The majority of statements are expression statements, taking the form:

```
expression
```

These statements are usually assignments or function calls.

5.3 Conditional Statements

```
if(expression) {
   statement;
}

else {
   statement;
}
```

If the expression is true, the (first) statement is executed. If the expression is false and there is an else, the second statement is executed. The elseless if problem is resolved by attaching an else to the last encountered if.

5.4 While Statements

```
while(expression) {
   statement;
}
```

The statement is executed as long as the expression is true. The evaluation of the expression occurs after each execution of the statement.

5.5 For Statements

```
for(expr1; expr2; expr3) {
   statement;
}
```

expr1 specifies initialization for the loop, expr2 is a test condition (evaluated before each iteration), and expr3 is an increment specification. The loop exits when expr2 is false.

5.6 Return statements

```
return
return (expression)
```

A function returns to its caller via a return statement. The second case returns the value of the expression. If the type expected by the caller does not match that of the return statement, an error will be thrown.

6. Scope Rules

6.1 Variable Scope

Variables declared outside of functions have global scope and can be accessed anywhere within the program. If declared within a function, variables only remain in scope for the duration of the function's execution. Parameters passed into a function as arguments are declared as local variables within the scope of the function.

6.2 Function Scope

A function may not be called before it has been declared. All functions have global scope by default.

6.3 Function Call

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

```
nameOfMethod (argument1, argument2, argument3) ;
```

7. Primitives

7.1 Page Creation

This function creates a new page and adds it after the last page of a document. The function expects zero arguments and returns no value.

```
addPage();
```

7.2 Text

italic: This function enables the user to change the current font to italics by calling italic(). The function expects zero arguments and returns no value. Upon its call, the italics style with persist until it is changed with a bold() or regular() call.

```
italic();
```

bold: This function enables the user to change the current font to bold by calling bold(). The function expects zero arguments and returns no value. Upon its call, the bold style with persist until it is changed with an italic() or regular() call.

```
bold();
```

regular: This function enables the user to change the current font from italics or bold back to standard font by calling regular(). The function

expects zero arguments and returns no value. The regular font style is set by default in a new document. This regular font styling will continue from the start of the document and upon its call until it is changed with a bold() or italic() call.

```
regular();
```

changeFontSize: Sets the size, font and style of the current font. Two arguments, the font name and font size, are required. The font options are: Helvetica, Times, and Courier. There are no number restrictions on the font size.

```
changeFontSize(string font, int size);
```

changeColor: This function sets the color of the text. The set color will persist from its call until the color is changed again. The changeColor function requires three parameters r (red), g (green), b (blue)-- the level of each color element. The argument values must be float values between 0 and 1.

```
changeColor(float r, float g, float b);
```

moveTo: Sets the current position for text. Sets the start point for the path to the point (x, y). Valid x and y coordinates are not limited by the page width and height.

```
moveTo(int x, int y);
```

textWidth: changes the width of a page where text can be written. This function expects a integer value less than the set page width.

```
textWidth(int val);
```

drawLine: prints a line from a specified coordinate position to a particular end coordinate. This function requires four parameters, where both the starting and ending x and y coordinates must be coordinates that lay within the set page width and height to be displayed otherwise the overflow will be cut off at the page limits.

```
drawLine(int beginX, float beginY, int endX, int endY);
```

drawRectangle: prints a rectangle from a specified position to a specified width and height. This function requires four parameters, where both the starting x and y coordinates, along with the coordinates after the addition of the width and height integers, must be coordinates that lay within the set page width and height. The width and height parameters may be negative and are oriented around the specified x and y point.

```
drawRectangle(int beginX, float beginY, int width, int
height);
```

textout: This function enables a user to write a piece of text at specific x, y coordinate. This function does handle text wrapping.

```
textout(int x, int y, string myText);
```

write: This function enables a user to write a piece of text at the current position of the cursor. This function does handle text wrapping.

```
write(string myText);
```

getPageHeight: This function takes zero parameters and returns the height of the page.

```
getPageHeight();
```

getPageWidth: This function takes zero parameters and returns the width of the page.

```
getPageWidth();
```

pageNumber: This function takes two parameters (an x coordinate and a y coordinate) and prints the page number of the current page at the passed x and y coordinates.

```
pageNumber(int x, int y);
```

7.3 Alignment

left: This function sets the alignment of the text to left. The default alignment at the creation of a new document is also left. Left alignment persists from the start of the document and its call until it is set to a different alignment with a center() or right() call. Zero arguments are expected in this function.

```
left();
```

right: This function sets the alignment of the text to centered, and this alignment persists until it is set to a different alignment with a left() or center() call. Zero arguments are expected in this function.

```
right();
```

center: This function sets the alignment of the text to centered, and this alignment persists until it is set to a different alignment with a left() or right() call. Zero arguments are expected in this function.

```
center();
```

8. Standard Library

8.1 Title

pageTitle: This function takes a string as it argument and centers it in large font at the current position on the current page from which it is called.

```
pageTitle(string myTitle);
```

8.2 Drawing

horizontalLine: This function draws a line horizontally across the width of the page at the current position of the text. Zero arguments are expected in this function.

```
horizontalLine();
```

table: This function draws a table on the page based on four arguments: the number of rows in the table, the number of columns in the table, and the table width and length. If the width or height go beyond the dimensions of a page, the table by default will fill the page in that dimension.

```
table(int row, int column, int tableWidth, int tableHeight);
```

8.3 Headings

heading: This function takes string and formats it based on HTML heading standards. Headings are defined with the heading1 to heading6 calls. heading1 headings should be used for main headings, followed by heading2 headings, then the less important heading3, and so on. This function takes no parameters and the style will persist for all text until the size of current font is reset using either the heading or changeFontSize functions.

```
heading1();
heading2();
heading3();
heading4();
heading5();
heading6();
```

9. PDF Defaults

Defaults in text++ documents are portrait style layout, with a width of 595 pixels and a height of 842 pixels. It is also important to note that in our documents the bottom left corner of the page is the origin of the x-y coordinate system. The default font and size for the text on the page is: Helvetica, 12.

Sample Logo Program:

```
def void logo(){

   int i;
   int ph;
   int pw;
   int offsetX;
   int offsetY;

   pw = getPageWidth();
   ph = getPageHeight();
   offsetX = 10;
   offsetY = 100;

   for (i = 0; i < 3; i = i + 1){
      drawRectangle(pw/2 - offsetX, ph - offsetY, 50, 50);
      offsetX = offsetX + 5;
      offsetY = offsetY + 5;
    }
    heading1();
    textOut( "G" , pw/2 - 5, ph - 90, 0);
}

def void start(){
  logo();
}
```

4. Project Plan

4.1 Process Used

text++ was planned in two major settings: roughly biweekly meetings with Professor Edwards and weekly team meetings. In the first portion of the semester, these meetings served to address the broad language goals and working on the milestone assignments. However, as the semester progressed, so did the technical specificity of our meetings and task lists. Our team then set goals each week in the form of agile sprints. Our sprints were based on the feedback of Professor Edwards and the remaining tasks for our language. Using this method were able to make consist progress towards our goals while ensuring the quality of completed tasks.

4.2 Programming Style Guide

- Indent to indicate scope.
- Writing multiple statements on the same line is discouraged.
- Continuation lines should use a hanging indent.
- Surround assignment, boolean, and concat operators with a single space on either side.
- Wrap lines at 120 characters.

4.3 Project Timeline

9/11 — Assigning team roles and brainstorming ideas for the project.
9/14 — Language brainstorming and decisions on language.
9/16 — Brainstorming applications of language.
9/17 — Discuss more ideas to smooth out usage.
9/18 — Proposal drafting.
9/19 — Submission of proposal and searching of open source PDF generator
10/5 — Discuss and set up environment which includes Virtual Box and LibHaru.
10/7 — Resolved issues with installing the same environment. Explored LibHaru
10/12 — Began the implementation of AST, parser, and scanner. Revisited grammar errors.
10/15 — Continuation of AST, parser, and scanner. Submitted.
10/28 — Worked on issues in the grammar. Hammered down core syntax of the language.
11/2 — Began creation of the type checker. Decided what additional standard library functions we would need.
11/4 — Started working on the Hello World Demo. Encountered shared folder issues.

11/11 — Continued implementing syntax of text++. Worked to resolve Issues with Hello World Demo. Start implementing basic standard library functions.
11/16 — Hello World Demo Due.
11/18 — Work to resolve integration issues and bugs with LibHaru.
11/25 — Start work on semantic checker including for assignment and binary operators
11/30 — Implementation of strings. Starting implementing PDF functions in standard library.
12/2 — Continued implementing PDF functions in standard library.
12/7 — Working on passing text writing function. Continue implementing other PDF functions in standard library.
12/9 — Resolving issues in translator. Continue implementing standard library. Wrote wrapper functions for LibHaru functions.
12/14 — Implemented error messages.
12/16 — Concluded the implementation of the write function and other standard library functions. Worked on the development of sample programs.
12/17 — Writing final report.
12/19 — Final Report Due. Concluded editing final report.

4.4 Roles and Responsibilities

Joi Anderson: Project Management and Tester

Maria Javier: System Architect and Tester

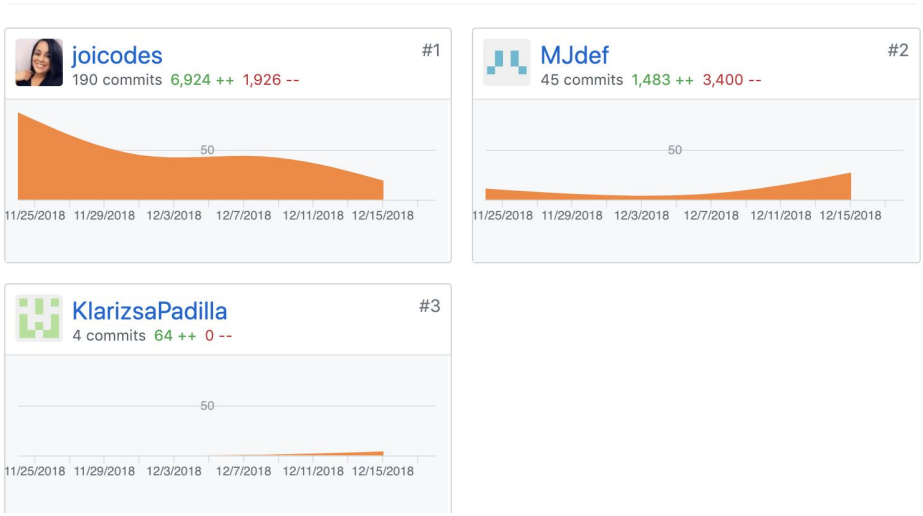
Klarizsa Padilla: Language Guru and Documentation

4.5 Software Development Environment Used

We used the following programming and development environment:

- Libraries and Languages: Ocaml version 4.07, including Ocamlyacc version 4.07 and Ocammllex version 4.07 extensions. LLVM Ocaml version 5.0. gcc version 9.0
- Software: Development was done on SublimeText
- OS: Development was done on OSX 10.13 and on Ubuntu 18.04.

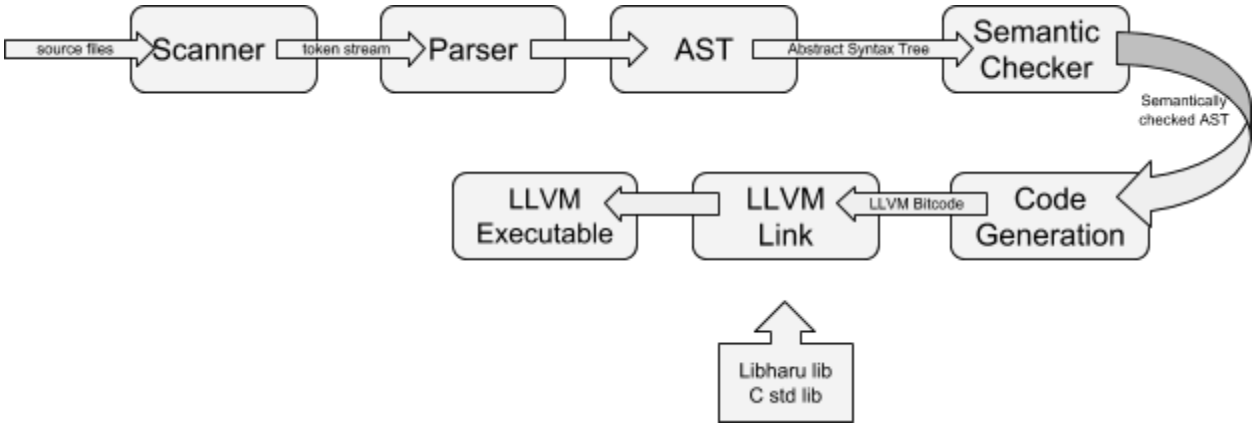
4.6 Project Log



* Note: Joi's number of commits is higher than Maria's because Joi was unable to configure a shared file.

5. Architectural Design

5.1 Block Diagram



5.2 Interfaces Between the Components

Scanner
The scanner.mll was implemented using ocamellex. The scanner takes in source files as a symbol stream and tokenizes it. This tokenization process provides syntax checking, and rejects illegal symbols. The scanner is responsible for stripping out information that is not necessary (e.g. comments and whitespace) for the the compilation process.

Parser and AST
The ast.ml and parser.mly files were implemented using using ocamlyacc. The token stream produced by the scanner is then input to the parser. The parser produces an abstract syntax tree (AST) from the input. The abstract syntax tree describes the structure of the program. An acceptable structure of the AST is provided to the parser.mly by the ast.ml file. In this parsing process further synat checking is performed. Programs that do not meet AST syntactic requirements are then rejected.

Semantic Checker
The analyzer.ml and sast.ml files form the text++ analyzer and semantic checker. These files, the semantic checker, were implemented in OCaml. The input into the semantic checker is the the AST produced by the parser. The output of the semantic checker is a semantically analyzed abstract syntax tree (SAST). The SAST, in addition to describing the overall program structure,

contains information attached in the analyzer. The sast.ml file provides the acceptable form of the SAST to the analyzer.ml file. In this phase, the input undergoes rigorous semantic checking. Programs that violate declaration, type, order, or any text++ requirements are rejected. In this phase built-in variables and functions are added to the sast by the analyzer.

Code Generator
The generator.ml file was implemented in OCaml. The input into the code generator is the SAST produced by the analyzer, in turn C code is produced. The majority of the code this file generates is hard coded in codegen.ml. However, the codegen also draws on code from our standard library - written in C utilizing libharu (a third-party library).

text++ Library
The text++ Library was implemented in C - the associated files are hello.c. The text++ library makes use of the libharu library for carrying out some of the more complex conversions from text++ to C code in the generator (e.g. rendering a pdf).

5.2 Who Implemented Each Component

Component	Contributors
Scanner	Joi, Maria, Klarizsa
Parser	Joi, Maria, Klarizsa
AST	Joi, Maria
SAST	Joi, Maria
Semant	Joi, Maria
Codegen	Joi, Maria
textPlusPlus.ml	Joi, Maria
hello.c	Joi, Maria

6. Test Plan

6.1 Source Language Programs

<test-while.tpp>

```
def void start(){
	int x;

	x = 0;
	while(x < 10){
		write("Falalalala lala la la");
		x = x + 1;
	}
}
```

<test-while.out>

```
Falalalala lala la la
Falalalala lala la la
Falalalala lala la la
Falalalala lala la la
Falalalala lala la la
Falalalala lala la la
Falalalala lala la la
Falalalala lala la la
Falalalala lala la la
Falalalala lala la la
```

<test-for.tpp>

```
def void start(){
	int x;

	for(x = 0;x < 4; x = x+1;){
		write("Do re me");
	}
```

```
}
```

<test-for.out>

```
Do re me
Do re me
Do re me
Do re me
```

6.2 Test Suites to Test Translators

All tests are stored in the /tests/ folder. Tests are split into a test suite and output based on name.

6.3 Why and How These Test Cases Were Chosen

text++ used MicroC's test source files as a foundation to automate the build of our own tests. The readMe provides instructions on how to run the test programs. An environment that already has Ocaml, ocamlbuild, LLVM, and opam is necessary. For this project, the developers utilized the VM image provided in order to replicate an environment with all of these installations. On top of this, Libharu is required because some of its library components are used in our language.

Libharu has the following dependencies: automake, autoconf, zlib1g-dev, libpng-dev, and libtools. Once these dependencies are installed, LibHaru should be installed successfully. Directions for doing this are on the project README. The final component especially needed to run tests is python-pdfminer. python-pdfminor contains the source code for a command line tool, pdf2txt which converts a pdf file into a valid ASCII file.

In order to execute the tests, run make in the project directory after downloading the tar file. Tests are all located in a tests directory. Each test tests a small component of the language in order to see if valid ASCII is produced. The test are named test-<name of component>.tpp. Make clean should clean any intermediary files that were created after running make.

The output of a successfully compiled text++ program is a PDF file with ASCII characters, stylized fonts, and lines for graphics. To check whether test programs were correctly rendered, the contents of the PDF files were extracted into a stream of ASCII characters using pdf2text. While many features of text++ were tested in the test suite, the test are primarily meaningful for the built-in functions involving changes to the outputted ASCII characters that were extracted from the document.

6.4 Automation Used in Testing

As aforementioned, the testing automation is based on the Micro-C testing suite. In the /tests/ folder the testall.sh script compiles and runs all *.cqm files. It then will look at the corresponding *.err or *.out file of the same name and compare the output of the file to the output of the script . Errors or differences that arise in compilation in are passed to stdout.

6.5 Division of Tasks

The testing gurus were responsible for unit testing the features that were assigned to them. Additionally, the testing gurus were responsible for coordinating to build appropriate integration tests.

7. Lessons Learned

7.1 Lessons Learned

Maria Javier

I've never made a language before and after completing this project I have a better understanding about language design. I've learned that it is extremely important to define the overall goal and the characteristics of a language early on so that when the time comes to write code, there are no disputes about syntax and or structure. There are definitely more improvements that could be made on our project and I have developed a greater understanding about how hard it is to place a group of characters onto a page in a stylized matter. It's more than just adding a string to a page and I faced a lot obstacles as I was trying to get wrapping to work correctly. Although I did struggle with Ocaml in the beginning, I now understand how powerful of a language it can be when constructing a compiler for a language. It is a headache to get everything working but when it works, it's beautiful.

Joi Anderson

My biggest lesson from creating the compiler was to do my research first. After researching typesetting and markup languages like TeX and LaTeX, I was able to make note of some of their language design decisions and useful features to use in our own language. TeX has an extensive word wrapping, page break, and hyphenation algorithm that we try attempted to implement in our language to provide users with features they would typically like to see in a language like text++.

Klarizsa Padilla

Our team started early but we spent a lot of time hashing out the ideas our project. It was challenging to decide syntax, and functionality because each of us has a different idea of what is "intuitive" or looks more appealing. A lot of our long discussions early in the semester were about what we wanted our language to do. I learned that you've got to make a decision, a solid one, and stick to it if you want to make progress as opposed to swaying back and forth in indecision.

7.2 Advice for Future Teams

See your advisor early, and check in often. If you are having a blocker it is important to recognize when too much time has been spent in the same stage and get help as soon as possible. Having weekly meetings scheduled with your adviser is also a good way to ensure you are making progress. Try to finish the scanner and the parser as soon as possible. Do not leave testing until the end. Testing helps with being able to catch weird edge cases and you don’t want to get caught in a really big bug right before the deadline.

8. Code Listings

Scanner

```
(* Ocamllex scanner for textPlusPlus *)

{ open Parser }

let digit = ['0' - '9']
let digits = digit+

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LBRACKET }
| ']'      { RBRACKET }
| ';'      { SEMI }
| ','      { COMMA }

(* Arithmetic Operators *)
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '='      { ASSIGN }

(* Relational Operators *)
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
```

```
(* Logical Operators *)
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }

(* Control Flow *)
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }

(* Keywords *)
| "int"    { INT }
| "bool"   { BOOL }
| "float"  { FLOAT }
| "string" { STRING }
| "void"   { VOID }
| "true"   { BLIT(true)  }
| "false"  { BLIT(false) }
| "def"    { DEFINE }
(* Literals and Identifiers *)

| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLIT(lxm) }
| '"' (['\x20'-'\x7E']* as lxm) '"' { STRLITERAL(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }
```

Parser

```
/* Ocamlyacc parser for textPlusPlus */

%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET COMMA PLUS MINUS
TIMES DIVIDE ASSIGN
%token NOT EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF ELSE FOR WHILE INT BOOL FLOAT STRING VOID DEFINE
%token <int> LITERAL
%token <bool> BLIT
%token <string> STRLITERAL
%token <string> ID FLIT
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT

%%

program:
  declarations EOF { $1 }

declarations:
   /* nothing */ { ([], [])               }
 | declarations variable_declaration { (($2 :: fst $1), snd $1) }
 | declarations function_declaration { (fst $1, ($2 :: snd $1)) }

function_declaration:
```

```
   DEFINE typ ID LPAREN parameters RPAREN LBRACE vdecl_list codeblock
RBRACE
     { { typ = $2;
       fname = $3;
       formals = List.rev $5;
       locals = List.rev $8;
       body = List.rev $9 } }

parameters:
    /* nothing */ { [] }
  | formal_list   { $1 }

formal_list:
    typ ID                   { [($1,$2)]     }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
    INT   { Int   }
  | STRING { String }
  | BOOL  { Bool  }
  | FLOAT { Float }
  | VOID  { Void  }

vdecl_list:
    /* nothing */    { [] }
  | vdecl_list variable_declaration { $2 :: $1 }

variable_declaration:
   typ ID SEMI { ($1, $2) }

codeblock:
    /* nothing */  { [] }
  | codeblock stmt { $2 :: $1 }

stmt:
    expression SEMI                               { Expr $1           }
  | RETURN expr_opt SEMI                          { Return $2         }
  | LBRACE codeblock RBRACE                       { Block(List.rev $2) }
  | IF LPAREN expression RPAREN stmt %prec NOELSE   { If($3, $5, Block([]))
}
  | IF LPAREN expression RPAREN stmt ELSE stmt      { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expression SEMI expr_opt RPAREN stmt
```

```
                                                    { For($3, $5, $7, $9) }
  | WHILE LPAREN expression RPAREN stmt              { While($3, $5) }

expr_opt:
    /* nothing */       { Noexpr }
  | expression          { $1 }

expression:
    LITERAL                           { Literal($1) }
  | STRLITERAL                        { StrLiteral($1) }
  | FLIT                                { Fliteral($1) }
  | BLIT                              { BoolLit($1) }
  | ID                                { Id($1) }
  | expression PLUS   expression      { Binop($1, Add,   $3)   }
  | expression MINUS  expression      { Binop($1, Sub,   $3)   }
  | expression TIMES  expression      { Binop($1, Mult,  $3)   }
  | expression DIVIDE expression      { Binop($1, Div,   $3)   }
  | expression EQ     expression      { Binop($1, Equal, $3)   }
  | expression NEQ    expression      { Binop($1, Neq,   $3)   }
  | expression LT     expression      { Binop($1, Less,  $3)   }
  | expression LEQ    expression      { Binop($1, Leq,   $3)   }
  | expression GT     expression      { Binop($1, Greater, $3) }
  | expression GEQ    expression      { Binop($1, Geq,   $3)   }
  | expression AND    expression      { Binop($1, And,   $3)   }
  | expression OR     expression      { Binop($1, Or,    $3)   }
  | MINUS expression %prec NOT        { Unop(Neg, $2)  }
  | NOT expression                    { Unop(Not, $2)  }
  | ID ASSIGN expression              { Assign($1, $3) }
  | ID LPAREN optional_arguments RPAREN { Call($1, $3)   }
  | LPAREN expression RPAREN          { $2 }

optional_arguments:
    /* nothing */  { [] }
  | arguments      { List.rev $1 }

arguments:
    expression                  { [$1] }
  | arguments COMMA expression  { $3 :: $1 }
```

AST

```
(* Abstract Syntax Tree *)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq
|
          And | Or

type uop = Neg | Not

type typ = Int | Bool | Float | Void | String

type bind = typ * string

type expr =
    Literal of int
  | StrLiteral of string
  | Fliteral of string
  | BoolLit of bool
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | Noexpr

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

type func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
    locals : bind list;
    body : stmt list;
  }

type program = bind list * func_decl list
```

```
(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | StrLiteral(l) -> l
  | Fliteral(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
```

```
string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_typ = function
    Int -> "int"
  | String -> "string"
  | Bool -> "bool"
  | Float -> "float"
  | Void -> "void"

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_fdecl fdecl =
  "def " ^ string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_program (vars, funcs) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

Semant

```
(* Semantic checking for textPlusPlus compiler *)

open Ast
open Sast

module StringMap = Map.Make(String)

(* Semantic checking of the AST. Returns an SAST if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (globals, functions) =

  (* Verify a list of bindings has no void types or duplicate names *)
  let check_binds (kind : string) (binds : bind list) =
    List.iter (function
      (Void, b) -> raise (Failure ("Illegal void " ^ kind ^ " " ^ b))
      | _ -> ()) binds;
    let rec dups = function
        [] -> ()
      |     ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
        raise (Failure ("Duplicate name" ^ kind ^ " " ^ n1))
      | _ :: t -> dups t
    in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
  in

  (**** Check global variables ****)

  check_binds "global" globals;

  (**** Check functions ****)

  (* Collect function declarations for built-in functions: no bodies *)

    let built_in_decls =

    StringMap.add "addPage"
     { typ = Int; fname = "addPage"; formals = [];
       locals = []; body = [] }
```

```
      (StringMap.add "left"
    { typ = Void; fname = "left"; formals = [];
      locals = []; body = [] }

      (StringMap.add "right"
    { typ = Void; fname = "right"; formals = [];
      locals = []; body = [] }

      (StringMap.add "center"
    { typ = Void; fname = "center"; formals = [];
      locals = []; body = [] }

      (StringMap.add "write"
      { typ = Void; fname = "write"; formals = [(String, "x")];
        locals = []; body = [] }

       (StringMap.add "textOut"
     { typ = Void; fname = "textOut"; formals = [(Int, "y"); (Int, "z");
(String, "x");];
       locals = []; body = [] }

       ( StringMap.add "moveTo"
     { typ = Void; fname = "moveTo"; formals = [(Int, "x"); (Int, "y")];
      locals = []; body = [] }



      (StringMap.add "bold"
    { typ = Void; fname = "bold"; formals = [];
      locals = []; body = [] }

       (StringMap.add "italic"
   { typ = Void; fname = "italic"; formals = [];
       locals = []; body = [] }

       (StringMap.add "regular"
   { typ = Void; fname = "regular"; formals = [];
       locals = []; body = [] }

       (StringMap.add "changeColor"
```

```
    { typ = Void; fname = "changeColor"; formals = [(Float, "x"); (Float,
"y"); (Float, "z")];
        locals = []; body = [] }

        (StringMap.add "changeFontSize"
    { typ = Void; fname = "changeFontSize"; formals = [(String, "x"); (Int,
"y")];
        locals = []; body = [] }



        (StringMap.add "drawLine"
    { typ = Void; fname = "drawLine"; formals = [(Int, "x"); (Int, "y");
(Int, "z"); (Int, "a")];
        locals = []; body = [] }

        (StringMap.add "drawRectangle"
     { typ = Void; fname = "drawRectangle"; formals = [(Int, "x"); (Int,
"y"); (Int, "z"); (Int, "a")];
        locals = []; body = [] }



        (StringMap.add "pageNumber"
        { typ = Int; fname = "pageNumber"; formals = [(Int, "x"); (Int,
"y");];
           locals = []; body = [] }



        (StringMap.add "getTextWidth"
     { typ = Int; fname = "getTextWidth"; formals = [(String, "x")];
        locals = []; body = [] }

        (StringMap.add "getPageHeight"
     { typ = Int; fname = "getPageHeight"; formals = [];
        locals = []; body = [] }

        (StringMap.add "getPageWidth"
     { typ = Int; fname = "getPageWidth"; formals = [];
        locals = []; body = [] }
```

```
        (StringMap.add "pageTitle"
     { typ = Void; fname = "pageTitle"; formals = [(String, "x");];
        locals = []; body = [] }

        (StringMap.add "table"
        { typ = Void; fname = "table"; formals = [(Int, "x"); (Int, "y");
(Int, "z"); (Int, "a")];
           locals = []; body = [] }


        (StringMap.add "heading1"
        { typ = Void; fname = "heading1"; formals = [];
           locals = []; body = [] }

        (StringMap.add "heading2"
        { typ = Void; fname = "heading2"; formals = [];
          locals = []; body = [] }

        (StringMap.add "heading3"
        { typ = Void; fname = "heading3"; formals = [];
            locals = []; body = [] }

        (StringMap.add "heading4"
        { typ = Void; fname = "heading4"; formals = [];
            locals = []; body = [] }

        (StringMap.add "heading5"
        { typ = Void; fname = "heading5"; formals = [];
          locals = []; body = [] }

        (StringMap.add "heading6"
        { typ = Void; fname = "heading6"; formals = [];
            locals = []; body = [] }

        (StringMap.add "getCurrentY"
        { typ = Int; fname = "getCurrentY"; formals = [];
            locals = []; body = [] }

        (StringMap.add "getCurrentX"
        { typ = Int; fname = "getCurrentX"; formals = [];
            locals = []; body = [] }
```

```
        (StringMap.add "getCapHeight"
        { typ = Int; fname = "getCapHeight"; formals = [];
            locals = []; body = [] }

        (StringMap.add "getLowHeight"
        { typ = Int; fname = "getLowHeight"; formals = [];
            locals = []; body = [] }

        (StringMap.add "getTextBytes"
        { typ = Int; fname = "getTextBytes"; formals = [(String, "x");
(Int, "y"); (Int, "z")];
            locals = []; body = [] }

        (StringMap.add "setRMargin"
        { typ = Int; fname = "setRMargin"; formals = [(Int, "y")];
            locals = []; body = [] }

        (StringMap.add "setLMargin"
        { typ = Int; fname = "setLMargin"; formals = [(Int, "y")];
            locals = []; body = [] }

        (StringMap.add "setTopMargin"
        { typ = Int; fname = "setTopMargin"; formals = [(Int, "y")];
            locals = []; body = [] }

        (StringMap.add "setBotMargin"
        { typ = Int; fname = "setBotMargin"; formals = [(Int, "y")];
            locals = []; body = [] } StringMap.empty

    ))))))))))))))))))))))))))))))))))

  in

  (* Add function name to symbol table *)
  let add_func map fd =
    let built_in_err = "The function " ^ fd.fname ^ " is a built in
function and may not be defined."
    and dup_err = "Duplicate function name: " ^ fd.fname
    and make_err er = raise (Failure er)
    and n = fd.fname (* Name of the function *)
    in match fd with (* No duplicate functions or redefinitions of
```

```
built-ins *)
         _ when StringMap.mem n built_in_decls -> make_err built_in_err
       | _ when StringMap.mem n map -> make_err dup_err
       | _ ->  StringMap.add n fd map
  in

  (* Collect all function names into one symbol table *)
  let function_decls = List.fold_left add_func built_in_decls functions
  in

  (* Return a function from our symbol table *)
  let find_func s =
    try StringMap.find s function_decls
    with Not_found -> raise (Failure ("The following function is undefined:
" ^ s))
  in

  (* let _ = find_func "main" in (* Ensure "main" is defined *) *)

  let check_function func =
    (* Make sure no formals or locals are void or duplicates *)
    check_binds "formal" func.formals;
    check_binds "local" func.locals;

    (* Raise an exception if the given rvalue type cannot be assigned to
       the given lvalue type *)
    let check_assign lvaluet rvaluet err =
       if lvaluet = rvaluet then lvaluet else raise (Failure err)
    in

    (* Build local symbol table of variables for this function *)
    let symbols = List.fold_left (fun m (ty, name) -> StringMap.add name ty
m)
                    StringMap.empty (globals @ func.formals @ func.locals
)
    in

    (* Return a variable from our local symbol table *)
    let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("The following variable is
undeclared: " ^ s))
```

```
    in

    (* Return a semantically-checked expression, i.e., with a type *)
    let rec expr = function
        Literal  l -> (Int, SLiteral l)
      | StrLiteral l -> (String, SStrLiteral l)
      | Fliteral l -> (Float, SFliteral l)
      | BoolLit l  -> (Bool, SBoolLit l)
      | Noexpr     -> (Void, SNoexpr)
      | Id s       -> (type_of_identifier s, SId s)
      | Assign(var, e) as ex ->
          let lt = type_of_identifier var
          and (rt, e') = expr e in
          let err = "Illegal assignment of a" ^ string_of_typ lt ^ " to a "
^
            string_of_typ rt ^ " in " ^ string_of_expr ex
          in (check_assign lt rt err, SAssign(var, (rt, e')))
      | Unop(op, e) as ex ->
          let (t, e') = expr e in
          let ty = match op with
            Neg when t = Int || t = Float -> t
          | Not when t = Bool -> Bool
          | _ -> raise (Failure ("Illegal use of unary operator with a " ^
                                 string_of_uop op ^ string_of_typ t ^
                                 " in " ^ string_of_expr ex))
          in (ty, SUnop(op, (t, e')))
      | Binop(e1, op, e2) as e ->
          let (t1, e1') = expr e1
          and (t2, e2') = expr e2 in
          (* All binary operators require operands of the same type *)
          let same = t1 = t2 in
          (* Determine expression type based on operator and operand types
*)
          let ty = match op with
            Add | Sub | Mult | Div when same && t1 = Int   -> Int
          | Add | Sub | Mult | Div when same && t1 = Float -> Float
          | Equal | Neq            when same               -> Bool
          | Less | Leq | Greater | Geq
                     when same && (t1 = Int || t1 = Float) -> Bool
          | And | Or when same && t1 = Bool -> Bool
          | _ -> raise (
            Failure ("Illegal binary operator " ^
```

```
                      string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                      string_of_typ t2 ^ " in " ^ string_of_expr e))
          in (ty, SBinop((t1, e1'), op, (t2, e2')))
      | Call(fname, args) as call ->
          let fd = find_func fname in
          let param_length = List.length fd.formals in
          if List.length args != param_length then
            raise (Failure ("The function expected " ^ string_of_int
param_length ^
                            " arguments in " ^ string_of_expr call))
          else let check_call (ft, _) e =
            let (et, e') = expr e in
            let err = "Illegal argument found " ^ string_of_typ et ^
              " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
            in (check_assign ft et err, e')
          in
          let args' = List.map2 check_call fd.formals args
          in (fd.typ, SCall(fname, args'))
    in

    let check_bool_expr e =
      let (t', e') = expr e
      and err = "Expected Boolean expression in " ^ string_of_expr e
      in if t' != Bool then raise (Failure err) else (t', e')
    in

    (* Return a semantically-checked statement i.e. containing sexprs *)
    let rec check_stmt = function
        Expr e -> SExpr (expr e)
      | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt b1, check_stmt
b2)
      | For(e1, e2, e3, st) ->
        SFor(expr e1, check_bool_expr e2, expr e3, check_stmt st)
      | While(p, s) -> SWhile(check_bool_expr p, check_stmt s)
      | Return e -> let (t, e') = expr e in
        if t = func.typ then SReturn (t, e')
        else raise (
        Failure ("Return gives " ^ string_of_typ t ^ " expected " ^
               string_of_typ func.typ ^ " in " ^ string_of_expr e))

          (* A block is correct if each statement is correct and nothing
             follows any Return statement.  Nested blocks are flattened. *)
```

```
      | Block sl ->
          let rec check_stmt_list = function
              [Return _ as s] -> [check_stmt s]
            | Return _ :: _   -> raise (Failure "Nothing may follow a
return statement.")
            | Block sl :: ss  -> check_stmt_list (sl @ ss) (* Flatten
blocks *)
            | s :: ss         -> check_stmt s :: check_stmt_list ss
            | []              -> []
          in SBlock(check_stmt_list sl)

    in (* body of check_function *)
    { styp = func.typ;
      sfname = func.fname;
      sformals = func.formals;
      slocals  = func.locals;
      sbody = match check_stmt (Block func.body) with
      SBlock(sl) -> sl
      | _ -> raise (Failure ("internal error: block didn't become a
block?"))
    }
  in (globals, List.map check_function functions)
```

Sast

```
(* Semantically-checked Abstract Syntax Tree *)

open Ast

type sexpr = typ * sx
and sx =
    SLiteral of int
  | SStrLiteral of string
  | SFliteral of string
  | SBoolLit of bool
  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of string * sexpr
  | SCall of string * sexpr list
  | SNoexpr

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt

type sfunc_decl = {
    styp : typ;
    sfname : string;
    sformals : bind list;
    slocals : bind list;
    sbody : sstmt list;
  }

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLiteral(l) -> string_of_int l
  | SStrLiteral(l) -> l
```

```
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SFliteral(l) -> l
  | SId(s) -> s
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SNoexpr -> ""
                          ) ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SFor(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
      string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt
s

let string_of_sfdecl fdecl =
  "def " ^ string_of_typ fdecl.styp ^ " " ^
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd fdecl.sformals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.slocals) ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
  "}\n"

let string_of_sprogram (vars, funcs) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_sfdecl funcs)
```

Textplusplus

```
(* Top-level of the textPlusPlus compiler: scan & parse the input,
   check the resulting AST and generate an SAST from it, generate LLVM IR,
   and dump the module *)

type action = Ast | Sast | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./textplusplus.native [-a|-s|-l|-c] [file.tpp]"
in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename)
usage_msg;

  let lexbuf = Lexing.from_channel !channel in
  let ast = Parser.program Scanner.token lexbuf in
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | _ -> let sast = Semant.check ast in
    match !action with
      Ast     -> ()
    | Sast    -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate
sast))
    | Compile -> let m = Codegen.translate sast in
      Llvm_analysis.assert_valid_module m;
      print_string (Llvm.string_of_llmodule m)
```

Testall

```
#!/bin/sh

# Regression testing script for textPlusPlus
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the textplusplus compiler.  Usually "./textplusplus.native"
# Try "_build/textplusplus.native" if ocamlbuild was unable to create a
symbolic link.
#TEXTPLUSPLUS="./textplusplus.native"
TEXTPLUSPLUS="_build/textplusplus.native"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
    echo "Usage: testall.sh [options] [.tpp files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

SignalError() {
```

```
    if [ $error -eq 0 ] ; then
      echo "FAILED"
      error=1
    fi
    echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to
difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
      SignalError "$1 differs"
      echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
      SignalError "$1 failed on $*"
      return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
    echo $* 1>&2
    eval $* && {
      SignalError "failed: $* did not report an error"
      return 1
    }
    return 0
}

Check() {
    error=0
```

```
    basename=`echo $1 | sed 's/.*\\///
                             s/.tpp//'`
    reffile=`echo $1 | sed 's/.tpp$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s
${basename}.exe ${basename}.out" &&
    Run "$TEXTPLUSPLUS" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s"
&&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "hello.o " "-lhpdf" &&
    Run "./${basename}.exe" &&
    Run "pdf2txt" "-o" "${basename}.out" "text.pdf" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
      if [ $keep -eq 0 ] ; then
          rm -f $generatedfiles
      fi
      echo "OK"
      echo "###### SUCCESS" 1>&2
    else
      echo "###### FAILED" 1>&2
      globalerror=$error
    fi
}

CheckFail() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.tpp//'`
    reffile=`echo $1 | sed 's/.tpp$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
```

```
    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$TEXTPLUSPLUS" "<" $1 "2>" "${basename}.err" ">>" $globallog
&&
    Compare ${basename}.err ${reffile}.err ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
      if [ $keep -eq 0 ] ; then
          rm -f $generatedfiles
      fi
      echo "OK"
      echo "###### SUCCESS" 1>&2
    else
      echo "###### FAILED" 1>&2
      globalerror=$error
    fi
}

while getopts kdpsh c; do
    case $c in
      k) # Keep intermediate files
          keep=1
          ;;
      h) # Help
          Usage
          ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
```

```
  echo "Check your LLVM installation and/or modify the LLI variable in
testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ ! -f hello.o ]
then
    echo "Could not find hello.o"
    echo "Try \"make hello.o\""
    exit 1
fi

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test-*.tpp tests/fail-*.tpp"
fi

for file in $files
do
    case $file in
      *test-*)
          Check $file 2>> $globallog
          ;;
      *fail-*)
          CheckFail $file 2>> $globallog
          ;;
      *)
          echo "unknown file type $file"
          globalerror=1
          ;;
    esac
done

exit $globalerror
```

Codegen

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
  let context    = L.global_context () in

  (* Create the LLVM compilation module into which
     we will generate code *)
  let the_module = L.create_module context "textPlusPlus" in

  (* Get types from the context *)
  let i32_t      = L.i32_type    context
  and i1_t       = L.i1_type     context
  and float_t    = L.double_type context
  and str_t      = L.pointer_type (L.i8_type context)
  and void_t     = L.void_type   context in

  (* Return the LLVM type for a textPlusPlus type *)
  let ltype_of_typ = function
      A.Int   -> i32_t
    | A.Bool  -> i1_t
    | A.Float -> float_t
    | A.Void  -> void_t
```

```
  | A.String -> str_t
in

(* Create a map of global variables after creating each *)
let global_vars : L.llvalue StringMap.t =
  let global_var m (t, n) =
    let init = match t with
        A.Float -> L.const_float (ltype_of_typ t) 0.0
      | _ -> L.const_int (ltype_of_typ t) 0
    in StringMap.add n (L.define_global n init the_module) m in
  List.fold_left global_var StringMap.empty globals in


let addPage_t : L.lltype =
    L.function_type i32_t [| |] in
let addPage_func : L.llvalue =
    L.declare_function "addPage" addPage_t the_module in



let left_t : L.lltype =
  L.function_type i32_t [| |] in
let left_func : L.llvalue =
    L.declare_function "left" left_t the_module in

let right_t : L.lltype =
  L.function_type i32_t [| |] in
let right_func : L.llvalue =
    L.declare_function "right" right_t the_module in

let center_t : L.lltype =
  L.function_type i32_t [| |] in
let center_func : L.llvalue =
    L.declare_function "center" center_t the_module in

let write_t : L.lltype =
  L.function_type i32_t [| str_t |] in
let write_func : L.llvalue =
    L.declare_function "write" write_t the_module in

let textOut_t : L.lltype =
  L.function_type i32_t [| i32_t ; i32_t ; str_t|] in
```

```
let textOut_func : L.llvalue =
    L.declare_function "textOut" textOut_t the_module in

let moveTo_t : L.lltype =
  L.function_type i32_t [| i32_t; i32_t |] in
let moveTo_func : L.llvalue =
    L.declare_function "moveTo" moveTo_t the_module in



let bold_t : L.lltype =
    L.function_type i32_t [| |] in
let bold_func : L.llvalue =
    L.declare_function "bold" bold_t the_module in

  let italic_t : L.lltype =
    L.function_type i32_t [| |] in
let italic_func : L.llvalue =
    L.declare_function "italic" italic_t the_module in

let regular_t : L.lltype =
    L.function_type i32_t [| |] in
let regular_func : L.llvalue =
    L.declare_function "regular" regular_t the_module in

let changeColor_t : L.lltype =
  L.function_type i32_t [| float_t; float_t; float_t |] in
let changeColor_func : L.llvalue =
    L.declare_function "changeColor" changeColor_t the_module in

let changeFontSize_t : L.lltype =
    L.function_type i32_t [| str_t; i32_t |] in
let changeFontSize_func : L.llvalue =
    L.declare_function "changeFontSize" changeFontSize_t the_module in



let drawLine_t : L.lltype =
    L.function_type i32_t [| i32_t; i32_t; i32_t; i32_t |] in
let drawLine_func : L.llvalue =
    L.declare_function "drawLine" drawLine_t the_module in
```

```
let drawRectangle_t : L.lltype =
    L.function_type i32_t [| i32_t; i32_t; i32_t; i32_t |] in
let drawRectangle_func : L.llvalue =
    L.declare_function "drawRectangle" drawRectangle_t the_module in



let pageNumber_t : L.lltype =
  L.function_type i32_t [| i32_t; i32_t |] in
let pageNumber_func : L.llvalue =
    L.declare_function "pageNumber" pageNumber_t the_module in

let getTextWidth_t : L.lltype =
  L.function_type i32_t [| str_t |] in
let getTextWidth_func : L.llvalue =
    L.declare_function "getTextWidth" getTextWidth_t the_module in

let getPageHeight_t : L.lltype =
  L.function_type i32_t [|  |] in
let getPageHeight_func : L.llvalue =
    L.declare_function "getPageHeight" getPageHeight_t the_module in

let getPageWidth_t : L.lltype =
  L.function_type i32_t [|  |] in
let getPageWidth_func : L.llvalue =
    L.declare_function "getPageWidth" getPageWidth_t the_module in



let pageTitle_t : L.lltype =
    L.function_type i32_t [| str_t |] in
let pageTitle_func : L.llvalue =
    L.declare_function "pageTitle" pageTitle_t the_module in

let table_t : L.lltype =
  L.function_type i32_t [| i32_t; i32_t; i32_t; i32_t |] in
let table_func : L.llvalue =
    L.declare_function "table" table_t the_module in

let heading1_t : L.lltype =
  L.function_type i32_t [| |] in
let heading1_func : L.llvalue =
```

```
  L.declare_function "heading1" heading1_t the_module in
let heading2_t : L.lltype =
  L.function_type i32_t [| |] in
let heading2_func : L.llvalue =
  L.declare_function "heading2" heading2_t the_module in
let heading3_t : L.lltype =
  L.function_type i32_t [| |] in
let heading3_func : L.llvalue =
  L.declare_function "heading3" heading3_t the_module in
let heading4_t : L.lltype =
  L.function_type i32_t [| |] in
let heading4_func : L.llvalue =
  L.declare_function "heading4" heading4_t the_module in
let heading5_t : L.lltype =
  L.function_type i32_t [| |] in
let heading5_func : L.llvalue =
  L.declare_function "heading5" heading5_t the_module in
let heading6_t : L.lltype =
  L.function_type i32_t [| |] in
let heading6_func : L.llvalue =
  L.declare_function "heading6" heading6_t the_module in
let getCurrentY_t : L.lltype =
L.function_type i32_t [| |] in
let getCurrentY_func : L.llvalue =
  L.declare_function "getCurrentY" getCurrentY_t the_module in
let getCurrentX_t : L.lltype =
L.function_type i32_t [| |] in
let getCurrentX_func : L.llvalue =
  L.declare_function "getCurrentX" getCurrentX_t the_module in
let getCapHeight_t : L.lltype =
L.function_type i32_t [| |] in
let getCapHeight_func : L.llvalue =
  L.declare_function "getCapHeight" getCapHeight_t the_module in
let getLowHeight_t : L.lltype =
L.function_type i32_t [| |] in
let getLowHeight_func : L.llvalue =
  L.declare_function "getLowHeight" getLowHeight_t the_module in
let getTextBytes_t : L.lltype =
L.function_type i32_t [| str_t; i32_t; i32_t |] in
let getTextBytes_func : L.llvalue =
  L.declare_function "getTextBytes" getTextBytes_t the_module in
  (***)
```

```
  let setRMargin_t : L.lltype =
  L.function_type i32_t [| i32_t|] in
  let setRMargin_func : L.llvalue =
  L.declare_function "setRMargin" setRMargin_t the_module in
  let setLMargin_t : L.lltype =
  L.function_type i32_t [| i32_t|] in
  let setLMargin_func : L.llvalue =
  L.declare_function "setLMargin" setLMargin_t the_module in
  let setTopMargin_t : L.lltype =
  L.function_type i32_t [| i32_t |] in
  let setTopMargin_func : L.llvalue =
  L.declare_function "setTopMargin" setTopMargin_t the_module in
  let setBotMargin_t : L.lltype =
  L.function_type i32_t [| i32_t |] in
  let setBotMargin_func : L.llvalue =
  L.declare_function "setBotMargin" setBotMargin_t the_module in

  (* Define each function (arguments and return type) so we can
     call it even before we've created its body *)
  let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
    let function_decl m fdecl =
      let name = fdecl.sfname
      and formal_types =
      Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.sformals)
      in let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types
in
      StringMap.add name (L.define_function name ftype the_module, fdecl) m
in
    List.fold_left function_decl StringMap.empty functions in

  (* Fill in the body of the given function *)
  let build_function_body fdecl =
    let (the_function, _) = StringMap.find fdecl.sfname function_decls in
    let builder = L.builder_at_end context (L.entry_block the_function) in

    (* Construct the function's "locals": formal arguments and locally
       declared variables.  Allocate each on the stack, initialize their
       value, if appropriate, and remember their values in the "locals" map
*)
    let local_vars =
      let add_formal m (t, n) p =
        L.set_value_name n p;
```

```
      let local = L.build_alloca (ltype_of_typ t) n builder in
        ignore (L.build_store p local builder);
      StringMap.add n local m

      (* Allocate space for any locally declared variables and add the
       * resulting registers to our map *)
      and add_local m (t, n) =
      let local_var = L.build_alloca (ltype_of_typ t) n builder
      in StringMap.add n local_var m
      in

      let formals = List.fold_left2 add_formal StringMap.empty
fdecl.sformals
          (Array.to_list (L.params the_function)) in
      List.fold_left add_local formals fdecl.slocals
    in

    (* Return the value for a variable or formal argument.
       Check local names first, then global names *)
    let lookup n = try StringMap.find n local_vars
                   with Not_found -> StringMap.find n global_vars
    in

    (* Construct code for an expression; return its value *)
    let rec expr builder ((_, e) : sexpr) = match e with
            SLiteral i  -> L.const_int i32_t i
      | SStrLiteral i -> L.build_global_stringptr i "string" builder
      | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
      | SFliteral l -> L.const_float_of_string float_t l
      | SNoexpr     -> L.const_int i32_t 0
      | SId s       -> L.build_load (lookup s) s builder
      | SAssign (s, e) -> let e' = expr builder e in
                          ignore(L.build_store e' (lookup s) builder); e'
      | SBinop ((A.Float,_ ) as e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
        (match op with
          A.Add     -> L.build_fadd
        | A.Sub     -> L.build_fsub
        | A.Mult    -> L.build_fmul
        | A.Div     -> L.build_fdiv
        | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
```

```
        | A.Neq     -> L.build_fcmp L.Fcmp.One
        | A.Less    -> L.build_fcmp L.Fcmp.Olt
        | A.Leq     -> L.build_fcmp L.Fcmp.Ole
        | A.Greater -> L.build_fcmp L.Fcmp.Ogt
        | A.Geq     -> L.build_fcmp L.Fcmp.Oge
        | A.And | A.Or ->
            raise (Failure "internal error: semant should have rejected
and/or on float")
        ) e1' e2' "tmp" builder
      | SBinop (e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
        (match op with
          A.Add     -> L.build_add
        | A.Sub     -> L.build_sub
        | A.Mult    -> L.build_mul
          | A.Div     -> L.build_sdiv
        | A.And     -> L.build_and
        | A.Or      -> L.build_or
        | A.Equal   -> L.build_icmp L.Icmp.Eq
        | A.Neq     -> L.build_icmp L.Icmp.Ne
        | A.Less    -> L.build_icmp L.Icmp.Slt
        | A.Leq     -> L.build_icmp L.Icmp.Sle
        | A.Greater -> L.build_icmp L.Icmp.Sgt
        | A.Geq     -> L.build_icmp L.Icmp.Sge
        ) e1' e2' "tmp" builder
      | SUnop(op, ((t, _) as e)) ->
          let e' = expr builder e in
        (match op with
          A.Neg when t = A.Float -> L.build_fneg
        | A.Neg                  -> L.build_neg
          | A.Not                  -> L.build_not) e' "tmp" builder


    | SCall ("addPage", []) ->
      L.build_call addPage_func [| |] "addPage" builder

    | SCall ("left", []) ->
      L.build_call left_func [| |] "left" builder
    | SCall ("right", []) ->
      L.build_call right_func [| |] "right" builder
    | SCall ("center", []) ->
```

```
      L.build_call center_func [| |] "center" builder

    | SCall ("write", [e]) ->
      L.build_call write_func [| (expr builder e) |] "write" builder
    | SCall ("textOut", [e; y; z]) ->
      L.build_call textOut_func [| (expr builder e); (expr builder y);
(expr builder z) |] "textOut" builder
    | SCall ("moveTo", [e; y]) ->
      L.build_call moveTo_func [| (expr builder e); (expr builder y)|]
"moveTo" builder

    | SCall ("bold", []) ->
      L.build_call bold_func [|  |] "bold" builder
    | SCall ("italic", []) ->
      L.build_call italic_func [|  |] "italic" builder
    | SCall ("regular", []) ->
      L.build_call regular_func [|  |] "regular" builder
    | SCall ("changeColor", [e; y; z]) ->
      L.build_call changeColor_func [| (expr builder e); (expr builder y);
(expr builder z) |] "changeColor" builder
    | SCall ("changeFontSize", [e ; y]) ->
      L.build_call changeFontSize_func [| (expr builder e); (expr builder
y)  |] "changeFontSize" builder

    | SCall ("drawLine", [e; y; z; a]) ->
      L.build_call drawLine_func [| (expr builder e); (expr builder y);
(expr builder z); (expr builder a)|] "drawLine" builder
    | SCall ("drawRectangle", [e; y; z; a]) ->
      L.build_call drawRectangle_func [| (expr builder e); (expr builder
y); (expr builder z); (expr builder a) |] "drawRectangle" builder

    | SCall ("pageNumber", [e; y]) ->
      L.build_call pageNumber_func [|(expr builder e); (expr builder y) |]
"pageNumber" builder
    | SCall ("getTextWidth", [e]) ->
      L.build_call getTextWidth_func [| (expr builder e) |] "getTextWidth"
builder
    | SCall ("getPageHeight", []) ->
      L.build_call getPageHeight_func [| |] "getPageHeight" builder
    | SCall ("getPageWidth", []) ->
      L.build_call getPageWidth_func [| |] "getPageWidth" builder
```

```
    | SCall ("pageTitle", [e]) ->
      L.build_call pageTitle_func [| (expr builder e) |] "pageTitle"
builder
    | SCall ("table", [e; y; z; a]) ->
      L.build_call table_func [| (expr builder e); (expr builder y); (expr
builder z); (expr builder a) |] "table" builder

    | SCall ("heading1", []) ->
      L.build_call heading1_func [| |] "heading1" builder
    | SCall ("heading2", []) ->
      L.build_call heading2_func [| |] "heading2" builder
    | SCall ("heading3", []) ->
      L.build_call heading3_func [| |] "heading3" builder
    | SCall ("heading4", []) ->
      L.build_call heading4_func [| |] "heading4" builder
    | SCall ("heading5", []) ->
      L.build_call heading5_func [| |] "heading5" builder
    | SCall ("heading6", []) ->
      L.build_call heading6_func [| |] "heading6" builder
    | SCall ("getCurrentY", []) ->
      L.build_call getCurrentY_func [| |] "getCurrentY" builder
    | SCall ("getCurrentX", []) ->
      L.build_call getCurrentX_func [| |] "getCurrentX" builder
    | SCall ("getCapHeight", []) ->
      L.build_call getCapHeight_func [| |] "getCapHeight" builder
    | SCall ("getLowHeight", []) ->
      L.build_call getLowHeight_func [| |] "getLowHeight" builder
    | SCall ("getTextBytes", [x;y;z]) ->
      L.build_call getTextBytes_func [| (expr builder x); (expr builder y);
(expr builder z) |] "getTextBytes" builder
    | SCall ("setRMargin", [e]) ->
      L.build_call setRMargin_func [| (expr builder e) |] "setRMargin"
builder
    | SCall ("setLMargin", [e]) ->
      L.build_call setLMargin_func [| (expr builder e) |] "setLMargin"
builder
    | SCall ("setTopMargin", [e]) ->
      L.build_call setTopMargin_func [| (expr builder e) |] "setTopMargin"
builder
    | SCall ("setBotMargin", [e]) ->
      L.build_call setBotMargin_func [| (expr builder e) |] "setBotMargin"
builder
```

```
      | SCall (f, args) ->
        let (fdef, fdecl) = StringMap.find f function_decls in
       let llargs = List.rev (List.map (expr builder) (List.rev args)) in
       let result = (match fdecl.styp with
                        A.Void -> ""
                      | _ -> f ^ "_result") in
        L.build_call fdef (Array.of_list llargs) result builder
    in

    (* LLVM insists each basic block end with exactly one "terminator"
       instruction that transfers control.  This function runs "instr
builder"
       if the current block does not already have a terminator.  Used,
       e.g., to handle the "fall off the end of the function" case. *)
    let add_terminal builder instr =
      match L.block_terminator (L.insertion_block builder) with
      Some _ -> ()
      | None -> ignore (instr builder) in

    (* Build the code for the given statement; return the builder for
       the statement's successor (i.e., the next instruction will be built
       after the one generated by this call) *)

    let rec stmt builder = function
      SBlock sl -> List.fold_left stmt builder sl
      | SExpr e -> ignore(expr builder e); builder
      | SReturn e -> ignore(match fdecl.styp with
                              (* Special "return nothing" instr *)
                              A.Void -> L.build_ret_void builder
                              (* Build return statement *)
                            | _ -> L.build_ret (expr builder e) builder );
                     builder
      | SIf (predicate, then_stmt, else_stmt) ->
        let bool_val = expr builder predicate in
       let merge_bb = L.append_block context "merge" the_function in
        let build_br_merge = L.build_br merge_bb in (* partial function *)

       let then_bb = L.append_block context "then" the_function in
       add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
```

```
        build_br_merge;

      let else_bb = L.append_block context "else" the_function in
      add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
        build_br_merge;

      ignore(L.build_cond_br bool_val then_bb else_bb builder);
      L.builder_at_end context merge_bb

    | SWhile (predicate, body) ->
      let pred_bb = L.append_block context "while" the_function in
      ignore(L.build_br pred_bb builder);

      let body_bb = L.append_block context "while_body" the_function in
      add_terminal (stmt (L.builder_at_end context body_bb) body)
        (L.build_br pred_bb);

      let pred_builder = L.builder_at_end context pred_bb in
      let bool_val = expr pred_builder predicate in

      let merge_bb = L.append_block context "merge" the_function in
      ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
      L.builder_at_end context merge_bb

    (* Implement for loops as while loops *)
    | SFor (e1, e2, e3, body) -> stmt builder
        ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] )
  in

  (* Build the code for each statement in the function *)
  let builder = stmt builder (SBlock fdecl.sbody) in

  (* Add a return if the last block falls off the end *)
  add_terminal builder (match fdecl.styp with
      A.Void -> L.build_ret_void
    | A.Float -> L.build_ret (L.const_float float_t 0.0)
    | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
in

List.iter build_function_body functions;
the_module
```

Hello.c

```
/*
 * << Haru Free PDF Library 2.0.0 >> -- attach.c
 * Copyright (c) 1999-2006 Takeshi Kanno <takeshi_kanno@est.hi-ho.ne.jp>
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <setjmp.h>
#include "hpdf.h"


// Document Handling
char fname[256];
HPDF_Doc pdf;

// Page Handling
HPDF_Page firstPage;
HPDF_Page currentPage;

HPDF_REAL pageHeight;
HPDF_REAL pageWidth;

int pnumber;

//Text Handling
HPDF_REAL currentX;
HPDF_REAL currentY;

float tw;

// Font Handling
HPDF_Font defaultFont;
HPDF_Font currentFont;
HPDF_REAL defaultSize;
HPDF_REAL currentSize;

HPDF_Font helvetica;
HPDF_Font helveticaItalic;
HPDF_Font helveticaBold;
```

```
HPDF_Font times;
HPDF_Font timesItalic;
HPDF_Font timesBold;

HPDF_Font courier;
HPDF_Font courierItalic;
HPDF_Font courierBold;

float textWidth;
int alignment;
int lmarg;
int rmarg;
int bmarg;
int tmarg;


extern void start();

jmp_buf env;

// Error Handling

void
error_handler (HPDF_STATUS   error_no,
               HPDF_STATUS   detail_no,
               void         *user_data
)
{    printf ("ERROR: error_no=%04X, detail_no=%u\n", (HPDF_UINT)error_no,
                (HPDF_UINT)detail_no);
    longjmp(env, 1);
}


// PAGE HANDLING FUNCTIONS

int  addPage(){

      /* creates and adds new page to PDF */
      HPDF_Page newPage;
      newPage = HPDF_AddPage(pdf);
```

```
      /* updates current page and page number */
      currentPage = newPage;
      pnumber = pnumber + 1;

      /* sets the font, size, and line width for page */
      HPDF_Page_SetFontAndSize(currentPage, currentFont, currentSize);
      HPDF_Page_SetLineWidth(currentPage, 1);

    /* initializes value for pageHeight and pageWidth */
    pageHeight = HPDF_Page_GetHeight(currentPage);
    pageWidth = HPDF_Page_GetWidth(currentPage);

      /* sets  X and Y cooridintes to top left of page with margins*/
      currentX = 25;
      currentY = pageHeight - 25;

      return 0;
}

// TEXT HANDLING FUNCTIONS

int left(){

      alignment = 0;
      return 0;
}

int right(){
      alignment = 1;
      return 0;
}

int center(){
      alignment = 2;
      return 0;
}

int getCapHeight(){
    int f_height_point = HPDF_Font_GetCapHeight(currentFont);
    return (int)(f_height_point * currentSize / 1000.0);
}
```

```
int getLowHeight(){
    int f_height_point = HPDF_Font_GetXHeight(currentFont);
    return (int)(f_height_point * currentSize / 1000.0);
}

//gets how much bytes can fit in one line given some margins on a line
int getTextBytes(char * text, int lmargin, int rmargin){
    ///assuming left and right margins are the same
    int page_limit = pageWidth - lmargin - rmargin;
    return (int)( HPDF_Page_MeasureText(currentPage, text, page_limit,
HPDF_TRUE, NULL));
}

int setLMargin(int marg){
    lmarg = marg;
    return 0;
}

int setRMargin(int marg){
    rmarg = marg;
    return 0;
}

int setTopMargin(int marg){
    tmarg = marg;
    return 0;
}

int setBotMargin(int marg){
    bmarg = marg;
    return 0;
}



int write(char * text){
    // align: 0 means left, 1 means right, 2 means center

    int page_limit = pageWidth - lmarg - rmarg;

    int f_height_point = HPDF_Font_GetCapHeight(currentFont);
    int f_real_h = f_height_point * currentSize / 1000.0;
```

```
    int last_line = 0;

    int pos = 0;

    int textWidth = 0;
    int move_right = 0;
    currentX = lmarg;
    currentY = pageHeight - tmarg;

    HPDF_Page_SetFontAndSize(currentPage, currentFont, currentSize);
    HPDF_Page_BeginText(currentPage);

    for(;;){

      int bytes = HPDF_Page_MeasureText(currentPage, text, page_limit,
HPDF_TRUE, NULL);
        char *start = &text[pos];
        char *end = &text[pos + bytes];
        size_t length = end - start;

        char *curr_string = (char *) malloc(length + 1);
        memcpy(curr_string, start, length);
        curr_string[length] = '\0';

        switch(alignment) {
            case 0: ; //left alighnment
                  HPDF_Page_TextOut (currentPage, currentX, currentY,
curr_string);
                  break;

            case 1: ; //right alighnment
                  textWidth = HPDF_Page_TextWidth(currentPage,
curr_string);
                move_right = (pageWidth - rmarg) - (textWidth + lmarg);
                HPDF_Page_TextOut (currentPage, currentX + move_right,
currentY, curr_string);
                break;

            case 2: ; //means center
                  textWidth = HPDF_Page_TextWidth(currentPage,
curr_string);
```

```
                move_right = ((pageWidth - rmarg) - (textWidth + lmarg)) /
2;
                HPDF_Page_TextOut (currentPage, currentX + move_right,
currentY, curr_string);
                break;
        }

        free(curr_string);
        currentY = currentY - 2 * f_real_h;
        text = text + bytes;

        //for the case the person writes stuff thats longer
        //than the page can fit
        if (currentY <= bmarg){ //default bottom marg is 25 is the bottom
margin
            HPDF_Page_EndText (currentPage);

            HPDF_Page newPage;
            newPage = HPDF_AddPage(pdf);

            currentPage = newPage;

            HPDF_Page_BeginText (currentPage);
            HPDF_Page_SetFontAndSize (currentPage, currentFont,
currentSize);

            currentX = lmarg; //set it to the margin
            currentY = pageHeight - tmarg;
        }

        if (last_line == 1){
            break;
        }

        if (strlen(text) <= bytes){
            last_line = 1;
        }

    }

    HPDF_Page_EndText (currentPage);
```

```
    return 0;
}

int textOut(int x, int y, char * text){
      // align: 0 means left, 1 means right, 2 means center

    currentX = x;
    currentY = y;

    int page_limit = pageWidth - lmarg - rmarg;

    int f_height_point = HPDF_Font_GetCapHeight(currentFont);
    int f_real_h = f_height_point * currentSize / 1000.0;

    int last_line = 0;

    int pos = 0;

    int textWidth = 0;
    int move_right = 0;

    HPDF_Page_SetFontAndSize(currentPage, currentFont, currentSize);
    HPDF_Page_BeginText(currentPage);

    for(;;){

      int bytes = HPDF_Page_MeasureText(currentPage, text, page_limit,
HPDF_TRUE, NULL);
        char *start = &text[pos];
        char *end = &text[pos + bytes];
        size_t length = end - start;

        char *curr_string = (char *) malloc(length + 1);
        memcpy(curr_string, start, length);
        curr_string[length] = '\0';

        switch(alignment) {
            case 0: ; //left alighnment
                  HPDF_Page_TextOut (currentPage, currentX, currentY,
curr_string);
                  break;
```

```
            case 1: ; //right alighnment
                  textWidth = HPDF_Page_TextWidth(currentPage,
curr_string);
                move_right = (pageWidth - rmarg) - (textWidth + lmarg);
                HPDF_Page_TextOut (currentPage, currentX + move_right,
currentY, curr_string);
                break;

            case 2: ; //means center
                  textWidth = HPDF_Page_TextWidth(currentPage,
curr_string);
                move_right = ((pageWidth - rmarg) - (textWidth + lmarg)) /
2;
                HPDF_Page_TextOut (currentPage, currentX + move_right,
currentY, curr_string);
                break;
        }

        free(curr_string);
        currentY = currentY - 2 * f_real_h;
        text = text + bytes;

        //for the case the person writes stuff thats longer
        //than the page can fit
        if (currentY <= bmarg){ //here 25 is the bottom margin
            HPDF_Page_EndText (currentPage);

            HPDF_Page newPage;
            newPage = HPDF_AddPage(pdf);

            currentPage = newPage;

            HPDF_Page_BeginText (currentPage);
            HPDF_Page_SetFontAndSize (currentPage, currentFont,
currentSize);

            currentX = lmarg; //set it to the margin
            currentY = pageHeight - tmarg;
        }

        if (last_line == 1){
            break;
```

```
        }

        if (strlen(text) <= bytes){
            last_line = 1;
        }

    }

    HPDF_Page_EndText (currentPage);

    return 0;

}

int moveTo(int x , int y){
      //take page, x, and y position
      HPDF_REAL x_pos = x; //Harcoded for now 50
      HPDF_REAL y_pos = y;
      HPDF_Page_MoveTo(currentPage, x_pos, y_pos);

      return 0;
}


// FONT HANDLING FUNCTIONS

int bold(){

      /* changes current font to Helvetica Bold */
      if ((currentFont == helvetica) || (currentFont == helveticaItalic)){
            currentFont = helveticaBold;
            HPDF_Page_SetFontAndSize(currentPage, currentFont,
currentSize);
      }

      /* changes current font to Times Bold */
      if ((currentFont == times) || (currentFont == timesItalic)){
            currentFont = timesBold;
            HPDF_Page_SetFontAndSize(currentPage, currentFont,
currentSize);
      }
```

```
	/* changes current font to Courier Bold */
	if ((currentFont == courier) || (currentFont == courierItalic)){
		currentFont = courierBold;
		HPDF_Page_SetFontAndSize(currentPage, currentFont,
currentSize);
	}

	return 0;
}

int italic(){

	/* changes current font to Helvetica Italic */
	if ((currentFont == helvetica) || (currentFont == helveticaBold)){

		currentFont = helveticaItalic;
		HPDF_Page_SetFontAndSize(currentPage, currentFont,
currentSize);
	}

	/* changes current font to Times Italic */
	if ((currentFont == times) || (currentFont == timesBold)){
		currentFont = timesItalic;
		HPDF_Page_SetFontAndSize(currentPage, currentFont,
currentSize);
	}

	/* changes current font to Courier Italic */
	if ((currentFont == courier) || (currentFont == courierBold)){
		currentFont = courierItalic;
		HPDF_Page_SetFontAndSize(currentPage, currentFont,
currentSize);
	}

	return 0;
}

int regular(){

	/* changes current font to Helvetica */
	if ((currentFont == helveticaItalic) || (currentFont ==
helveticaBold)){
```

```
            currentFont = helvetica;
            HPDF_Page_SetFontAndSize(currentPage, currentFont,
currentSize);
      }

      /* changes current font to Times */
      if ((currentFont == timesItalic) || (currentFont == timesBold)){
            currentFont = times;
            HPDF_Page_SetFontAndSize(currentPage, currentFont,
currentSize);
      }

      /* changes current font to Courier */
      if ((currentFont == courierItalic) || (currentFont == courierBold)){
            currentFont = courier;
            HPDF_Page_SetFontAndSize(currentPage, currentFont,
currentSize);
      }

      return 0;
}

int changeColor( float red, float green, float blue){

      /* sets the RGB values for the font */
      HPDF_Page_SetRGBFill(currentPage, red, green, blue);
      return 0;
}

int changeFontSize (char * font, int newSize){

      /* updates current font and size */
      currentFont = HPDF_GetFont(pdf, font, NULL);
      currentSize = newSize;

      /* set new font and size to current page */
      HPDF_Page_SetFontAndSize(currentPage, currentFont, currentSize);
      return 0;
}


// SHAPE + LINE HANDLING FUNCTIONS
```

```
int drawLine( int startX, int startY, int endX, int endY){

      // Draws a line from (startX, startY) to (endX, endY)
      HPDF_Page_MoveTo(currentPage, startX, startY);
      HPDF_Page_LineTo(currentPage, endX, endY);
      HPDF_Page_Stroke(currentPage);

      return 0;
}

int drawRectangle( int lowerLeftX, int lowerLeftY, int rectangleWidth, int
rectangleHeight){

      /* draws a rectangle on dimentions (rectangleWidth x rectangleHeight)
       with bottom left corner of rectangle at (lowerLeftX, lowerLeftY) */

      HPDF_Page_Rectangle(currentPage, lowerLeftX, lowerLeftY,
rectangleWidth, rectangleHeight);
      HPDF_Page_Stroke(currentPage);

      return 0;
}



int pageNumber(int x, int y){

      char strPageNumber[100];
      sprintf(strPageNumber, "%d", pnumber);

    HPDF_Page_BeginText(currentPage);
    HPDF_Page_TextOut(currentPage, x, y, strPageNumber);
    HPDF_Page_EndText(currentPage);

      return 0;
}


// Getter Functions
int getTextWidth(char *text){
      return HPDF_Page_TextWidth(currentPage, text);
```

```
}

int getPageHeight(){

      return (int)pageHeight;
}

int getPageWidth(){

      return (int)pageWidth;
}

int getCurrentX(){
    return (int)currentX;
}

int getCurrentY(){
    return (int)currentY;
}

// FUNCTIONS FOR STANDARD LIBRARY


// Writes a centered single line on the current page
int pageTitle(char* text){

      HPDF_Page_SetFontAndSize (currentPage, currentFont, currentSize);
      tw = HPDF_Page_TextWidth (currentPage, text);
      HPDF_Page_BeginText (currentPage);
      HPDF_Page_TextOut (currentPage, (HPDF_Page_GetWidth(currentPage) -
tw) / 2, HPDF_Page_GetHeight (currentPage) - currentSize, text);
      HPDF_Page_EndText (currentPage);

      return 0;

}

// Headings - changes the font size accordingly (based on HTML standards)
int heading1(){
      currentSize = 32;
      HPDF_Page_SetFontAndSize(currentPage, currentFont, currentSize);
      return 0;
```

```
}

int heading2(){
	currentSize = 24;
	HPDF_Page_SetFontAndSize(currentPage, currentFont, currentSize);
	return 0;
}

int heading3(){
	currentSize = 18;
	HPDF_Page_SetFontAndSize(currentPage, currentFont, currentSize);
	return 0;
}

int heading4(){
	currentSize = 16;
	HPDF_Page_SetFontAndSize(currentPage, currentFont, currentSize);
	return 0;
}

int heading5(){
	currentSize = 14;
	HPDF_Page_SetFontAndSize(currentPage, currentFont, currentSize);
	return 0;
}

int heading6(){
	currentSize = 12;
	HPDF_Page_SetFontAndSize(currentPage, currentFont, currentSize);
	return 0;
}

// Draws a horizantal in the current position
int horizantalLine(){

	HPDF_Page_MoveTo(currentPage, currentX, currentY);
	HPDF_Page_LineTo(currentPage, pageWidth, currentY);
	HPDF_Page_Stroke(currentPage);

	return 0;
}
```

```
int table(int row, int column, int tableWidth, int tableHeight){

	int horizontalMax;
	int verticalMax;
	int rowHeight;
	int columnWidth;

	int r;
	int c;

	int i;
	int j;

	r = row +1;
	c = column + 1;


	if (tableWidth > (pageWidth - currentX)){
		horizontalMax = pageWidth - currentX;
	}
	else {
		horizontalMax = tableWidth;
	}

	if (tableHeight >  currentY){
		verticalMax = currentY;
	}
	else {
		verticalMax = tableHeight;
	}

	rowHeight = verticalMax / row;
	columnWidth = horizontalMax / column;


	// Draw horizantal lines
	 for ( i = 0 ; i < r ; i++ ){

		HPDF_Page_MoveTo(currentPage, currentX, currentY - (rowHeight *
i));
		HPDF_Page_LineTo(currentPage, horizontalMax, currentY -
(rowHeight * i));
```

```
            HPDF_Page_Stroke(currentPage);

       }

      // Draw vertical lines
       for ( j = 1 ; j < c ; j++ ){

            HPDF_Page_MoveTo(currentPage, currentX + (columnWidth * j),
currentY);
            HPDF_Page_LineTo(currentPage, currentX + (columnWidth * j),
currentY - verticalMax);
            HPDF_Page_Stroke(currentPage);

       }

       return 0;

}







int main(int argc){

    /* starts program
       * creates a PDF document */

    pdf = HPDF_New(error_handler, NULL);

    if (!pdf) {
        printf ("error: cannot create PdfDoc object\n");
        return 1;
    }


    /* initializing fonts */
      helvetica = HPDF_GetFont(pdf, "Helvetica", NULL);
      helveticaItalic = HPDF_GetFont(pdf, "Helvetica-Oblique", NULL);
      helveticaBold = HPDF_GetFont(pdf, "Helvetica-Bold", NULL);
```

```
  times = HPDF_GetFont(pdf, "Times-Roman", NULL);
  timesItalic = HPDF_GetFont(pdf, "Times-Italic", NULL);
  timesBold = HPDF_GetFont(pdf, "Times-Bold", NULL);

  courier = HPDF_GetFont(pdf, "Courier", NULL);
  courierItalic = HPDF_GetFont(pdf, "Courier-Oblique", NULL);
  courierBold = HPDF_GetFont(pdf, "Courier-Bold", NULL);


  /* creates and adds new page to PDF */
firstPage = HPDF_AddPage(pdf);
currentPage = firstPage;
pnumber = 1;


/* sets default color, size, and font */
defaultFont = HPDF_GetFont (pdf, "Helvetica", NULL);
currentFont = defaultFont;
defaultSize = 12;
currentSize = defaultSize;
HPDF_Page_SetFontAndSize(firstPage, defaultFont, defaultSize);

  /* sets the default alignment to left*/
  alignment = 0;

/* initializes value for pageHeight and pageWidth */
pageHeight = HPDF_Page_GetHeight(firstPage);
pageWidth = HPDF_Page_GetWidth(firstPage);

/* sets line  stroke width */
HPDF_Page_SetLineWidth(firstPage, 1);

/* set default margins to 25 */
rmarg = 25;
lmarg = 25;
bmarg = 25;
tmarg = 25;


  /* sets  X and Y cooridintes to top left of page */
currentX = lmarg;
```

```
    currentY = pageHeight - tmarg;

    /* program starts */
    start();


    /* ends program
     * saves file as 'text.pdf' */
    HPDF_SaveToFile (pdf, "text.pdf");
    HPDF_Free (pdf);

    return 0;
}

#ifdef BUILD_TEST
int main()
{
    hello(0);
    return 0;
}
#endif
```

fail-add-page.tpp

```
addPage();
```

fail-bold.tpp

```
def void start(){
   bold("bold sentence");
}
```

fail-italic.tpp

```
def void start(){
   italic("italic sentence");
}
```

fail-regular.tpp

```
def void start(){
```

```
   regular("regular sentence");
}
```

fail-start.tpp

```
def void start(1){
}
```

fail-text-out.tpp

```
def void start(){
   textOut(10,10,10);
   textOut("Hello World");
}
```

fail-write.tpp

```
def void start(){
   write(123);
}
```

test-add-page.tpp

```
def void start(){
   write("Hello World!");
   addPage();
   write("Hello Again World!!");
}
```

test-bold.tpp

```
def void start(){
   bold();
   write("Bolding!");
}
```

test-bot-margin.tpp

```
def void start(){
  setBotMargin(100);
  write("At school we were given an hour-long break for
lunch each day. Because my mother didn't work and our
apartment was so close by, I usually marched home with four or five other
girls in tow, all of us talking nonstop, ready to sprawl on the kitchen
```

floor to play jacks and watch All My Children while my mom handed out sandwiches. This, for me, began a habit that has sustained me for life, keeping a close and high-spirited council of girlfriends safe harbor of female wisdom. In my lunch group, we dissected whatever had gone on that morning at school, any beefs we had with teachers, any assignments that struck us as useless. Our opinions were largely formed by committee. We idolized the Jackson 5 and weren't sure how we felt about the Osmonds. Watergate had happened, but none of us understood it. It seemed like a lot of old guys talking into microphones in Washington, D.C., which to us was just a faraway city filled with a lot of white buildings and white men. My mom, meanwhile, was plenty happy to serve us. It gave her an easy window into our world. As my friends and I ate and gossiped, she often stood by quietly, engaged in some household chore, not hiding the fact that she was taking in every word. In my family, with four of us packed into less than nine hundred square feet of living space, we'd never had any privacy anyway. It mattered only sometimes. Craig, who was suddenly interested in girls, had started taking his phone calls behind closed doors in the bathroom, the phone's curlicue cord stretched taut across the hallway from its wall-mounted base in the kitchen.As Chicago schools went, Bryn Mawr fell somewhere between a bad school and a good school. Racial and economic sorting in the South Shore neighborhood continued through the 1970s, meaning that the student population only grew blacker and poorer with each year. There was, for a time, a citywide integration movement to bus kids to new schools, but Bryn Mawr parents had successfully fought it off, arguing that the money was better spent improving the school itself. As a kid, I had no perspective on whether the facilities were run-down or whether it mattered that there were hardly any white kids left. The school ran from kindergarten all the way through eighth grade, which meant that by the time I had reached the upper grades, I knew every light switch, every chalkboard and cracked patch of hallway. I knew nearly every teacher and most of the kids. For me, Bryn Mawr was practically an extension of home. As I was entering seventh grade, the Chicago Defender, a weekly newspaper that was popular with African American readers, ran a vitriolic opinion piece that claimed Bryn Mawr had gone, in the span of a few years, from being one of the city's best public schools to a run- down slum governed by a ghetto mentality. Our school principal, Dr. Lavizzo, immediately hit back with a letter to the editor, defending his community of parents and students and deeming the newspaper piece an outrageous lie, which seems designed to incite only feelings of failure and flight.Dr. Lavizzo was a round, cheery man who had an Afro that puffed out on either side of his bald spot and who spent most of his time in an office near the building's front door. It's clear from his letter that he understood precisely what he was up against.

```
Failure is a feeling long before it becomes an actual result. It's
vulnerability that breeds with self- doubt and then is escalated, often
deliberately, by fear. Those feelings of failure he mentioned were
everywhere already in my neighborhood, in the form of parents who couldn't
get ahead financially, of kids who were starting to suspect that their
lives would be no different, of families who watched their better-off
neighbors leave for the suburbs or transfer their children to Catholic
schools. There were predatory real estate agents roaming South Shore all
the while, whispering to homeowners that they should sell before it was too
late, that they'd help them get out while you still can. The inference
being that failure was coming, that it was inevitable, that it had already
half arrived. You could get caught up in the ruin or you could escape it.
They used the word everyone was most afraid of ghetto dropping it like a
lit match. My mother bought into none of this. She'd lived in South Shore
for ten years already and would end up staying another forty. She didn't
buy into fear mongering and at the same time seemed equally inoculated
against any sort of pie-in-the-sky idealism. She was a
straight-down-the-line realist, controlling what she could. At Bryn Mawr,
she became one of the most active members of the PTA, helping raise funds
for new classroom equipment, throwing appreciation dinners for the
teachers, and lobbying for the creation of a special multigrade classroom
that catered to higher-performing students. This last effort was the
brainchild of Dr. Lavizzo, who'd gone to night school to get his PhD in
education and had studied a new trend in grouping students by ability
rather than by again essence, putting the brighter kids together so they
could learn at a faster pace."); }
```

test-center-wrap.tpp

```
def void start(){
   center();
   write("At school we were given an hour-long break for lunch each day.
Because my mother didn't work and our apartment was so close by, I usually
marched home with four or five other girls in tow, all of us talking
nonstop, ready to sprawl on the kitchen floor to play jacks and watch All
My Children while my mom handed out sandwiches."); }
```

test-draw-line.tpp

```
def void start(){
   drawLine(25, 300, 70, 400);
}
```

test-for.tpp

```
def void start(){
   int x;
   for(x = 0; x < 4; x = x + 1){
      write("Do re me");
   }
}
```

test-get-bytes.tpp

```
def void start(){
   int x;
   x = getTextBytes("Hello World", 25, 25);
   /* Hello World is < page_limit so should return bytes in     Hello World
*/

   if(x == 11){
      write("Correct Bytes to fit on Page"); }
   else{
      write("Incorrect Bytes to fit on Page");
   }
}
```

test-get-low-height.tpp

```
def void start(){
   int x;
   changeFontSize("Helvetica", 12);
   x = getCapHeight();
   if(x == 8){
     write("Correct Height"); }
   else{
      write("Incorrect Height");
   }
}
```

test-get-cap-height.tpp

```
def void start(){
   int x;
   changeFontSize("Helvetica", 12);
   x = getCapHeight();
   if(x == 8){
      write("Correct Height");
   }else{
      write("Incorrect Height");
   }
}
```

test-heading.tpp

```
def void start(){
   heading2();
   write("Header");
}
```

test-hello.tpp

```
def void start(){
   write("Hello World!");
}
```

test-italic.tpp

```
def void start(){
   italic();
   write("Italiscing!");
}
```

test-left-margin.tpp

```
def void start(){
   setLMargin(100);
   write("At school we were given an hour-long break for lunch each day.
Because my mother didn't work and our apartment was so close by, I usually
marched home with four or five other girls in tow, all of us talking
nonstop, ready to sprawl on the kitchen floor to play jacks and watch All
My Children while my mom handed out sandwiches. This, for me, began a habit
```

that has sustained me for life, keeping a close and high-spirited council of girlfriends safe harbor of female wisdom. In my lunch group, we dissected whatever had gone on that morning at school, any beefs we had with teachers, any assignments that struck us as useless. Our opinions were largely formed by committee. We idolized the Jackson 5 and weren't sure how we felt about the Osmonds. Watergate had happened, but none of us understood it. It seemed like a lot of old guys talking into microphones in Washington, D.C., which to us was just a faraway city filled with a lot of white buildings and white men. My mom, meanwhile, was plenty happy to serve us. It gave her an easy window into our world. As my friends and I ate and gossiped, she often stood by quietly, engaged in some household chore, not hiding the fact that she was taking in every word. In my family, with four of us packed into less than nine hundred square feet of living space, we'd never had any privacy anyway. It mattered only sometimes. Craig, who was suddenly interested in girls, had started taking his phone calls behind closed doors in the bathroom, the phone's curlicue cord stretched taut across the hallway from its wall-mounted base in the kitchen.As Chicago schools went, Bryn Mawr fell somewhere between a bad school and a good school. Racial and economic sorting in the South Shore neighborhood continued through the 1970s, meaning that the student population only grew blacker and poorer with each year. There was, for a time, a citywide integration movement to bus kids to new schools, but Bryn Mawr parents had successfully fought it off, arguing that the money was better spent improving the school itself. As a kid, I had no perspective on whether the facilities were run-down or whether it mattered that there were hardly any white kids left. The school ran from kindergarten all the way through eighth grade, which meant that by the time I had reached the upper grades, I knew every light switch, every chalkboard and cracked patch of hallway. I knew nearly every teacher and most of the kids. For me, Bryn Mawr was practically an extension of home. As I was entering seventh grade, the Chicago Defender, a weekly newspaper that was popular with African American readers, ran a vitriolic opinion piece that claimed Bryn Mawr had gone, in the span of a few years, from being one of the city's best public schools to a run- down slum governed by a ghetto mentality. Our school principal, Dr. Lavizzo, immediately hit back with a letter to the editor, defending his community of parents and students and deeming the newspaper piece an outrageous lie, which seems designed to incite only feelings of failure and flight.Dr. Lavizzo was a round, cheery man who had an Afro that puffed out on either side of his bald spot and who spent most of his time in an office near the building's front door. It's clear from his letter that he understood precisely what he was up against. Failure is a feeling long before it becomes an actual result. It's vulnerability that breeds with

```
self- doubt and then is escalated, often deliberately, by fear. Those
feelings of failure he mentioned were everywhere already in my
neighborhood, in the form of parents who couldn't get ahead financially, of
kids who were starting to suspect that their lives would be no different,
of families who watched their better-off neighbors leave for the suburbs or
transfer their children to Catholic schools. There were predatory real
estate agents roaming South Shore all the while, whispering to homeowners
that they should sell before it was too late, that they'd help them get out
while you still can. The inference being that failure was coming, that it
was inevitable, that it had already half arrived. You could get caught up
in the ruin or you could escape it. They used the word everyone was most
afraid of ghetto dropping it like a lit match. My mother bought into none
of this. She'd lived in South Shore for ten years already and would end up
staying another forty. She didn't buy into fear mongering and at the same
time seemed equally inoculated against any sort of pie-in-the-sky idealism.
She was a straight-down-the-line realist, controlling what she could. At
Bryn Mawr, she became one of the most active members of the PTA, helping
raise funds for new classroom equipment, throwing appreciation dinners for
the teachers, and lobbying for the creation of a special multigrade
classroom that catered to higher-performing students. This last effort was
the brainchild of Dr. Lavizzo, who'd gone to night school to get his PhD in
education and had studied a new trend in grouping students by ability
rather than by again essence, putting the brighter kids together so they
could learn at a faster pace.");
}
```

test-page-number.tpp

```
def void start(){
   pageNumber(25, 300);
}
```

test-page-title.tpp

```
def void start(){
   pageTitle("My name is textPlusPlus");
}
```

test-rect.tpp

```
def void start(){
   drawRectangle(25, 300, 100, 100);
}
```

test-regular.tpp

```
def void start(){
   regular();
   write("Regular!");
}
```

test-right-margin.tpp

```
def void start(){
   setRMargin(100);
   write("At school we were given an hour-long break for lunch each day.
Because my mother didn't work and our apartment was so close by, I usually
marched home with four or five other girls in tow, all of us talking
nonstop, ready to sprawl on the kitchen floor to play jacks and watch All
My Children while my mom handed out sandwiches. This, for me, began a habit
that has sustained me for life, keeping a close and high-spirited council
of girlfriends safe harbor of female wisdom. In my lunch group, we
dissected whatever had gone on that morning at school, any beefs we had
with teachers, any assignments that struck us as useless. Our opinions were
largely formed by committee. We idolized the Jackson 5 and weren't sure how
we felt about the Osmonds. Watergate had happened, but none of us
understood it. It seemed like a lot of old guys talking into microphones in
Washington, D.C., which to us was just a faraway city filled with a lot of
white buildings and white men. My mom, meanwhile, was plenty happy to serve
us. It gave her an easy window into our world. As my friends and I ate and
gossiped, she often stood by quietly, engaged in some household chore, not
hiding the fact that she was taking in every word. In my family, with four
of us packed into less than nine hundred square feet of living space, we'd
never had any privacy anyway. It mattered only sometimes. Craig, who was
suddenly interested in girls, had started taking his phone calls behind
closed doors in the bathroom, the phone's curlicue cord stretched taut
across the hallway from its wall-mounted base in the kitchen.As Chicago
schools went, Bryn Mawr fell somewhere between a bad school and a good
school. Racial and economic sorting in the South Shore neighborhood
continued through the 1970s, meaning that the student population only grew
blacker and poorer with each year. There was, for a time, a citywide
integration movement to bus kids to new schools, but Bryn Mawr parents had
successfully fought it off, arguing that the money was better spent
improving the school itself. As a kid, I had no perspective on whether the
```

```
facilities were run-down or whether it mattered that there were hardly any
white kids left. The school ran from kindergarten all the way through
eighth grade, which meant that by the time I had reached the upper grades,
I knew every light switch, every chalkboard and cracked patch of hallway. I
knew nearly every teacher and most of the kids. For me, Bryn Mawr was
practically an extension of home. As I was entering seventh grade, the
Chicago Defender, a weekly newspaper that was popular with African American
readers, ran a vitriolic opinion piece that claimed Bryn Mawr had gone, in
the span of a few years, from being one of the city's best public schools
to a run- down slum governed by a ghetto mentality. Our school principal,
Dr. Lavizzo, immediately hit back with a letter to the editor, defending
his community of parents and students and deeming the newspaper piece an
outrageous lie, which seems designed to incite only feelings of failure and
flight.Dr. Lavizzo was a round, cheery man who had an Afro that puffed out
on either side of his bald spot and who spent most of his time in an office
near the building's front door. It's clear from his letter that he
understood precisely what he was up against. Failure is a feeling long
before it becomes an actual result. It's vulnerability that breeds with
self- doubt and then is escalated, often deliberately, by fear. Those
feelings of failure he mentioned were everywhere already in my
neighborhood, in the form of parents who couldn't get ahead financially, of
kids who were starting to suspect that their lives would be no different,
of families who watched their better-off neighbors leave for the suburbs or
transfer their children to Catholic schools. There were predatory real
estate agents roaming South Shore all the while, whispering to homeowners
that they should sell before it was too late, that they'd help them get out
while you still can. The inference being that failure was coming, that it
was inevitable, that it had already half arrived. You could get caught up
in the ruin or you could escape it. They used the word everyone was most
afraid of ghetto dropping it like a lit match. My mother bought into none
of this. She'd lived in South Shore for ten years already and would end up
staying another forty. She didn't buy into fear mongering and at the same
time seemed equally inoculated against any sort of pie-in-the-sky idealism.
She was a straight-down-the-line realist, controlling what she could. At
Bryn Mawr, she became one of the most active members of the PTA, helping
raise funds for new classroom equipment, throwing appreciation dinners for
the teachers, and lobbying for the creation of a special multigrade
classroom that catered to higher-performing students. This last effort was
the brainchild of Dr. Lavizzo, who'd gone to night school to get his PhD in
education and had studied a new trend in grouping students by ability
rather than by again essence, putting the brighter kids together so they
could learn at a faster pace.");
```

```
}
```

test-right-wrap.tpp

```
def void start(){
   right();
   write("At school we were given an hour-long break for lunch each day.
Because my mother didn't work and our apartment was so close by, I usually
marched home with four or five other girls in tow, all of us talking
nonstop, ready to sprawl on the kitchen floor to play jacks and watch All
My Children while my mom handed out sandwiches.");
}
```

test-text-out.tpp

```
def void start(){
   int x;
   int y;
   x = 10;
   y = 50;
   textOut(x, y, "Hello World!");
}
```

test-top-margin.tpp

```
def void start(){
   setTopMargin(100);
   write("At school we were given an hour-long break for lunch each day.
Because my mother didn't work and our apartment was so close by, I usually
marched home with four or five other girls in tow, all of us talking
nonstop, ready to sprawl on the kitchen floor to play jacks and watch All
My Children while my mom handed out sandwiches. This, for me, began a habit
that has sustained me for life, keeping a close and high-spirited council
of girlfriends safe harbor of female wisdom. In my lunch group, we
dissected whatever had gone on that morning at school, any beefs we had
with teachers, any assignments that struck us as useless. Our opinions were
largely formed by committee. We idolized the Jackson 5 and weren't sure how
we felt about the Osmonds. Watergate had happened, but none of us
understood it. It seemed like a lot of old guys talking into microphones in
Washington, D.C., which to us was just a faraway city filled with a lot of
white buildings and white men. My mom, meanwhile, was plenty happy to serve
us. It gave her an easy window into our world. As my friends and I ate and
gossiped, she often stood by quietly, engaged in some household chore, not
```

hiding the fact that she was taking in every word. In my family, with four of us packed into less than nine hundred square feet of living space, we'd never had any privacy anyway. It mattered only sometimes. Craig, who was suddenly interested in girls, had started taking his phone calls behind closed doors in the bathroom, the phone's curlicue cord stretched taut across the hallway from its wall-mounted base in the kitchen.As Chicago schools went, Bryn Mawr fell somewhere between a bad school and a good school. Racial and economic sorting in the South Shore neighborhood continued through the 1970s, meaning that the student population only grew blacker and poorer with each year. There was, for a time, a citywide integration movement to bus kids to new schools, but Bryn Mawr parents had successfully fought it off, arguing that the money was better spent improving the school itself. As a kid, I had no perspective on whether the facilities were run-down or whether it mattered that there were hardly any white kids left. The school ran from kindergarten all the way through eighth grade, which meant that by the time I had reached the upper grades, I knew every light switch, every chalkboard and cracked patch of hallway. I knew nearly every teacher and most of the kids. For me, Bryn Mawr was practically an extension of home. As I was entering seventh grade, the Chicago Defender, a weekly newspaper that was popular with African American readers, ran a vitriolic opinion piece that claimed Bryn Mawr had gone, in the span of a few years, from being one of the city's best public schools to a run- down slum governed by a ghetto mentality. Our school principal, Dr. Lavizzo, immediately hit back with a letter to the editor, defending his community of parents and students and deeming the newspaper piece an outrageous lie, which seems designed to incite only feelings of failure and flight.Dr. Lavizzo was a round, cheery man who had an Afro that puffed out on either side of his bald spot and who spent most of his time in an office near the building's front door. It's clear from his letter that he understood precisely what he was up against. Failure is a feeling long before it becomes an actual result. It's vulnerability that breeds with self- doubt and then is escalated, often deliberately, by fear. Those feelings of failure he mentioned were everywhere already in my neighborhood, in the form of parents who couldn't get ahead financially, of kids who were starting to suspect that their lives would be no different, of families who watched their better-off neighbors leave for the suburbs or transfer their children to Catholic schools. There were predatory real estate agents roaming South Shore all the while, whispering to homeowners that they should sell before it was too late, that they'd help them get out while you still can. The inference being that failure was coming, that it was inevitable, that it had already half arrived. You could get caught up in the ruin or you could escape it. They used the word everyone was most

```
afraid of ghetto dropping it like a lit match. My mother bought into none
of this. She'd lived in South Shore for ten years already and would end up
staying another forty. She didn't buy into fear mongering and at the same
time seemed equally inoculated against any sort of pie-in-the-sky idealism.
She was a straight-down-the-line realist, controlling what she could. At
Bryn Mawr, she became one of the most active members of the PTA, helping
raise funds for new classroom equipment, throwing appreciation dinners for
the teachers, and lobbying for the creation of a special multigrade
classroom that catered to higher-performing students. This last effort was
the brainchild of Dr. Lavizzo, who'd gone to night school to get his PhD in
education and had studied a new trend in grouping students by ability
rather than by again essence, putting the brighter kids together so they
could learn at a faster pace.");
}
```

test-while.tpp

```
def void start(){
   int x;
   x = 0;
   while(x < 10){
      write("Falalalala lala la la");
      x = x + 1;
   }
}
```

test-wrap.tpp

```
def void start(){
   write("At school we were given an hour-long break for lunch each day.
Because my mother didn't work and our apartment was so close by, I usually
marched home with four or five other girls in tow, all of us talking
nonstop, ready to sprawl on the kitchen floor to play jacks and watch All
My Children while my mom handed out sandwiches.");
}
```