CS 241: Systems Programming
Lecture 9. More C

Spring 2020

Prof. Stephen Checkoway

Operators

The same as Java

- Arithmetic: `+`, `-`, `*`, `/`, `%`
- Logical: `&&`, `||`, `!`
- Bitwise: `&`, `|`, `^`, `~`, `<<`, `>>`
- Pre/post increment, decrement: `++`, `--`
- Relational: `==`, `!=`, `<`, `<=`, `>`, `>=`
- Assignment: `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `<<=`, `>>=`

There are some others we'll talk about later

- `sizeof`
- `.`
- `->`

C has pre- and post-increment (++) and -decrement (--) operators. What does this code print? (%d means print an integer)

```
int main(void) {
  int x = 3;
  int y = 5;
  printf("%d %d\n", x--, ++y);
  return 0;
}
```

A. 2 5

B. 2 6

C. 3 5

D. 3 6

E. Undefined

C has pre- and post-increment (++) and -decrement (--) operators. What does this code print? (%d means print an integer)

```
int main(void) {
  int x = 3;
  printf("%d\n", x-- + --x);
  return 0;
}
```

A. 3

B. 4

C. 5

D. 6

E. Undefined

Huge difference from Java

C is **full** of undefined behavior, implementation-defined behavior, and unspecified behavior

Undefined behavior gives the compiler license to do whatever it wants, including nothing

Implementation-defined behavior means the compiler gets to choose (and document) its behavior

Unspecified behavior means the compiler gets to pick from among several choices

What does the code print?

A.
```
foo
bar
1 2
```

B.
```
bar
foo
1 2
```

C.
```
1 2
foo
bar
```

D. Undefined behavior, could print anything

E. Unspecified behavior, either A or B.

```c
#include <stdio.h>

int foo(void) {
  printf("foo\n");
  return 1;
}

int bar(void) {
  printf("bar\n");
  return 2;
}

int main(void) {
  printf("%d %d\n", foo(), bar());
  return 0;
}
```

Control flow

`if` statements; `for`, `while`, `do-while` loops almost identical to Java

zero is false, nonzero is true

Examples

```
int signum(int x) {
  if (x < 0)
    return -1;
  if (x > 0)
    return 1;
  return 0;
}
```

```
int sum_of_squares(int n) {
  int result = 0;
  for (int i = 1; i < n; ++i)
    result += i * i;
  return result;
}
```

Examples

```
bool get_reponse(void) {
  int response;
  do {
    printf("Enter y or n\n");
    response = getchar();
  } while (response != EOF
           && response != 'y'
           && response != 'n');
  return response == 'y';
}
```

Compiling code

```
$ ⟨compiler⟩ ⟨options⟩ ⟨.c files⟩ ⟨libraries⟩
```

```
$ clang -Wall -o program -std=c11 *.c -lm
```

If you omit `-o output`, the default is a.out

If you omit `-std=c11`, `clang` and `gcc` have different defaults!

Compiler options (gcc/clang)

Compiler options (gcc/clang)

`-E` preprocessor only

Compiler options (gcc/clang)

`-E`	preprocessor only
`-S`	compile only (no assembly or linking)

Compiler options (gcc/clang)

`-E`	preprocessor only
`-S`	compile only (no assembly or linking)
`-c`	compile/assemble (produce `.o` file)

Compiler options (gcc/clang)

`-E`	preprocessor only
`-S`	compile only (no assembly or linking)
`-c`	compile/assemble (produce `.o` file)
`-o foo`	specify output file as `foo`

Compiler options (gcc/clang)

`-E`	preprocessor only
`-S`	compile only (no assembly or linking)
`-c`	compile/assemble (produce `.o` file)
`-o foo`	specify output file as `foo`
`-lxxx`	use library named `libxxx.so` or `libxxx.a`

Compiler options (gcc/clang)

`-E`	preprocessor only
`-S`	compile only (no assembly or linking)
`-c`	compile/assemble (produce `.o` file)
`-o foo`	specify output file as `foo`
`-lxxx`	use library named `libxxx.so` or `libxxx.a`
`-g`	emit debugging symbols (enables debugging)

Compiler options (gcc/clang)

`-E`	preprocessor only
`-S`	compile only (no assembly or linking)
`-c`	compile/assemble (produce `.o` file)
`-o foo`	specify output file as `foo`
`-lxxx`	use library named `libxxx.so` or `libxxx.a`
`-g`	emit debugging symbols (enables debugging)
`-std=c11`	use C11 standard

Compiler options (gcc/clang)

`-E`	preprocessor only
`-S`	compile only (no assembly or linking)
`-c`	compile/assemble (produce `.o` file)
`-o foo`	specify output file as `foo`
`-lxxx`	use library named `libxxx.so` or `libxxx.a`
`-g`	emit debugging symbols (enables debugging)
`-std=c11`	use C11 standard
`-pedantic`	be pedantic
`-Wall`	turn on "all" warnings
`-Wextra`	turn on extra warnings
`-Werror`	make warnings into errors

Formatting your code

It's more important to be consistent than anything else when it comes to format

Use tools!

```
$ clang-format foo.c      # Writes formatted code to stdout

$ clang-format -i foo.c   # Writes formatted code back to foo.c
```

In-class exercise

https://checkoway.net/teaching/cs241/2020-spring/exercises/Lecture-09.html

Grab a laptop and a partner and try to get as much of that done as you can!