CS 240: Programming in C

Lecture 25: Assembly, goto, Makefiles

PURDUE UNIVERSITY

Announcements

- No class next week
    - No labs either
    - There *will* be office hours on Monday and Tuesday
    - Finish homework 12 early!

Assembly language

- C gets compiled into a lower-level language called assembly language
- Each instruction represents one CPU instruction
- We can tell gcc to output assembly in a human-readable format

```
$ gcc -S helloworld.c
```

Assembly output

```
#include <stdio.h>

int main() {
  printf(“Hello, world!\n”);
  return 0;
}
```

Assembly output

```
    .file   "helloworld.c"
    .text
    .section    .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string "Hello, world!"
    .section    .text.startup,"ax",@progbits
    .p2align 4
    .globl  main
    .type   main, @function
main:
.LFB11:
    .cfi_startproc
    subq   $8, %rsp
    .cfi_def_cfa_offset 16
    leaq   .LC0(%rip), %rdi
    call   puts@PLT
    xorl   %eax, %eax
    addq   $8, %rsp
    .cfi_def_cfa_offset 8
    ret
    .cfi_endproc
```

Assembly output

```
    .file   "helloworld.c"
    .text
    .section    .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string "Hello, world!"
    .section    .text.startup,"ax",@progbits
    .p2align 4
    .globl  main
    .type   main, @function
main:
.LFB11:
    .cfi_startproc
    subq   $8, %rsp
    .cfi_def_cfa_offset 16
    leaq   .LC0(%rip), %rdi
    call   puts@PLT
    xorl   %eax, %eax
    addq   $8, %rsp
    .cfi_def_cfa_offset 8
    ret
    .cfi_endproc
```

printf() was replaced by puts()

Using assembly language in C

- We generally do not have control over which assembly language instructions are chosen
- Most compilers support a way of embedding specific instructions

Using assembly in C

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
  unsigned char x = atoi(argv[1]);

  __asm__ __volatile__(“add $1,%0” : “=r”(x) : “0”(x));

  printf(“%d\n”, x);
  return x;
}
```

Using assembly in C

```
      .cfi_startproc
      pushq   %rbx
      .cfi_def_cfa_offset 16
      .cfi_offset 3, -16
      movq 8(%rsi), %rdi
      movl $10, %edx
      xorl %esi, %esi
      call strtol@PLT
      add $1,%al
      movzbl  %al, %ebx
      leaq .LC0(%rip), %rdi
      xorl %eax, %eax
      movl %ebx, %esi
      call printf@PLT
      movl %ebx, %eax
      popq %rbx
      .cfi_def_cfa_offset 8
      ret
      .cfi_endproc
```

Assembly

- Most details of assembly are out of the scope of this course
- I don't expect you to be able to read or write assembly on an exam
- Just know that you can output assembly and inject it into C code if necessary
    - It's rarely necessary, but there are applications for it

goto

- In C, we can define “labels” and then “goto” them

```
int func(int x) {
  int sum = 0;

again:
  sum = sum + x;
  x = x - 1;
  if (x <= 0)
    goto get_out;
  else
    goto again;

get_out:
  return sum;
}
```

Why is goto bad?

- In this class, use of goto is forbidden
- Most people will tell you to avoid using it
- Dijkstra made the case that goto was harmful for the following reasons
    - It prevents the compiler from being able to optimize / reduce the program
    - It makes your code unreadable
    - It is really not necessary
        - You can always rewrite to have the same functionality without goto

Why does C have a goto?

- Because…
  - The compiler doesn’t have any more difficulty analyzing a program with gotos in it
  - It often makes the program clearer to read
  - It is very useful, sometimes

goto example

- How can goto make a program clearer to read?

```
start_over:
  for (int x = 0; x < 5000; x++) {
    ptr = array[x];
    while (ptr->val < level) {
      while (ptr->next != 0 && ptr->val < level) {
        if (ptr->total == 0) {
          level++;
          goto start_over;
        }
      }
      sum += ptr->total;
    }
  }
```

When is goto useful?

- When it is necessary to break out of deeply nested loops (previous example)
- When you’re building a state machine in software

- It can be a powerful tool if you know what you’re doing
- But in general, you should still avoid using goto unless there is a really good reason

Makefile

- A Makefile is a file named “Makefile” in your build directory
- It’s a simple way to help organize code compilation
- Composed of rules
    - Target - usually a file to generate
        - Can be an action (e.g., “make clean”)
    - Prerequisites - used to create the target
    - Recipe - action to carry out
        - Must start with a tab!

Makefile

- Say we want to compile using this command:

```
$ gcc -o hello hello.c hellofunc.c -I.
```

- Our Makefile could look like this:

```
hello: hello.c hellofunc.c
  gcc -o hello hello.c hellofunc.c -I.
```

- And now we can compile by just running:

```
$ make hello
```

Makefile variables

- We can use variables to generalize

```
CC=gcc
CFLAGS=-I.

hello: hello.c hellofunc.c
  $(CC) -o hello hello.c hellofunc.c $(CFLAGS)
```

Makefile variables

- There are also special variables for targets and prerequisites

```
CC=gcc
CFLAGS=-I.

hello: hello.c hellofunc.c
  $(CC) -o $@ $^ $(CFLAGS)
```

  - `$@` is the target name
  - `$^` is the list of prerequisites

Generic targets

- You can specify patterns for targets and prereqs

```
CC=gcc
CFLAGS=-I.
DEPS=hello.h

%.o: %.c $(DEPS)
  $(CC) -c -o $@ $< $(CFLAGS)

hello: hello.o hellofunc.o
  $(CC) -o $@ $^
```

  - $< is the name of the FIRST prerequisite in the list

More variables

```
CC=gcc
CFLAGS=-I.
DEPS=hello.h
OBJ=hello.o hellofunc.o

%.o: %.c $(DEPS)
  $(CC) -c -o $@ $< $(CFLAGS)

hello: $(OBJ)
  $(CC) -o $@ $^
```

More targets

```
CC=gcc
CFLAGS=-I.
DEPS=hello.h
OBJ=hello.o hellofunc.o

all: hello

%.o: %.c $(DEPS)
  $(CC) -c -o $@ $< $(CFLAGS)

hello: $(OBJ)
  $(CC) -o $@ $^

clean:
  rm -f hello *.o
```

Makefiles

- You can use Makefiles for more than just compiling C code!
- For example, all the homework handouts and exams use Makefiles to compile LaTeX documents

```
TARGETS=final.pdf

all: $(TARGETS)

%.pdf:
  pdflatex $*
  pdflatex $*
  pdflatex $*
```

Lots more

- There is a lot more to learn about Makefiles
- We’ve only scratched the surface
- https://www.gnu.org/software/make/manual/html_node/index.html