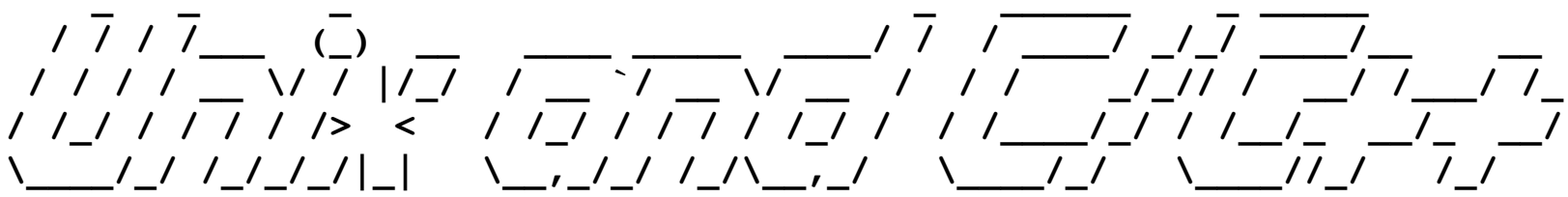
UNIVERSITY OF OREGON

CIS 330:



Lecture 5: memory errors

April 13th, 2016

Hank Childs, University of Oregon

Announcements

- Projects
  - 1C delayed until Thurs the 14th
  - 4A assigned today
  - 2B assigned on Friday
- Discussion this week on debuggers/memory checkers
- No class on Weds April 27
  - likely a short YouTube replacement lecture

Outline

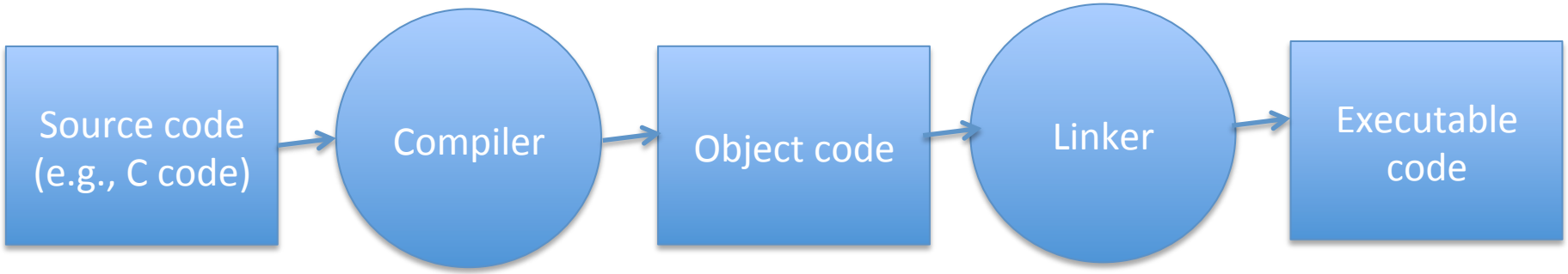
- Review
- Project 1C Overview
- Tar
- Memory Errors
- Project 4A
- Background Info for 4A

Outline

- Review
- Project 1C Overview
- Tar
- Memory Errors
- Project 4A
- Background Info for 4A

Build: The Actors

- File types
  - Source code
  - Object code
  - Executable code
- Programs
  - Compiler
  - Linker



Compilers, Object Code, and Linkers

- Compilers transform source code to object code
  - Confusing: most compilers also secretly have access to linkers and apply the linker for you.
- Object code: statements in machine code
  - not executable
  - intended to be part of a program
- Linker: turns object code into executable programs

Our first gcc program: named output

CIS330 — bash — 80×24

```
C02LN00GFD58:CIS330 hank$ cat t.c
#include <stdio.h>
int main()
{
   printf("hello world!\n");
}
C02LN00GFD58:CIS330 hank$ gcc t.c
C02LN00GFD58:CIS330 hank$ ./a.out
hello world!
C02LN00GFD58:CIS330 hank$ gcc -o helloworld t.c
C02LN00GFD58:CIS330 hank$ ./helloworld
hello world!
C02LN00GFD58:CIS330 hank$ ls -l helloworld
-rwxr-xr-x  1 hank  staff  8496 Apr  3 15:15 helloworld
C02LN00GFD58:CIS330 hank$
```

"-o" sets name of output

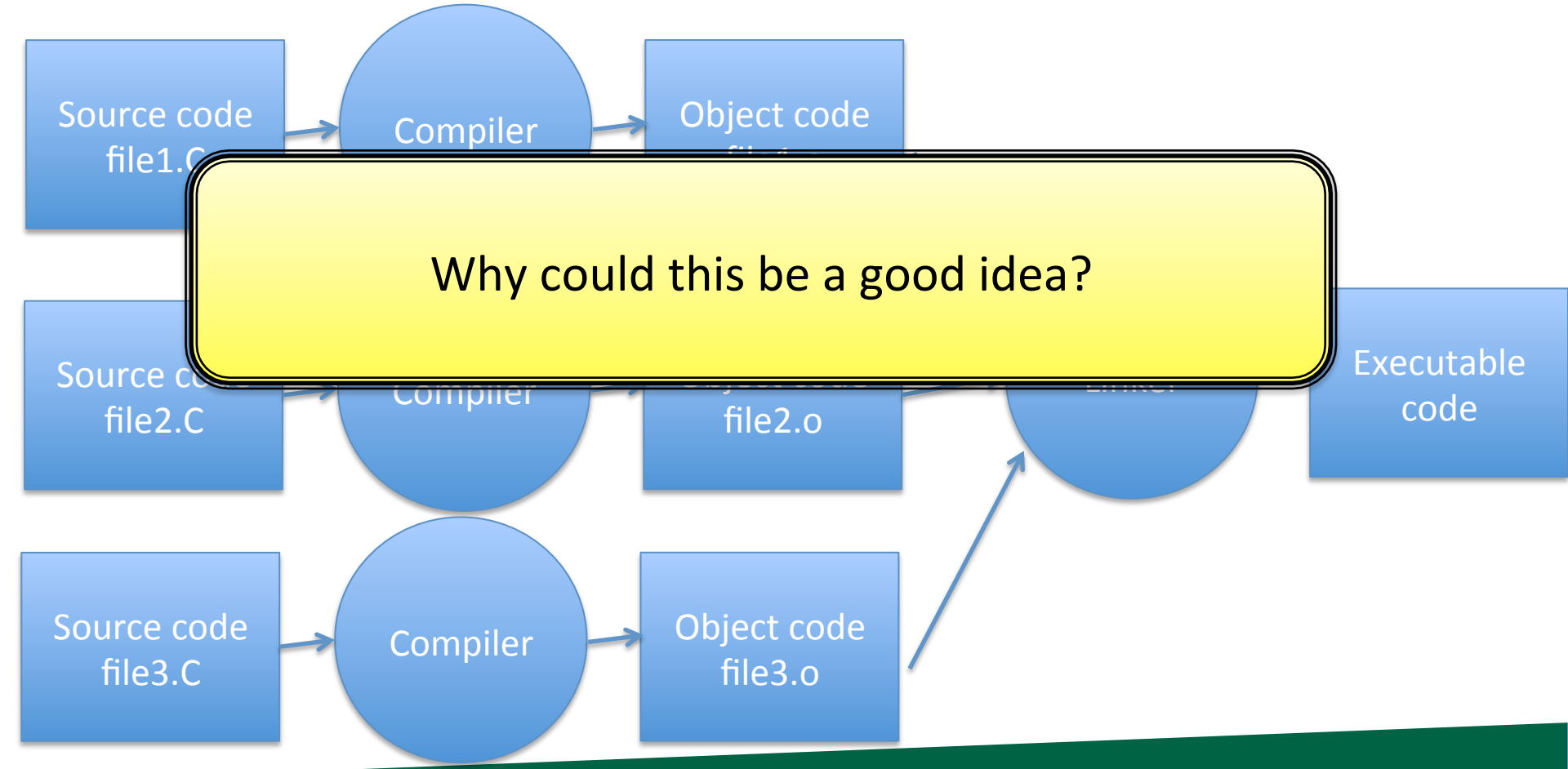
Output name is different

Output has execute permissions

gcc flags: debug and optimization

- "gcc –g": debug symbols
  - Debug symbols place information in the object files so that debuggers (gdb) can:
    - set breakpoints
    - provide context information when there is a crash
- "gcc –O2": optimization
  - Add optimizations … never fails
- "gcc –O3": provide more optimizations
  - Add optimizations … sometimes fails
- "gcc –O3 –g"
  - Debugging symbols slow down execution … and sometimes compiler won't do it anyways…

Large code development



Multi-file development: example

```
fawcett:330 childs$ cat t1.c
int doubler(int x)
{
   return 2*x;
}
fawcett:330 childs$ cat t2.c
int main()
{
    return doubler(5);
}
fawcett:330 childs$ gcc -c t1.c
fawcett:330 childs$ gcc -c t2.c
fawcett:330 childs$ gcc -o both t2.o t1.o
fawcett:330 childs$ ./both
fawcett:330 childs$ echo $?
10
```

cat is a Unix command that prints the contents of a file

$? is a shell construct that has the return value of the last executed program

Multi-file development: example

```
fawcett:330 childs$ cat t1.c
int doubler(int x)
{
   return 2*x;
}
fawcett:330 childs$ cat t2.c
int main()
{
    return doubler(5);
}
fawcett:330 childs$ gcc -c t1.c
fawcett:330 childs$ gcc -c t2.c
fawcett:330 childs$ gcc -o both t2.o t1.o
fawcett:330 childs$ ./both
fawcett:330 childs$ echo $?
10
```

```
fawcett:330 childs$ gcc -o both t2.o
Undefined symbols:
  "_doubler", referenced from:
      _main in t2.o
ld: symbol(s) not found
collect2: ld returned 1 exit status
fawcett:330 childs$ gcc -o both t1.o
Undefined symbols:
  "_main", referenced from:
     start in crt1.10.6.o
ld: symbol(s) not found
collect2: ld returned 1 exit status
```

Multi-file development: example

```
fawcett:330 childs$ cat t1.c
int doubler(int x)
{
   return 2*x;
}
fawcett:330 childs$ cat t2.c
int main()
{
    return doubler(5);
}
fawcett:330 childs$ gcc -c t1
fawcett:330 childs$ gcc -c t2
fawcett:330 childs$ gcc -o bo
fawcett:330 childs$ ./both
fawcett:330 childs$ echo $?
10
```

```
fawcett:330 childs$ gcc -o both t1.o t2.o
fawcett:330 childs$
```

Linker order matters for some linkers (not Macs). Some linkers need the .o with "main" first and then extract the symbols they need as they go.
Other linkers make multiple passes.

Libraries

- Library: collection of "implementations" (functions!) with a well defined interface
- Interface comes through "header" files.
- In C, header files contain functions and variables.
  - Accessed through "#include <file.h>"

Includes and Libraries

- gcc support for libraries
  - "-I": path to headers for library
    - when you say "#include <file.h>, then it looks for file.h in the directories -I points at
  - "-L": path to library location
  - "-lname": link in library libname

Making a static library

multiplier — bash — 80×24

```
C02LN00GFD58:multiplier hank$ cat multiplier.h # here's the header file
int doubler(int);
int tripler(int);
C02LN00GFD58:multiplier hank$ cat doubler.c # here's one of the c files
int doubler(int x) {return 2*x;}
C02LN00GFD58:multiplier hank$ cat tripler.c # here's the other c files
int tripler(int x) {return 3*x;}
C02LN00GFD58:multiplier hank$ gcc -c doubler.c # make an object file
C02LN00GFD58:multiplier hank$ ls doubler.o # we now have a .o
doubler.o
C02LN00GFD58:multiplier hank$ gcc -c tripler.c
C02LN00GFD58:multiplier hank$ ar r multiplier.a doubler.o tripler.o
C02LN00GFD58:multiplier hank$
```

(should have called this libmultiplier.a)

Note the '#' is the comment character

Typical library installations

- Convention
  - Header files are placed in “include” directory
  - Library files are placed in “lib” directory

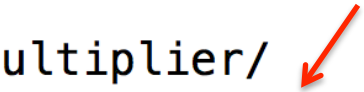
- Many standard libraries are installed in /usr
  - /usr/include
  - /usr/lib
- Compilers automatically look in /usr/include and /usr/lib (and other places)

Installing the library

multiplier — bash — 80×24

```
C02LN00GFD58:multiplier hank$ mkdir ~/multiplier
C02LN00GFD58:multiplier hank$ mkdir ~/multiplier/include
C02LN00GFD58:multiplier hank$ cp multiplier.h ~/multiplier/include/
C02LN00GFD58:multiplier hank$ mkdir ~/multiplier/lib
C02LN00GFD58:multiplier hank$ cp
doubler.c     multiplier.a  tripler.c
doubler.o     multiplier.h  tripler.o
C02LN00GFD58:multiplier hank$ cp multiplier.a ~/multiplier/
C02LN00GFD58:multiplier hank$ mv multiplier.a libmultiplier.a
C02LN00GFD58:multiplier hank$ cp libmultiplier.a ~/multiplier/lib/
C02LN00GFD58:multiplier hank$
```

(fixing my mistake)



"mv": unix command for renaming a file

Example: compiling with a library

CIS330 — bash — 80×24

```
C02LN00GFD58:CIS330 hank$ cat t.c
#include <multiplier.h>
#include <stdio.h>
int main()
{
   printf("Twice 6 is %d, triple 6 is %d\n", doubler(6), tripler(6));
}
C02LN00GFD58:CIS330 hank$ gcc -o mult_example t.c -I/Users/hank/multiplier/inclu
de -L/Users/hank/multiplier/lib -lmultiplier
C02LN00GFD58:CIS330 hank$ ./mult_example
Twice 6 is 12, triple 6 is 18
C02LN00GFD58:CIS330 hank$
```

- gcc support for libraries
  - "-I": path to headers for library
  - "-L": path to library location
  - "-lname": link in library libname

Makefiles

- There is a Unix command called "make"
- make takes an input file called a "Makefile"
- A Makefile allows you to specify rules
  - "if timestamp of A, B, or C is newer than D, then carry out this action" (to make a new version of D)
- make's functionality is broader than just compiling things, but it is mostly used for computation

Basic idea: all details for compilation are captured in a configuration file … you just invoke "make" from a shell

Makefile syntax

- Makefiles are set up as a series of rules
- Rules have the format:

  target: dependencies

  [tab] system command

Makefile example: multiplier lib

code — bash — 80×26

```
C02LN00GFD58:code hank$ cat Makefile
lib: doubler.o tripler.o
        ar r libmultiplier.a doubler.o tripler.o
        cp libmultiplier.a ~/multiplier/lib
        cp multiplier.h ~/multiplier/include

doubler.o: doubler.c
        gcc -c doubler.c

tripler.o: tripler.c
        gcc -c tripler.c
C02LN00GFD58:code hank$
C02LN00GFD58:code hank$
C02LN00GFD58:code hank$ make
ar r libmultiplier.a doubler.o tripler.o
cp libmultiplier.a ~/multiplier/lib
cp multiplier.h ~/multiplier/include
C02LN00GFD58:code hank$
C02LN00GFD58:code hank$ touch doubler.c
C02LN00GFD58:code hank$ make
gcc -c doubler.c
ar r libmultiplier.a doubler.o tripler.o
cp libmultiplier.a ~/multiplier/lib
cp multiplier.h ~/multiplier/include
C02LN00GFD58:code hank$
```

Outline

- Review
- Project 1C Overview
- Tar
- Memory Errors
- Project 4A
- Background Info for 4A

CIS 330: Project #1C
Assigned: April 7th, 2016
Due April ~~12th~~, 2016 ← Due April 14th
(which means submitted by 6am on April 13th, 2016)
Worth 2% of your grade

Assignment: Download the file "Proj1C.tar". This file contains a C-based project. You will build a Makefile for the project, and also extend the project.

Project 1C

== Build a Makefile for math330 ==

Your Makefile should:

(1) create an include directory
(2) copy the Header file to the include directory
(3) create a lib directory
(4) compile the .c files in trig and exp as object files (.o's)
(5) make a library
(6) install the library to the lib directory
(7) compile the "cli" program against the include and library directory

== Extend the math330 library ==

You should:

(1) add 3 new functions: arccos, arcsin, and arctan (each in their own file)
(2) Extend the "cli" program to support these functions
(3) Extend your Makefile to support the new functions

Outline

- Review
- Project 1C Overview
- Tar
- Memory Errors
- Project 4A
- Background Info for 4A

Unix command: tar

- Anyone know what tar stands for?

tar = tape archiver

IBM tape library

Unix command: tar

- Problem: you have many files and you want to…
  - move them to another machine
  - give a copy to a friend
  - etc.
- Tar: take many files and make one file
  - Originally so one file can be written to tape drive
- Serves same purpose as “.zip” files.

Unix command: tar

- tar cvf 330.tar file1 file2 file3
  - puts 3 files (file1, file2, file3) into a new file called 330.tar
- scp 330.tar @ix:~
- ssh ix
- tar xvf 330.tar
- ls

  file1 file2 file

Outline

- Review
- Project 1C Overview
- Tar
- Memory Errors
- Project 4A
- Background Info for 4A

Memory Errors

- Array bounds read

```
int main()
{
    int var;
    int arr[3] = { 0, 1, 2 };
    var=arr[3];
}
```

- Array bounds write

```
int main()
{
    int var = 2;
    int arr[3];
    arr[3]=var;
}
```

Memory Errors

Vocabulary: “dangling pointer”: pointer that points to memory that has already been freed.

```
int main()
{
    int *var = malloc(sizeof(int)*2);
    var[0] = 0;
    var[1] = 2;
    free(var);
    var[0] = var[1];
}
```

When does this happen in real-world scenarios?

Memory Errors

- Freeing unallocated memory

```
int main()
{
    int *var = malloc(sizeof(int)*2);
    var[0] = 0;
    var[1] = 2;
    free(var);
    free(var);
}
```

When does this happen in real-world scenarios?

Memory Errors

- Freeing non-heap memory

```
int main()
{
    int var[2]
    var[0] = 0;
    var[1] = 2;
    free(var);
}
```

When does this happen in real-world scenarios?

Memory Errors

- NULL pointer read / write

```
int main()
{
    char *str = NULL;
    printf(str);
    str[0] = 'H';
}
```

- NULL is never a valid location to read from or write to, and accessing them results in a "segmentation fault"
  - …. remember those memory segments?

When does this happen in real-world scenarios?

Memory Errors

- Uninitialized memory read

```
int main()
{
    int *arr = malloc(sizeof(int)*10);
    int V2=arr[3];
}
```

When does this happen in real-world scenarios?

Memory error in action

```
fawcett:error childs$ cat t.c
#include <stdio.h>

int main()
{
    int *X = NULL;
    printf("X is %d\n", *X);
}
fawcett:error childs$ gcc t.c
fawcett:error childs$ ./a.out
Segmentation fault
fawcett:error childs$
```

Outline

- Review
- Project 1C Overview
- Tar
- Memory Errors
- Project 4A
- Background Info for 4A

Project 4A

- Posted now
- You will practice debugging & using a debugger
  - There are 3 programs you need to debug
    - In this case, “debug” means identify the bug
      - Does not mean fix the bug
    - Can use gdb or lldb
    - May want to run on ix
- Worksheet due in class next week

Outline

- Review
- Project 1C Overview
- Tar
- Memory Errors
- Project 4A
- Background Info for 4A

ASCII Character Set

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

There have been various extensions to ASCII …
now more than 128 characters
Many special characters are handled outside this convention

image source: granneman.com

signed vs unsigned chars

- signed char (“char”):
  - valid values: -128 to 127
  - size: 1 byte
  - used to represent characters with ASCII
    - values -128 to -1 are not valid
- unsigned char:
  - valid values: 0 to 255
  - size: 1 byte
  - used to represent data

character strings

- A character “string” is:
  - an array of type “char”
  - that is terminated by the NULL character
- Example:

  char str[12] = “hello world”;
  - str[11] = ‘\0’ (the compiler did this automatically)
- The C library has multiple functions for handling strings

Character strings example

```
128-223-223-72-wireless:330 hank$ cat string.c
#include <stdio.h>

int main()
{
    char str[12] = "hello world";
    char *str2 = str+6;

    printf("str is \"%s\" and str2 is \"%s\"\n",
           str, str2);

    str[5] = '\0';

    printf("Now str is \"%s\" and str2 is \"%s\"\n",
           str, str2);
}
128-223-223-72-wireless:330 hank$ gcc string.c
128-223-223-72-wireless:330 hank$ ./a.out
str is "hello world" and str2 is "world"
Now str is "hello" and str2 is "world"
```

Useful C library string functions

- strcpy: string copy
- strncpy: string copy, but just first N characters
- strlen: length of a string

```
128-223-223-72-wireless:330 hank$ cat strcpy.c
#include <string.h>
#include <stdio.h>

int main()
{
    char str[12] = "hello world";
    char str2[6], str3[7];
    strcpy(str2, str+strlen("hello "));
    strncpy(str3, str, strlen("hello "));
    printf("%s,%s\n", str2, str3);
}
128-223-223-72-wireless:330 hank$ gcc strcpy.c
128-223-223-72-wireless:330 hank$ ./a.out
world,hello
```

Useful C library string functions

- strcpy: string copy
- strncpy: string copy, but just first N characters
- strlen: length of a string

```
128-223-223-72-wireless:330 hank$ cat strcpy.c
#include <string.h>
#include <stdio.h>

int main()
{
    char str[12] = "hello world";
    char str2[7], str3[6];
    strcpy(str2, str+strlen("hello "));
    strncpy(str3, str, strlen("hello "));
    printf("%s,%s\n", str2, str3);
}
128-223-223-72-wireless:330 hank$ gcc strcpy.c
128-223-223-72-wireless:330 hank$ ./a.out
world,hello world
```

What happened here?

More useful C library string functions

Functions

Copying:

memcpy	Copy block of memory (function)
memmove	Move block of memory (function)
strcpy	Copy string (function)
strncpy	Copy characters from string (function)

Concatenation:

strcat	Concatenate strings (function)
strncat	Append characters from string (function)

Comparison:

memcmp	Compare two blocks of memory (function
strcmp	Compare two strings (function)
strcoll	Compare two strings using locale (functior
strncmp	Compare characters of two strings (functi
strxfrm	Transform string using locale (function)

Searching:

memchr	Locate character in block of memory (function)
strchr	Locate first occurrence of character in string (fu
strcspn	Get span until character in string (function)
strpbrk	Locate characters in string (function)
strrchr	Locate last occurrence of character in string (fu
strspn	Get span of character set in string (function)
strstr	Locate substring (function)
strtok	Split string into tokens (function)

Other:

memset	Fill block of memory (function)
strerror	Get pointer to error message string (function)
strlen	Get string length (function)

Macros

NULL	Null pointer (macro)

Types

size_t	Unsigned integral type (type)

source: cplusplus.com

memcpy

```
MEMCPY(3)                 BSD Library Functions Manual                 MEMCPY(3)

NAME
     memcpy -- copy memory area

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <string.h>

     void *
     memcpy(void *restrict dst, const void *restrict src, size_t n);

DESCRIPTION
     The memcpy() function copies n bytes from memory area src to memory area
     dst.  If dst and src overlap, behavior is undefined.  Applications in
     which dst and src might overlap should use memmove(3) instead.

RETURN VALUES
     The memcpy() function returns the original value of dst.
```

I mostly use C++, and I still use memcpy all the time

sscanf

- like printf, but it parses from a string

  sscanf(str, "%s\n%d %d\n%d\n", magicNum,
  &width, &height, &maxval);

  on:

  str="P6\n1000 1000\n255\n";

  gives:

  magicNum = "P6", width = 1000,

  height = 1000, maxval = 255

if-then-else

```
int val = (X < 2 ? X : 2);
```

←→

```
if (X<2)
{
   val = X;
}
else
{
   val = 2;
}
```