StoryBook

Anna Lawson
Beth Green
Nina Baculinao
Pratishta Yerakala

Motivation

- Pedagogical programming language
  - Reduce syntactical confusion
  - Syntax like English--e.g. periods instead of semicolons, “is” for assignment operator, avoid CS jargon
  - Encourage logical thinking and problem solving, keep syntax & vocab learning curve low
  - Programs read like stories
- Verbose and explicit OOP
  - Use the familiar context of stories--characters who have traits and perform actions, to enable children to more easily understand OOP

Before using StoryBook

After using StoryBook

Crash Course In StoryBook

Types:

number	(float)
words	(string)
letter	(char)
tof	(boolean)

Operators:

+ - * / % is > < >= <= , 's and or not

Comments:

~Block~

~~In line~~

Basic Program Structures:

Characters	(objects)
Actions	(methods)
Chapters	(function)

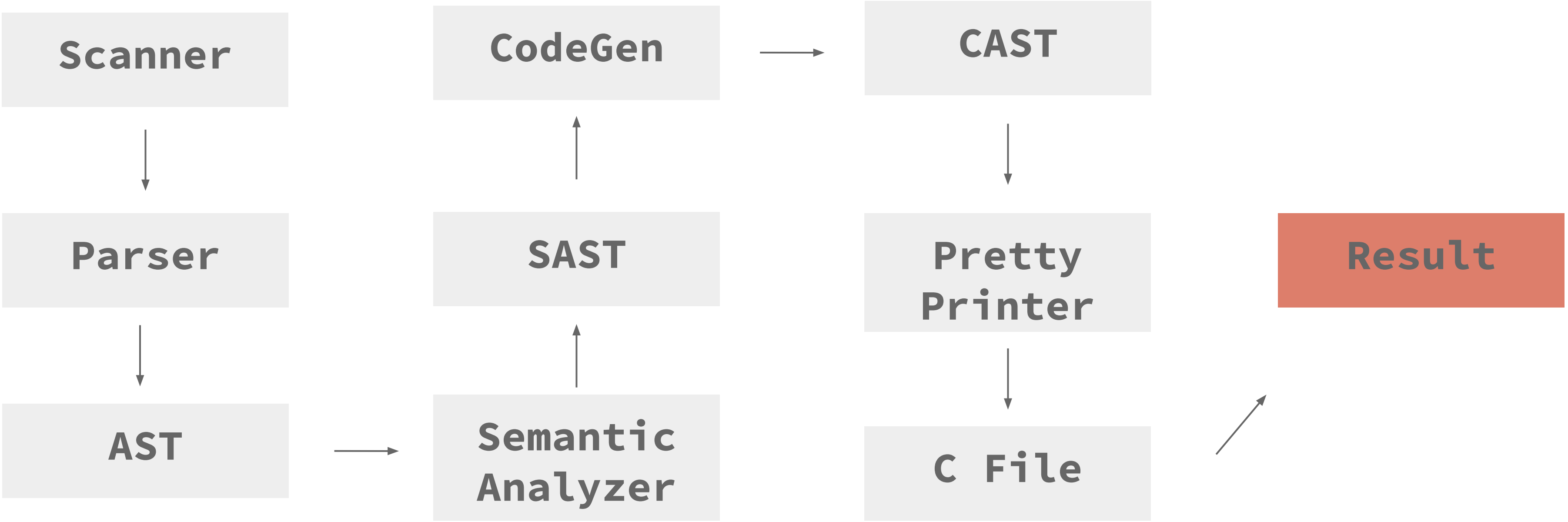
Example:

```
Character Monster( words n; number s ) {
  words name is n.
  number size is s.

  Action scare(words scream) returns
nothing {
    say(scream).
  }
}

Chapter plot() returns nothing {
  Character Monster Frank is new Monster
("Frankenstein"; 99).
  say(Frank's name + ":").
  Frank, scare("GLABARGHHHHH!").
}
```

Architecture



Challenges

1. + operator (concatenation)

2. Returning strings

3. Objects & Inheritance (structs, virtual tables)

Example (String)

```
Chapter whatTimeIsIt(words x) returns words {
  endwith("It's " + x + " o'clock." ).
}

Chapter plot() returns nothing {
  say(whatTimeIsIt("now" + " five")).
}
```

```
char * whatTimeIsIt(char * x) {
    char buf__1[ strlen("It's ") + strlen(x) + 1];
    sprintf(buf__1, "%s","It's ");
    sprintf(buf__1 + strlen(buf__1), "%s",x);
    char *_1 = buf__1;char buf__2[ strlen(_1) +
    strlen(" o'clock.") + 1];
    sprintf(buf__2, "%s",_1);
    sprintf(buf__2 + strlen(buf__2), "%s"," o'clock.");
    char *_2 = buf__2;
    char *_3 = malloc(strlen(_2));
    strcpy(_3, _2);
    return _3;
}

int main() {
    char buf__4[ strlen("now") + strlen(" five") + 1];
    sprintf(buf__4, "%s","now");
    sprintf(buf__4 + strlen(buf__4), "%s"," five");
    char *_4 = buf__4;char *_5 = whatTimeIsIt (_4 );
    char _6[strlen(_5)];
    strcpy(_6, _5);
    free(_5);
    printf ( "%s\n", _6);
}
```

Example (Objects)

```
Character Monster( words n; number s ) {
  words name is n.
  number size is s.

  Action scare(words scream) returns nothing {
    say(scream).
  }
}

Chapter plot() returns nothing {
  Character Monster Frank is new Monster
("Frankenstein"; 99).
  Frank, scare("GLABARGHHHHH!").
}
```

```
void *ptrs[1];
struct Monster;
struct table_Monster {
    void(*scare)(char * scream, struct Monster *_1);
};
struct Monster{
    const struct table_Monster *vtable;
    float size;
    char * name;
};
void Monster_scare(char * scream, struct Monster*_2) {
    printf ( "%s\n", scream);
}
static const struct table_Monster vtable_for_Monster = {
    Monster_scare
};
int main() {
    ptrs[0] = malloc((int)sizeof(struct Monster ));
    ((struct Monster *)ptrs[0])  -> name = "Frankenstein";
    ((struct Monster *)ptrs[0])  -> size = 99.;
    ((struct Monster *)ptrs[0])  ->vtable =
&vtable_for_Monster;

    struct Monster * Frank = ptrs[0];
    Frank->vtable->scare  ("GLABARGHHHHH!", Frank );
    for (int i = 0; i < (sizeof(ptrs)/sizeof(ptrs[0])); i++) {
        free(ptrs[i]); }
}
```

Lessons Learned

1. Writing OOP language is hard

2. Underestimating how much we have to do

3. Underestimating the power of testing