ECE 590.01
C++ Programming, Data structures, and Algorithms

C
Admin

• Groups:
  • Tell me who you are with
  • Once done, we’ll setup all svn repos at once
  • Please include netid when you send

• Recitation this week:
  • How to use svn
  • gcc
  • ssh
  • emacs
  • make
  • etc..
Everyone’s favorite first program

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

int main (void) {
    printf("Hello World\n");
    return EXIT_SUCCESS;
}
```

```
$ gcc -pedantic --std=gnu99 -Wall -Werror hello.c
$ ./a.out
Hello World
$
```

CS 250
Everyone’s favorite first program

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

int main (void) {
    printf("Hello World\n");
    return EXIT_SUCCESS;
}
```

What do these do?
C compilation

- .h files
- .c file
- Libraries
- Startup + exit code
- Preprocessor (cpp)
- Intermediate File
- Compiler (cc)
- Assembly (.S)
- Object File (.o)
- Assembler (as)
- Other Object Files
- Linker (ld)
- Executable (a.out)
What’s in a header?

• Preprocessor definition:
  • `#define EXIT_SUCCESS 0`

• Function *prototypes*
  • Declare the argument/return types
  • Example:
    ```
    int factorial(int x);
    ```
    Note: ends in semi-colon.

• External variable declarations (later)
• Type declarations (also later)
A first approximation:

/* maybe this is the prototype from stdio?*/
void printf(string x);

int main (void) {
    printf(“Hello World\n”);
    return 0; /* Replace #define’ed symbol */
}
Close, but…

• C does not have a “string” type
  • char * represents a sequence of characters
  • const char * represents a sequence of characters where you can’t change
    the characters in it (this is the type of string literals)
  • We’ll learn about pointers shortly…

• printf actually
  • Returns int (number of characters printed)
  • Takes a variable number of arguments:
    int printf(const char * format, …);
A variable number of arguments?

- First arg: format specifier string
  - Contains % directives (%d, %s, ...)
- Later args: values to replace % directives
- Example:

  ```c
  int x = 3;
  const char * s = "some string";
  printf("x is %d and s is %s \n", x, s);
  ```
Basically right:

```c
int printf(const char * format, ...);

int main (void) {
    printf(“Hello World\n”);
    return 0;
}

(Actually, lots of other stuff defined in those headers)
```
Two types of \#include

- \#include <stdio.h>
  - Use angle brackets for system files
- \#include "myheader.h"
  - Use quotes for header files in the local directory (i.e. that you wrote/were provided)
Basic syntax: just like Java

• This could be C or Java:

```c
int i;
int counter = 0;
for ( i = 0; i < 10 ; i ++) {
    if (x[i] < y[i]) {
        counter++;
    }
}
return counter;
```
Caveat:

- What about

```c
for (int i = 0; i < 10 ; i ++) {
    ...
}
```

Error: ‘for’ loop initial declaration used outside C99 mode

Solution: compile with –std=gnu99

Also, gives you // -style comments
Proper organization of C program

• When writing a C program
  • Make as many .c files as you want/need
  • Split code up logically between files
  • Should only include .h files
    • Never .c files
    • (Sometimes do whacky things where you include other files, but generally only .h files)
  • .h files: interface
  • .c files: implementation
C compilation refresher

- `.h files`
- Preprocessor (.cpp)
- Intermediate File
- Compiler (.cc)
- Assembly (.S)
- `.c file`
- Object File (.o)
- Assembler (as)
- Libraries
- Linker (.ld)
- Other Object Files
- Startup + exit code
- Executable (a.out)
What’s in a .o file?

- Global variables
- “Mostly” assembled code
  - Jump/call targets unresolved
    - call factorial
  - Global variable addresses unknown
    - Ld (x) => $r9
- Need to fix these to make a binary
The linker’s job

• “Glue” together object/library code
• Resolve symbols => addresses
  • call fact ⇒ call 0x40013770
Compiling multiple .c files

- Multiple .c files? Two options
  - List all .c files on the gcc command line
  - Gcc will internally compile each .c to a .o
  - Then link them all together
  - (Then throw away the .o files)
  - Big project: may take a very long time
    - But then again...
Long compilation = more free time?

- Maybe long compilation times are a good thing?
  - Probably not

Image from xkcd.com/303
Compiling multiple .c files

• Multiple .c files? Two options
  • List all .c files on the gcc command line
  • Gcc will internally compile each .c to a .o
  • Then link them all together
  • (Then throw away the .o files)
  • Big project: may take a very long time

• Option 2:
  • Compile each .c file to a .o file (give gcc the –c option)
    • Says “stop at .o, don’t try to link)
  • Re-compile only what changed...
    • Or depended on a header file that changed
  • Ok, so that’s faster... but I have to keep track of all that?
Make to the rescue

• Keeping track of what to re-compile would be painful
  • And error-prone: we don’t like error-prone

• People realized this a long time ago...
  • And that it would be easy/nice to make a tool to manage this

• Make:
  • Examines dependences, re-builds only what needs it
  • Reads Makefile
    • Specifies dependences
    • And commands to remake something
    • Will recursively make what is needed to build a target
  • More in recitation Friday
In Java...

- `int` is a 4 byte signed integer
- `short` is 2 byte signed integer
- `byte` is a 1 byte signed integer
- `char` is a 2 byte unsigned unicode char
- `float` is an IEEE single precision number
- `etc..`
In C...

- how many bytes is int?
- short?
- char?
- (byte doesn’t exist)
In C...

- how many bytes is int: sizeof(int)
- short: sizeof(short)
- char: sizeof(char)
- Types can be “unsigned”
- sizeof(expr) works too
  
  ```c
  int x;
  sizeof(x); /* same as sizeof(int) */
  ```
Let’s do a little Java...

```java
public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a =" + a + " b = " + b);
    }
}
```

• What does this print? Why?
Let’s do a little Java...

```java
public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a =" + a + " b = " + b);
    }
}
```

- What does this print? Why?

CS 250
Let’s do a little Java...

public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a =" + a + " b = " + b);
    }
}

• What does this print? Why?
Let’s do a little Java...

```java
public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a = " + a + " b = " + b);
    }
}
```

• What does this print? Why?
Let’s do a little Java...

public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a =" + a + " b = " + b);
    }
}

• What does this print? Why?
public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a =" + a + " b = " + b);
    }
}

• What does this print? Why?
public class Example {
    public static void swap (int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }
    public static void main (String[] args) {
        int a = 42;
        int b = 100;
        swap (a, b);
        System.out.println("a =" + a + " b = " + b);
    }
}

- What does this print? Why?
C: also passes by value

```c
void swap (int x, int y) {
    int temp = x;
    x = y;
    y = temp;
}

int main (void) {
    int a = 42;
    int b = 100;
    swap (a, b);
    printf("a = %d b = %d\n",a,b);
    return EXIT_SUCCESS;
}

• Same code but in C, same behavior

CS 250
Let’s do some different Java...

```java
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a =" + a.data + " b = " + b.data);
    }
}
```

- What does this print? Why?
Let’s do some different Java...

```java
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a = " + a.data + " b = " + b.data);
    }
}
```

• What does this print? Why?

CS 250
Let’s do some different Java...

```java
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a =" + a.data + " b =" + b.data);
    }
}
```

- What does this print? Why?

CS 250
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a = " + a.data + " b = " + b.data);
    }
}

• What does this print? Why?
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a = " + a.data +
                        " b = " + b.data);
    }
}
• What does this print? Why?
Let’s do some different Java...

```java
class Ex2 {  
    int data;  
    public Ex2 (int d) { data = d; }  
    public static void swap (Ex2 x, Ex2 y) {  
        int temp = x.data;  
        x.data = y.data;  
        y.data = temp;  
    }  
    public static void main (String[] args) {  
        Example a = new Example (42);  
        Example b = new Example (100);  
        swap (a, b);  
        System.out.println("a = " + a.data +  
            " b = " + b.data);  
    }  
}
```

What does this print? Why?
Let’s do some different Java...

```java
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a = " + a.data + ", b = " + b.data);
    }
}
```

• What does this print? Why?

CS 250
Let’s do some different Java...

```java
public class Ex2 {
    int data;
    public Ex2 (int d) { data = d; }
    public static void swap (Ex2 x, Ex2 y) {
        int temp = x.data;
        x.data = y.data;
        y.data = temp;
    }
    public static void main (String[] args) {
        Example a = new Example (42);
        Example b = new Example (100);
        swap (a, b);
        System.out.println("a =" + a.data +
                         " b = " + b.data);
    }
}
```

- What does this print? Why?
References and Pointers

- Java has **references**:
  - Any variable of object type is a reference
  - Point at objects (which are all in the heap)
    - Under the hood: is the memory address of the object
  - Cannot explicitly manipulate them (*e.g.*, add 4)
References and Pointers

• Java has **references**:
  • Any variable of object type is a reference
  • Point at objects (which are all in the heap)
    • Under the hood: is the memory address of the object
  • Cannot explicitly manipulate them (e.g., add 4)

• Some languages (C, assembly) have explicit **pointers**:
  • Hold the memory address of something
  • Can explicitly compute on them
  • Can **de-reference** the pointer (*ptr) to get thing-pointed-to
  • Can take the **address-of** (&x) to get something’s address
  • Can do very **unsafe** things, shoot yourself in the foot
Pointers

• For any type T, T* is a “pointer to a T”
  • int * = pointer to an int
  • int ** = pointer to a pointer to an int

• Specifies where in memory something is

• Two operators:
  • Address-of (&): “give me a pointer to”
    • If expr has type T, then &expr has type T*
    • Not all things can have their address taken: &42 does not make sense!
  • Dereference (*): “give me what is pointed to by”
    • If expr has type T*, then *expr has type T
    • If expr is not a pointer, *expr is an error
Pointers

```c
int x = 42;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1234000C</td>
<td>42</td>
<td>x</td>
</tr>
<tr>
<td>0x12340008</td>
<td>0x1234000C</td>
<td></td>
</tr>
<tr>
<td>0x12340004</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables reside in memory (*)

(*) Some get register allocated, but we won’t worry about that now
Pointers

Pointers are variables too (they also reside in memory)
Their value is the address of another variable
Address-of operator (unary-&) 
Gives the address where a variable is (rather than its value)
### Pointers

```c
int x = 42;
int *p = & x;
int y = *p;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1234000C</td>
<td>42</td>
<td>x</td>
</tr>
<tr>
<td>0x12340008</td>
<td>0x1234000C</td>
<td>p</td>
</tr>
<tr>
<td>0x12340004</td>
<td>42</td>
<td>y</td>
</tr>
</tbody>
</table>

**Dereference operator (*)**

“Follow the pointer”

Gives what is pointed to, instead of pointer value.
Pointers

int x = 42;
int *p = & x;
int y = *p;
*p = 99;

Dereference operator (*)
“Follow the pointer”
Gives what is pointed to, instead of pointer value.
Can be used to change values also

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1234000C</td>
<td>99</td>
<td>x</td>
</tr>
<tr>
<td>0x12340008</td>
<td>0x1234000C</td>
<td>p</td>
</tr>
<tr>
<td>0x12340004</td>
<td>42</td>
<td>y</td>
</tr>
</tbody>
</table>
Pointers

int x = 42;
int *p = &x;
int y = *p;
*p = 99;
p = &y;

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1234000C</td>
<td>99</td>
<td>x</td>
</tr>
<tr>
<td>0x12340008</td>
<td>0x12340004</td>
<td>p</td>
</tr>
<tr>
<td>0x12340004</td>
<td>42</td>
<td>y</td>
</tr>
</tbody>
</table>

Can change what p points to by assigning to p
Pointers

int x = 42;
int *p = &x;
int y = *p;
*p = 99;
p = &y;
p = (int *) 7;

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1234000C</td>
<td>99</td>
<td>x</td>
</tr>
<tr>
<td>0x12340008</td>
<td>7</td>
<td>p</td>
</tr>
<tr>
<td>0x12340004</td>
<td>42</td>
<td>y</td>
</tr>
</tbody>
</table>

Can change what p points to by assigning to p
Could try to assign a number to p, but asking for trouble  
(Doing *p will access memory address 7, not a valid range of memory, program will crash)
Memory Layout + Seg faults

- Picture of Memory:
  - Stack:
    - Local variables
    - Return addresses
    - Parameters
  - Heap:
    - Dynamically allocated data (new/malloc)
  - Data:
    - Global variables, string constants
  - Text:
    - Code
- Accesses in invalid regions
  - Crash the program
  - Segmentation Fault
A word about pointers

• For some reason, people find pointers hard
• ...master them and they will be incredibly useful

• Conceptually:
  • Value of a pointer is an arrow pointing at something else
  • &x means “arrow pointing at x”
  • *p means “follow the arrow that is p”
int x = 307;
int y = 4115;
int * px = &x;
int * py = &y;
int ** ppx = &px;

What are the types and values of *px, *py, *ppx, and **ppx?
Pointers and Arrays

• In C, arrays are just pointers to their data

```c
int myArray[4] = {42, 63, 55, 12};
int * p = myArray;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF000C</td>
<td>12</td>
</tr>
<tr>
<td>0xFFFF0008</td>
<td>55</td>
</tr>
<tr>
<td>0xFFFF0004</td>
<td>63</td>
</tr>
<tr>
<td>0xFFFF0000</td>
<td>42</td>
</tr>
<tr>
<td>0xFFFEFFC</td>
<td>0xFFFF0000</td>
</tr>
</tbody>
</table>
Pointers and Arrays

- In C, arrays are just pointers to their data

```c
int myArray[4] = {42, 63, 55, 12};
int * p = myArray;
myArray[1] = 0;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF000C</td>
<td>12</td>
</tr>
<tr>
<td>0xFFFF0008</td>
<td>55</td>
</tr>
<tr>
<td>0xFFFF0004</td>
<td>0</td>
</tr>
<tr>
<td>0xFFFF0000</td>
<td>42</td>
</tr>
<tr>
<td>0xFFFEFFC</td>
<td>0xFFFFF0000</td>
</tr>
</tbody>
</table>

CS 250
Pointers and Arrays

- In C, arrays are just pointers to their data

```c
int myArray[4] = {42, 63, 55, 12};
int * p = myArray;
myArray[1] = 0;
*p = 99;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF000C</td>
<td>12</td>
</tr>
<tr>
<td>0xFFFF0008</td>
<td>55</td>
</tr>
<tr>
<td>0xFFFF0004</td>
<td>0</td>
</tr>
<tr>
<td>0xFFFF0000</td>
<td>99</td>
</tr>
<tr>
<td>0xFFFEFFC</td>
<td>0xFFFF0000</td>
</tr>
</tbody>
</table>
Pointers and Arrays

- In C, arrays are just pointers to their data
- We can also do math on pointers, and treat them like arrays

```c
int myArray[4] = {42, 63, 55, 12};
int * p = myArray;
myArray[1] = 0;
*p = 99;
p[2]--;```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF000C</td>
<td>12</td>
</tr>
<tr>
<td>0xFFFF0008</td>
<td>54</td>
</tr>
<tr>
<td>0xFFFF0004</td>
<td>0</td>
</tr>
<tr>
<td>0xFFFF0000</td>
<td>99</td>
</tr>
<tr>
<td>0xFFFEFFFC</td>
<td>0x00FF0000</td>
</tr>
</tbody>
</table>
Pointers and Arrays

• In C, arrays are just pointers to their data
• We can also do math on pointers, and treat them like arrays

```c
int myArray[4] = {42, 63, 55, 12};
int * p = myArray;
myArray[1] = 0;
*p = 99;
p[2]--;  
p = p + 1;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF0000</td>
<td>99</td>
</tr>
<tr>
<td>0xFFFF0004</td>
<td>0</td>
</tr>
<tr>
<td>0xFFFF0008</td>
<td>54</td>
</tr>
<tr>
<td>0xFFFF000C</td>
<td>12</td>
</tr>
<tr>
<td>0xFFFFEFFF</td>
<td>0xFFFFF0004</td>
</tr>
</tbody>
</table>
Pointers and Arrays

• In C, arrays are just pointers to their data
• We can also do math on pointers, and treat them like arrays

```c
int myArray[4] = {42, 63, 55, 12};
int * p = myArray;
myArray[1] = 0;
p[2]--;  
p = p + 1;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF000C</td>
<td>12</td>
</tr>
<tr>
<td>0xFFFF0008</td>
<td>54</td>
</tr>
<tr>
<td>0xFFFF0004</td>
<td>0</td>
</tr>
<tr>
<td>0xFFFF0000</td>
<td>99</td>
</tr>
<tr>
<td>0xFFFEFFFC</td>
<td>0xFFFF0004</td>
</tr>
</tbody>
</table>

Notice: `p = p + 1` changed the value of `p` by 4

Why?
Pointer math

• When you do \( p + 1 \)
  • \( p \) has type int *
  • 1 has type int
  • In our example:
    • `sizeof(int) = 4`
  • Actually adding 1 makes no sense
    • 0xFFFF0001 points at part of one int... and part of the next
    • Its an un-aligned pointer
      • Many architectures do not allow un-aligned memory access
      • Some just do it very slowly...
  • We really mean “plus one int”
    • C automatically multiplies the int by `sizeof(int)`
    • Generally \( p + n \) turns into \( p + n * \text{sizeof(*p)} \)

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF0000C</td>
<td>12</td>
</tr>
<tr>
<td>0xFFFF0008</td>
<td>54</td>
</tr>
<tr>
<td>0xFFFF0004</td>
<td>0</td>
</tr>
<tr>
<td>0xFFFF0000</td>
<td>99</td>
</tr>
<tr>
<td>0xFFFEFFC</td>
<td>0xFFFF0004</td>
</tr>
</tbody>
</table>
Pointers, arrays etc..

- A few rules about the interchangability of pointers and arrays
  
  \[ p[i] = *(p + i) \]
  
  \[ &p[i] = (p+i) \]
Pointers, arrays etc..

- A few rules about the interchangability of pointers and arrays
  
  \[
  p[i] = *(p + i) \\
  &p[i] = (p+i)
  \]

Stupid C trick:

\[
p[i] = *(p + i) = *(i + p) = i[p]
\]

So... I can write

\[
3[myArray] \\
???
\]
• A few rules about the interchangability of pointers and arrays
  
  \[ p[i] = *(p + i) \]
  \[ &p[i] = (p+i) \]

Stupid C trick:

\[ p[i] = *(p + i) = *(i + p) = i[p] \]

So... I can write

3[myArray]

???

Yes, yes you can. Give it a try if you don’t believe me.
Strings: really just pointers

- No string type
- `char *`: pointer to a sequence of chars
- Char = character
  - Actually come in signed or unsigned
- "String’s end with ‘\0’ (null-terminator)
  - Don’t confuse the char ‘\0’ with a NULL pointer
- String literals are read only
  - “Hello” has type `const char *`
Strings, string literals etc

- String literals make an array and give you a pointer to it.
- Null terminator ('\0') automatically included
- Is in read only memory
  - Correct type: const char *
  - Trying to write to it will segfault

```c
const char * s = "xyz";
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00400003</td>
<td>'0'</td>
</tr>
<tr>
<td>0x00400002</td>
<td>'z'</td>
</tr>
<tr>
<td>0x00400001</td>
<td>'y'</td>
</tr>
<tr>
<td>0x00400000</td>
<td>'x'</td>
</tr>
</tbody>
</table>
"Hello"+3 = "lo"?

char * s = "Hello\n"
char * s2 = s + 3;
printf("%s", s2);
String concatenation

• In Java + on strings does concatenation
• In C, it just does pointer arithmetic
  • Use `strncat` or `snprintf`
  • (Typically do not use `strcat` or `sprintf`!)

• Speaking of library functions
  • Quick aside: standard library reference
    • Man pages
$ man strcat

NAME
   strcat, strncat - concatenate two strings

SYNOPSIS
   #include <string.h>

   char *strcat(char *dest, const char *src);

   char *strncat(char *dest, const char *src, size_t n);

DESCRIPTION
   The strcat() function appends the src string to the dest string, overwriting the terminating null byte ('\0') at the end of dest, and then adds a terminating null byte. The strings may not overlap, and the dest string must have enough space for the result.

...... (continues)......
$ man strcat

NAME
strcat, strncat - concatenate two strings

SYNOPSIS
#include <string.h>

char *strcat(char *dest, const char *src);

char *strncat(char *dest, const char *src, size_t n);

DESCRIPTION
The strcat() function appends the src string to the dest string, overwriting the terminating null byte ('\0') at the end of dest, and then adds a terminating null byte. The strings may not overlap, and the dest string must have enough space for the result.

...... (continues)......
$ man strcat

NAME
   strcat, strncat - concatenate two strings

SYNOPSIS
   #include <string.h>

   char *strcat(char *dest, const char *src);

   char *strncat(char *dest, const char *src, size_t n);

DESCRIPTION
   The strcat() function appends the src string to the dest string, overwriting the termin-
   nating null byte ('\0') at the end of dest, and then adds a terminating null byte. The
   strings may not overlap, and the dest string must have enough space for the result.

...... (continues)......
$ man strcat

NAME
    strcat, strncat - concatenate two strings

SYNOPSIS
    #include <string.h>
    char *strcat(char *dest, const char *src);
    char *strncat(char *dest, const char *src, size_t n);

DESCRIPTION
    The strcat() function appends the src string to the dest string, overwriting the terminating null byte ('\0') at the end of dest, and then adds a terminating null byte. The strings may not overlap, and the dest string must have enough space for the result.

...... (continues)......
$ man strcat

NAME
strcat, strncat - concatenate two strings

SYNOPSIS
#include <string.h>

char *strcat(char *dest, const char *src);

char *strncat(char *dest, const char *src, size_t n);

DESCRIPTION
The strcat() function appends the src string to the dest string, overwriting the terminating null byte ("\0") at the end of dest, and then adds a terminating null byte. The strings may not overlap, and the dest string must have enough space for the result.

...... (continues)......
**RETURN VALUE**

The `strcat()` and `strncat()` functions return a pointer to the resulting string `dest`.

**CONFORMING TO**

SVr4, 4.3BSD, C89, C99.

**SEE ALSO**

`bcopy(3)`, `memccpy(3)`, `memcpy(3)`, `strcpy(3)`, `string(3)`, `strncpy(3)`, `wcsCat(3)`, `wcsncat(3)`

What does it return?
Will describe error case return values if any
Later in the man page

• (scroll down)

RETURN VALUE

The strcat() and strncat() functions return a pointer to the resulting string dest.

CONFORMING TO

SVr4, 4.3BSD, C89, C99.

SEE ALSO

bcopy(3), memccpy(3), memcpy(3), strcpy(3), string(3), strncpy(3), wcscat(3), wcsncat(3)

What versions of the C standard library is this in?
C99 is what we use (actually C99 w/ gnu extensions)
RETURN VALUE

The strcat() and strncat() functions return a pointer to the resulting string dest.

CONFORMING TO

SVr4, 4.3BSD, C89, C99.

SEE ALSO

bcopy(3), memccpy(3), memcpy(3), strcpy(3), string(3), strncpy(3), wcscat(3), wcsncat(3)

Related man pages
Mostly other functions, string is a list of string functions
Man pages: How to find it?

• Man page is great if you know what you want
  • “I need strncat, but don’t remember the order of the arguments”

• What if you don’t know what you want?
  • “I need a function that duplicates a string”

• man –k keyword
  • Do a keyword search on the man pages
    $ man -k duplicate
    ...
    strdup (3) - duplicate a string
    ...

• Something close? Look in “See Also”
Man pages: sections

• What is the (3) in all those names?
  • E.g., strdup(3)

• Man pages cover more than just C library, organized in sections
  1. Programs and shell commands
     • e.g., make, bash, gcc, ...
  2. System calls
     • open, read, write, dup, fork, exec,...
  3. Library calls
     • printf, stdrup, strncat, fgets,...

For the other sections, see man man
File IO

- Stdio provides a lot of IO functions (standard IO)
- Man pages you might be interested in:
  - `fopen`
  - `fprintf`
  - `fgetc`
  - `fgets`
  - Note use `fgets`, NOT `gets`
  - `fclose`

- Operate on type `FILE *`
  - You don’t need to know the details of `FILE`, just pass pointers to it around
  - Yay abstraction!
Next time

• Next time:
  • Dynamic allocation: malloc, re-alloc, free
  • Structs
  • Debugging and gdb