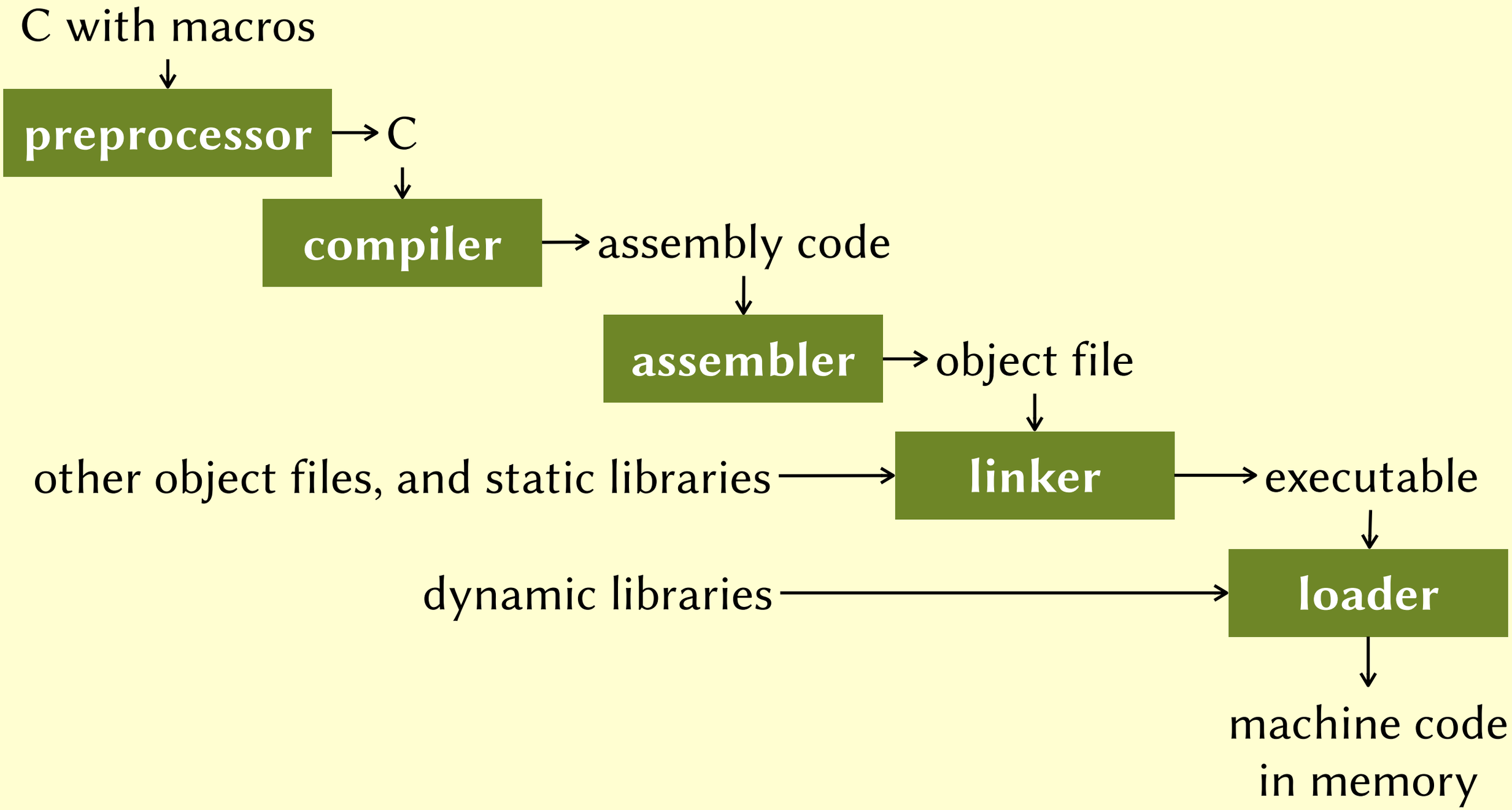
Lecture 11: Building

John Wickerson

Compilers

The compilation ecosystem



Coursework news

- Continuous integration improved!
  - Many thanks to Quentin Corradi, Filip Wojcicki, James Nock, William Huynh, and Simon Staal.
- To benefit from this, please:
  - Clone the master `langproc-cw` repo.
  - Replace its `src` and `include` directories with your own.
  - Overwrite your repo with this.

Coursework news

GitHub Actions / Test results
succeeded 3 days ago in 1s

1 passed, 85 failed and 0 skipped

tests 1 passed, 85 failed

❌ build/junit_results.xml

86 tests were completed in **NaNms** with **1** passed, **85** failed and **0** skipped.

Test suite	Passed	Failed	Skipped	Time
Integration test	1✅	85❌		NaNms

❌ Integration test

```
✅ tests/_example/example.c
❌ tests/array/declare_global.c
    > Failed to compile testcase:
❌ tests/array/declare_local.c
    > Failed to compile testcase:
❌ tests/array/index_constant.c
    > Failed to compile testcase:
❌ tests/array/index_expression.c
    > Failed to compile testcase:
❌ tests/array/index_variable.c
    > Failed to compile testcase:
❌ tests/control_flow/for_multiple_c
```

Coursework news

langproc-2025-cw-BondPotter

- **src**
- **include**
- **other**
- **stuff**

langproc-cw

- **src**
- **include**
- **other**
- **stuff**

Writing library code

average.c

```
int mean(int a, int b, int c) {
  return (a + b + c) / 3;
}

int median(int a, int b, int c) {
  if (a <= b && b <= c) return b;
  if (c <= b && b <= a) return b;
  if (a <= c && c <= b) return c;
  if (b <= c && c <= a) return c;
  return a;
}

int mode(int a, int b, int c) {
  if (a == b) return a;
  if (b == c) return b;
  if (c == a) return c;
  return a;
}
```

average.h

```
// Mean of three numbers
int mean(int a, int b, int c);

// Median of three numbers
int median(int a, int b, int c);

// Mode of three numbers
int mode(int a, int b, int c);
```

main.c

```
#include "average.h"
#include <stdio.h>

int main () {
  printf("%d\n", mean(1,2,3));
}
```

Building a static library

average.c

```
int mean(int a, int b, int c) {
  return (a + b + c) / 3;
}

int median(int a, int b, int c) {
  if (a <= b && b <= c) return b;
  if (c <= b && b <= a) return b;
  if (a <= c && c <= b) return c;
  if (b <= c && c <= a) return c;
  return a;
}

int mode(int a, int b, int c) {
  if (a == b) return a;
  if (b == c) return b;
  if (c == a) return c;
  return a;
}
```

gcc -c

average.o

```
01001010101000010010101010
00101010101001010101000001
00101010100101010101001001
01011101100111010001011111
11011001010010110010001010
00010100101011010010101010
00001001010101001010101010
01001010111011001110001000
01111110101010011001001000
10100101100111010001011111
11011001010010110100101010
01000010010101010010101010
01010010101110110011101000
01011111101100101001011000
10001010000100010100011010
```

gcc -L. -laverage

main.c

```
#include "average.h"
#include <stdio.h>

int main () {
  printf("%d\n", mean(1,2,3));
}
```

ar rcs

libaverage.a

```
01001010101000010010101010
00101010101001010101000001
00101010100101010101001001
01011101100111010001011111
11011001010010110010001010
00010100101011010010101010
00001001010101001010101010
01001010111011001110001000
01111110101010011001001000
10100101100111010001011111
11011001010010110100101010
01000010010101010010101010
01010010101110110011101000
01011111101100101001011000
10001010000100010100011010
```

a.out

```
01001010101000010010101010
00101010101001010101000001
00101010100101010101001001
01011101100111010001011111
11011001010010110010001010
00010100101011010010101010
00001001010101001010101010
01001010111011001110001000
01111110101010011001001000
10100101100111010001011111
11011001010010110100101010
01000010010101010010101010
01010010101110110011101000
01011111101100101001011000
10001010000100010100011010
```

Building a dynamic library

average.c

```
int mean(int a, int b, int c) {
  return (a + b + c) / 3;
}

int median(int a, int b, int c) {
  if (a <= b && b <= c) return b;
  if (c <= b && b <= a) return b;
  if (a <= c && c <= b) return c;
  if (b <= c && c <= a) return c;
  return a;
}

int mode(int a, int b, int c) {
  if (a == b) return a;
  if (b == c) return b;
  if (c == a) return c;
  return a;
}
```

gcc -fPIC -c

average.o

```
0100101010100001001010101
0010101010100101010100001
0010101010010101010101001
0101110110011101000101111
1101100101001011001000101
0001010010101110010101101
0000100101010100101010101
0100101011101100111010001
0111111010101001011001000
1010010110011101000101111
1101100101001011010010101
0100001001010101001010101
0101001010111011001110100
0101111110110010100101100
1000101000010001010001101
```

gcc -shared

libaverage.so

```
0100101010100001001010101
0010101010100101010100001
0010101010010101010101001
0101110110011101000101111
1101100101001011001000101
0001010010101110010101101
0000100101010100101010101
0100101011101100111010001
0111111010101001011001000
1010010110011101000101111
1101100101001011010010101
0100001001010101001010101
0101001010111011001110100
0101111110110010100101100
1000101000010001010001101
```

main.c

```
#include "average.h"
#include <stdio.h>

int main () {
  printf("%d\n", mean(1,2,3));
}
```

gcc -Wl,-rpath,. -L. -laverage

a.out

```
0100101010100001001010101
0010101010100101010100001
0010101010010101010101001
0101110110011101000101111
1101100101001011001000101
0001010010101110010101101
0000100101010100101010101
0100101011101100111010001
0111111010101001011001000
1010010110011101000101111
1101100101001011010010101
0100001001010101001010101
0101001010111011001110100
0101111110110010100101100
1000101000010001010001101
```

Static vs dynamic linking

Building manually

- With static linking:
  - `gcc -c average.c -o average.o`
  - `ar rcs libaverage.a average.o`
  - `gcc main.c -L. -laverage`

compile average.c

make a static library

compile main.c and link it with library

- With dynamic linking:
  - `gcc -c -fPIC average.c -o average.o`
  - `gcc -shared average.o -o libaverage.so`
  - `gcc main.c -Wl,-rpath,. -L. -laverage`

compile average.c

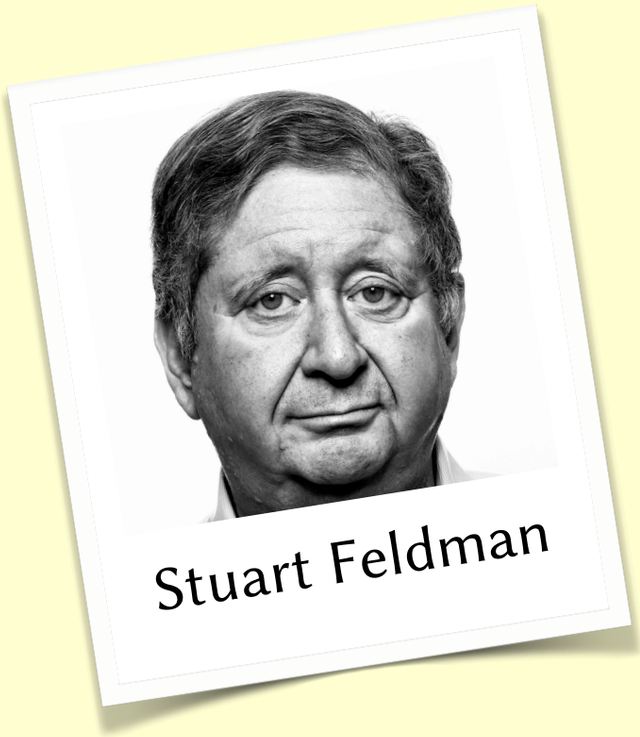
make a shared library

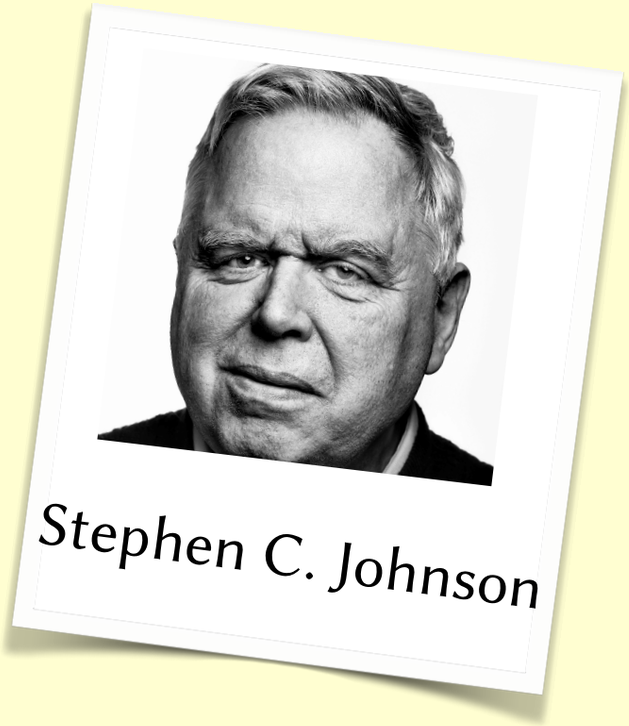
compile main.c and tell it where to find library at runtime

- Then later on:
  - `./a.out`

Building automatically

- Make was written in 1976 at AT&T Labs by Stuart Feldman.


Stuart Feldman


Stephen C. Johnson

Turn into a Makefile

```
average.o:
    gcc -c average.c -o average.o

libaverage.a:
    ar rcs libaverage.a average.o

static.out:
    gcc main.c -L. -laverage -o static.out

average_pic.o:
    gcc -c -fPIC average.c -o average_pic.o

libaverage.so:
    gcc -shared average_pic.o -o libaverage.so

dynamic.out:
    gcc main.c -Wl,-rpath,. -L. -laverage -o dynamic.out
```

Turn into a Makefile

```
average.o: average.c
    gcc -c average.c -o average.o

libaverage.a: average.o
    ar rcs libaverage.a average.o

static.out: main.c libaverage.a
    gcc main.c -L. -laverage -o static.out

average_pic.o: average.c
    gcc -c -fPIC average.c -o average_pic.o

libaverage.so: average_pic.o
    gcc -shared average_pic.o -o libaverage.so

dynamic.out: main.c libaverage.so
    gcc main.c -Wl,-rpath,. -L. -laverage -o dynamic.out
```

Make it shorter

```
average.o: average.c
    gcc -c $< -o $@

libaverage.a: average.o
    ar rcs $@ $<

static.out: main.c libaverage.a
    gcc $< -L. -laverage -o $@

average_pic.o: average.c
    gcc -c -fPIC $< -o $@

libaverage.so: average_pic.o
    gcc -shared $< -o $@

dynamic.out: main.c libaverage.so
    gcc $< -Wl,-rpath,. -L. -laverage -o $@
```

Add macros

```
CC=gcc
CCFLAGS=-g



average.o: average.c
    $(CC) $(CCFLAGS) -c $< -o $@

libaverage.a: average.o
    ar rcs $@ $<

static.out: main.c libaverage.a
    $(CC) $(CCFLAGS) $< -L. -laverage -o $@

average_pic.o: average.c
    $(CC) $(CCFLAGS) -c -fPIC $< -o $@

libaverage.so: average_pic.o
    $(CC) $(CCFLAGS) -shared $< -o $@

dynamic.out: main.c libaverage.so
    $(CC) $(CCFLAGS) $< -Wl,-rpath,. -L. -laverage -o $@
```

Cleaning up

```
CC=gcc
CCFLAGS=-g



average.o: average.c
    $(CC) $(CCFLAGS) -c $< -o $@

libaverage.a: average.o
    ar rcs $@ $<

static.out: main.c libaverage.a
    $(CC) $(CCFLAGS) $< -L. -laverage -o $@

average_pic.o: average.c
    $(CC) $(CCFLAGS) -c -fPIC $< -o $@

libaverage.so: average_pic.o
    $(CC) $(CCFLAGS) -shared $< -o $@

dynamic.out: main.c libaverage.so
    $(CC) $(CCFLAGS) $< -Wl,-rpath,. -L. -laverage -o $@
```

```
.PHONY: clean

clean:
    rm -f average.o
    rm -f average_pic.o
    rm -f libaverage.a
    rm -f libaverage.so
    rm -f static.out
    rm -f dynamic.out
```

Cleaning up

```
CC=gcc
CCFLAGS=-g



average.o: average.c
    $(CC) $(CCFLAGS) -c $< -o $@

libaverage.a: average.o
    ar rcs $@ $<

static.out: main.c libaverage.a
    $(CC) $(CCFLAGS) $< -L. -laverage -o $@

average_pic.o: average.c
    $(CC) $(CCFLAGS) -c -fPIC $< -o $@

libaverage.so: average_pic.o
    $(CC) $(CCFLAGS) -shared $< -o $@

dynamic.out: main.c libaverage.so
    $(CC) $(CCFLAGS) $< -Wl,-rpath,. -L. -laverage -o $@
```

```
.PHONY: clean

clean:
    rm -f *.o
    rm -f libaverage.a
    rm -f libaverage.so
    rm -f static.out
    rm -f dynamic.out
```

Aside

- Microsoft Excel is a build-automation system in disguise!

	A	B	C
1	42		36
2		=A1+17	
3			=C1-B2
4	=C3*A1	92	

Top tip

- Don't put files that can be generated by your Makefile into your GitHub repository.