Timothy Richard "8bit" Mallin Burchfield

Esolangs

Douglas Jenn"hott"ings Smith

Some Pre-work:

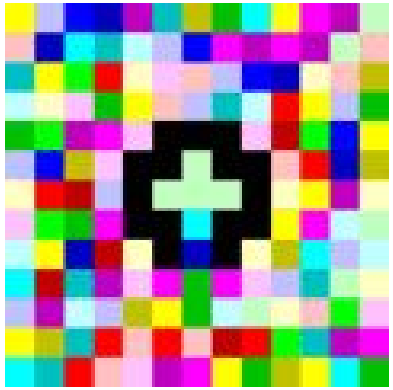
ssh into student06@cse.nd.edu

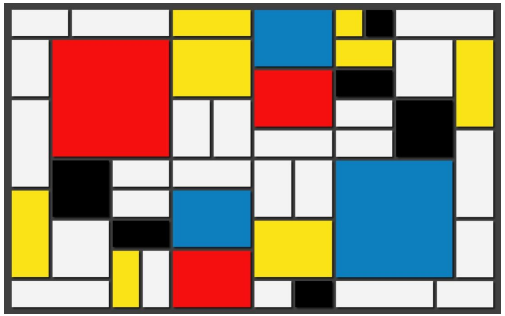
Add run scripts to your PATH:

```
export PATH=/afs/nd.edu/user7/dsmith47/pub/esolangs/Brainfuck/mindfuck:$PATH
export PATH=/afs/nd.edu/user7/dsmith47/pub/esolangs/Piet:$PATH
export PATH="/afs/nd.edu/user7/dsmith47/pub/esolangs/Shakespeare:$PATH"
export PATH="/afs/nd.edu/user15/pbui/pub/scratch/ghc/exec/bin/:$PATH"
```

Piet

- A programming...language.
- Piet Mondrian.

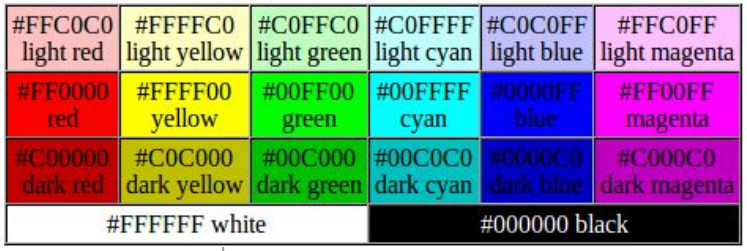




How?

- Codels - units
- Blocks - contiguous color sections
- Direction pointer ←,↑,→, or ↓
  - Initially → and upper left
- Codel Chooser ←, or→
  - Initially ←
- Stack of memory

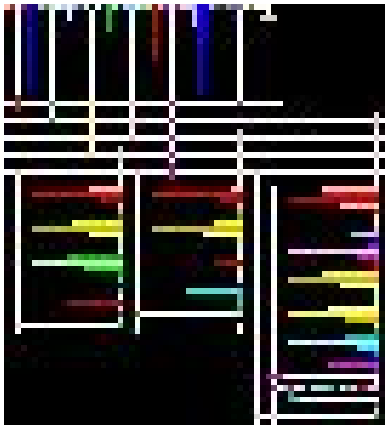
But wait, there's more



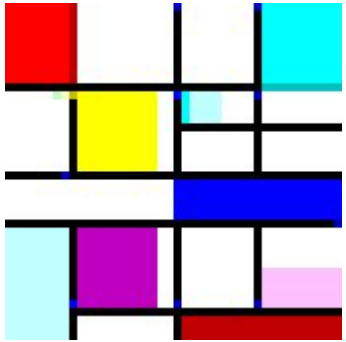
	Lightness change		
Hue Change	**None**	**1 Darker**	**2 Darker**
None		Push	Pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	Greater	Pointer	Switch
4 Steps	Duplicate	Roll	in(number)
5 Steps	in(char)	out(number)	out(char)

Examples

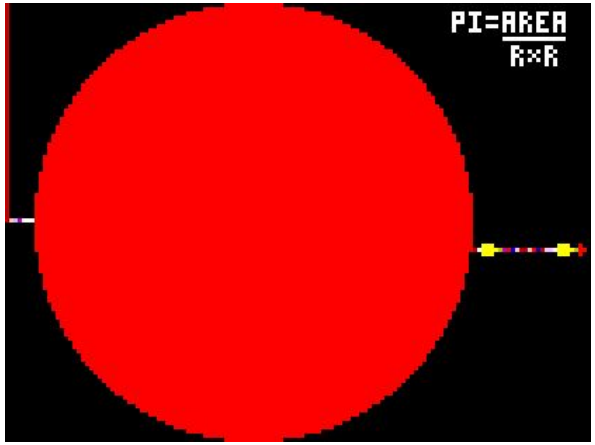
Fizzbuzz



"Piet"



Pi



Shakespeare Programming Language

Make Programming Beautiful (for the first time)

- A language designed for beautiful, “readable” source code
- One of the more complex grammars available in language, but flexible to the user/implementation
- View the project on sourceforge

How it works

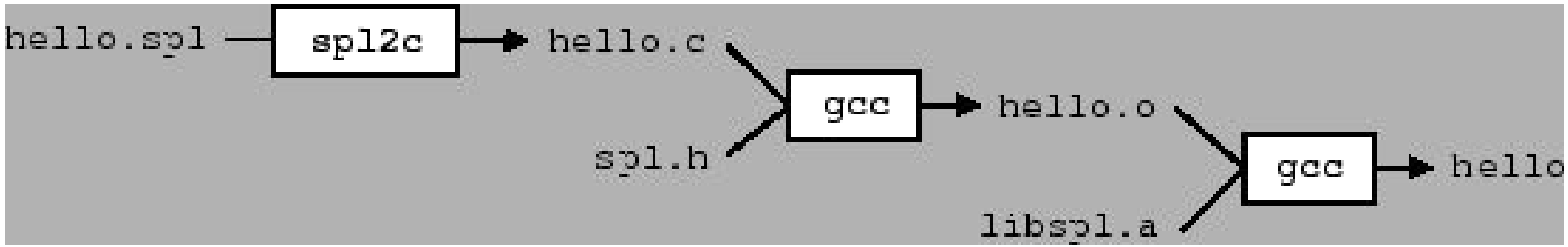
- **Variables: *Dramatis Personae* -** Declared at the beginning of code
- **Code Blocks: *Acts and Scenes*** - All programs have at least one Act and all Acts have at least one Scene, scenes contain the actual instructions
- **Scope Access: *Enter, Exit, Exunt* -** Characters cannot be used in a scene unless they are brought into the scene, they are freed from scope by Exti (plural Exunt)
- **Statements: *Lines*** - Characters make statments that:
    - Output their value
    - Change the value of another character

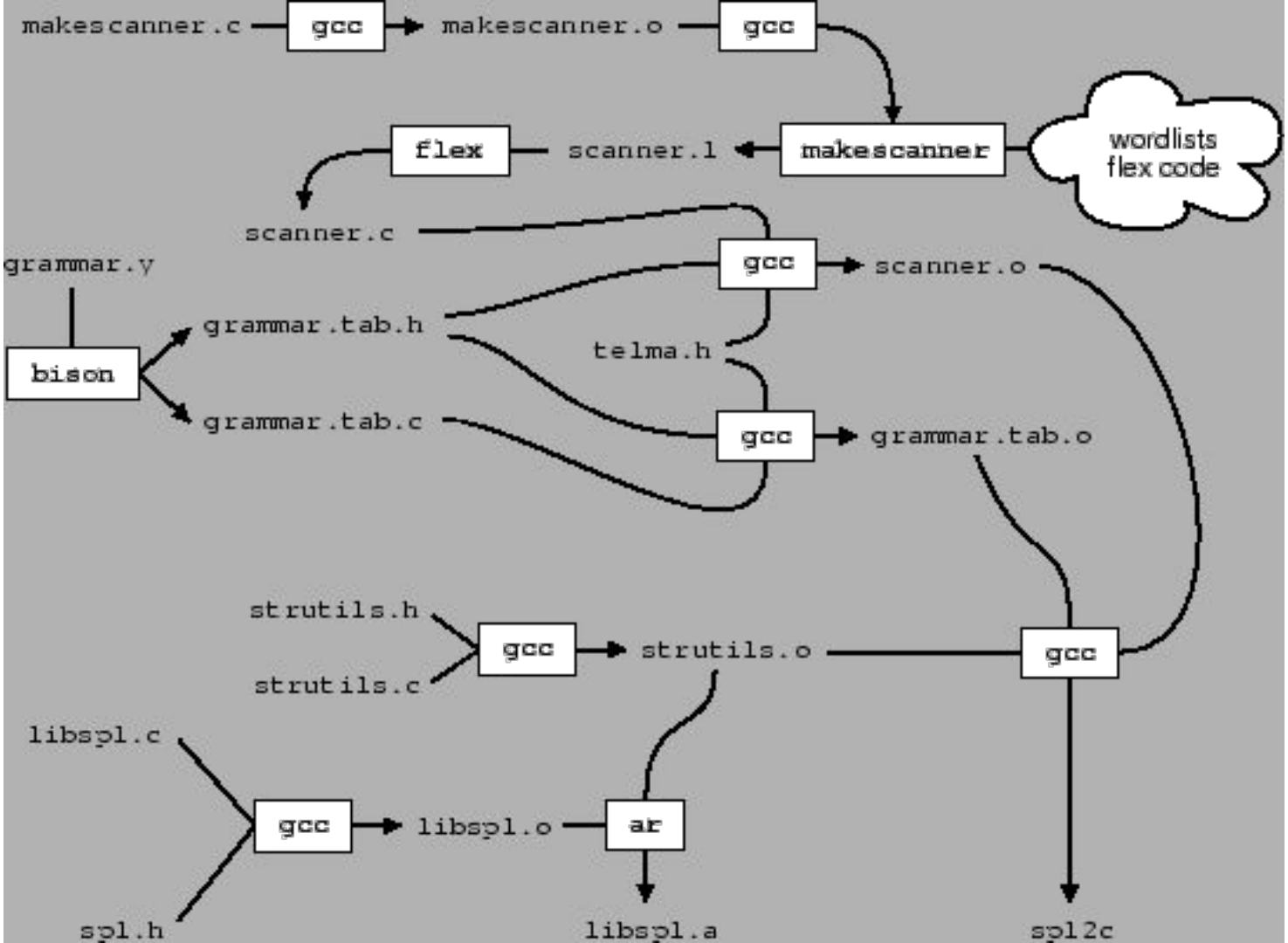
Doing the Math

- **Constants: nouns** - values of either 1 or -1
- **Adjectives: multipliers** - doubles value that it modifies in noun form
- **Operators:** operator-y things - “sum,” “difference,” etc.
- **Output:**
  - “Open your heart” - outputs numerical value of target character
  - “Speak your mind” - outputs Character evaluated by this number

Build. That. Play.

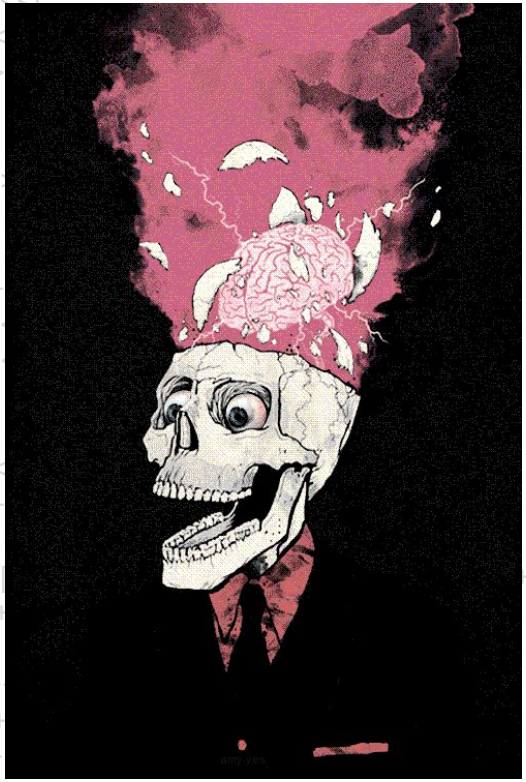
SPL transpiles to C, which can then compile to a binary executable




makescanner.c
gcc
makescanner.o
gcc
flex
scanner.l
makescanner
wordlists
flex code
scanner.c
grammar.y
gcc
scanner.o
grammar.tab.h
telma.h
bison
grammar.tab.c
gcc
grammar.tab.o
strutils.h
gcc
strutils.o
gcc
strutils.c
libspl.c
gcc
libspl.o
ar
spl.h
libspl.a
spl2c

Brainf*ck

- Very minimal
- 1993 Urban Müller
- Very much a puzzle language.



How?

Infinite array of integers. Initially all zero.

Pointer to element. Initially at first element.

0 9 1 3 2 4 2 4 1 2 4 0 0 0

^

But...how?

The compiler only cares about 8 characters.

+ Increment value at pointer by one.	**[** If the value at the pointer is nonzero, continue. Otherwise jump forward to the matching brace.
- Decrement value at pointer by one.	**]** If the value at the pointer is zero, continue. Otherwise jump backward to the matching brace.
> Move pointer to the right by one.	**,** Store one byte of input at pointer as ascii to decimal equivalent.
< Move pointer to the left by one.	**.** Output value at pointer as ascii.

How could anyone do anything in this language?

```
+++++++++[->++++++++<]>.

++++++++++++++++++++++++++++++++.

[-]

+++[->++++++++++<]>.

[-] +++++ +++++ .
```

Hi!

But for real, how do you do anything?

Control flow in BF

BF	Traditional Equivalent
, [*code*]	while (x != 0): *code*
, [- > *morecode* <]	for _ in range(N): *morecode*
, [*code* [-]]	if (x != 0): *code*
, >[-]+< [> - < [-]] > [*code*]	if (x == 0): *code* # *O(x)*

http://calmerthanyouare.org/2016/01/14/control-flow-in-brainfuck.html

MORE Control flow in BF

BF, example	Traditional Equivalent
>+< , [>-]>[>]<[*code* -]	if (x == 0): *code* *# faster*
>+< , ---- [>-]>[>]<[*code* -]	if (x == 4): *code*

+>,
--[
  ---[
  - [<->+++++[-]] *#fast clear*
  <[- foo]>
  ]
  <[- bar]>
  ]
<[- baz]
	switch (x) {
	  case 6 {
	  foo
	  }
	  case 5 {
	  bar
	  }
	  case 2 {
	  baz
	  }
	}

Examples

```
>,[>,]<[.<]
```

reverse

```
++++++++++[>++++++++++>+<<-]>[>.<-]
```

clear

```
>>+>>>>>,[>+>>,]>+[--[+<<<-]<[<+>-]<[<[->[<<<+>>>>+<-]<
<[>>+>[->]<<[<]
<-]>]>>>+<[[-]<[>+<-]<]>[[>>>]+<<<-<[<<[<<<]>>+>[>>>]<-]<
<[<<<]>[>>[>>
>]<+<<[<<<]>-]]+<<<]+[->>>]>>]>>[.>>>]
```

quicksort

```
>>>+[[-]>>[-]++>+>+++++++[<++++>>++<-]++>>+>+>
+++++[>++>++++++<<-]+>>>,<++[[>[
->>]<[>>]<<-]<[<]<+>>[>]>[<+>-[[<+>-]>]<[[[-]<]++<-[<
+++++++++>[<->-]>>]>>]]<<
]<]<[[<]>[[>]>>[>>]+[<<]<[<]<+>>-]>[>]+[->>]<<<<[[<<]
<[<]+<<[+>+<<-[>-->+<<-[>
+<[>>+<<-]]]>[<+>-]<]++>>-->[>]>>[>>]]<<[>>+<[[<]<]>
[[<<]<[<]+[-<+>>-[<<+>++>-
[<->[<<+>>-]]]<[>+<-]>]>[>]>]>[>>]>>]<<[>>+>>+>>]<<
[->>>>>>>>>]<<[>.>>>>>>>]<<[
>->>>>>]<<[>,>>>]<<[>+>]<<[+<<]<]
```

bf

One last example

Haskell: Esoterically Functional

- Haskell has a lot more mainstream support than the other languages covered
  - But it has a few quirks and hasn't kicked the "esoteric" label in some circles

Run in the ghci shell	Load functions and run in shell	Write scripts with main
> 1 + 1 2 > 4 / 2 - (10 * 5) -48.0	Double.hs: double x = x + x > :l double 4 8	

Everything's a Function!

Things that are Functions	Things that are not functions
let add1 x = x + 1 let 8bit = "<3" let deprecateDoug ="Get this over with already!!!" let carballNums = [3,5,7,9] ...	{}

…Even when something’s a list!

How to build a list	
: prepend	
++ concatenation	[‘h’, ‘e’, ‘l’] ++ [‘l’,’o’]
List Comprenshsions	[x*x + 1 | x <- [1..100]]
.. ranges	[1..20] [‘a’ .. ‘’z] [3,6..18] [‘’a,’c’ .. ‘y’]

Embracing Infinity

Haskell’s simulation of ‘infinite’ lists is one of most defining (and strange) features

Cycle	> cycle [2,4..10] [2,4,6,8,10,2,4,6,8,10...]
Repeat	> repeat 0 [0,0,0,0,0,0,0…]]