This is CS50.
quiz 0 details

wed oct 13, 1pm
see handout for locations
covers weeks 0 through 5
closed book
bring a 8.5” × 11”, 2-sided cheat sheet
75 minutes
15% of final grade
resources

old quizzes + solutions
lecture slides
lecture videos + transcripts
source code
scribe notes
section videos
pset specs
office hours
topics
review
Part 0

Scott Crouch
Binary Numbers

- Base-2 Representation
- The memory of your computer is contained in bits that are either 1 or 0
# Binary Numbers

<table>
<thead>
<tr>
<th>$2^7$</th>
<th>$2^6$</th>
<th>$2^5$</th>
<th>$2^4$</th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Maximum 8-digit binary value?

11111111 = 255 or $2^8 - 1$
Some Practice

What is 122 in Binary?

01111010

What is 00011001 in Decimal?

25
Binary Addition

$0 + 1 = 1$, $0 + 0 = 0$, $1 + 0 = 1$

$1 + 1 = 10$, but carry the 1

Example:

\[
\begin{array}{ccccccc}
 & & & 1 & 1 & 1 & 1 & 1 \\
\hline
0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
+ & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
\end{array}
\]
Hexadecimal

Base 16 with 16 distinct symbols

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

Each digit is a nibble or 4 bits

0001 = 0x1
1111 = 0xF
00011111 = 0x1F
10101111 = 0xAF
### ASCII

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
<td>NUL (null)</td>
<td>32</td>
<td>20</td>
<td>040</td>
<td> </td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td> </td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>SOH (start of heading)</td>
<td>33</td>
<td>21</td>
<td>041</td>
<td> </td>
<td> </td>
<td>65</td>
<td>41</td>
<td>101</td>
<td> </td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>002</td>
<td>STX (start of text)</td>
<td>34</td>
<td>22</td>
<td>042</td>
<td> </td>
<td> </td>
<td>66</td>
<td>42</td>
<td>102</td>
<td> </td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>003</td>
<td>ETX (end of text)</td>
<td>35</td>
<td>23</td>
<td>043</td>
<td> </td>
<td> </td>
<td>67</td>
<td>43</td>
<td>103</td>
<td> </td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>004</td>
<td>EOT (end of transmission)</td>
<td>36</td>
<td>24</td>
<td>044</td>
<td> </td>
<td> </td>
<td>68</td>
<td>44</td>
<td>104</td>
<td> </td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>005</td>
<td>ENQ (enquiry)</td>
<td>37</td>
<td>25</td>
<td>045</td>
<td> </td>
<td> </td>
<td>69</td>
<td>45</td>
<td>105</td>
<td> </td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>006</td>
<td>ACK (acknowledge)</td>
<td>38</td>
<td>26</td>
<td>046</td>
<td> </td>
<td> </td>
<td>70</td>
<td>46</td>
<td>106</td>
<td> </td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>007</td>
<td>BEL (bell)</td>
<td>39</td>
<td>27</td>
<td>047</td>
<td> </td>
<td> </td>
<td>71</td>
<td>47</td>
<td>107</td>
<td> </td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010</td>
<td>BS (backspace)</td>
<td>40</td>
<td>28</td>
<td>050</td>
<td> </td>
<td> </td>
<td>72</td>
<td>48</td>
<td>110</td>
<td> </td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>011</td>
<td>TAB (horizontal tab)</td>
<td>41</td>
<td>29</td>
<td>051</td>
<td> </td>
<td> </td>
<td>73</td>
<td>49</td>
<td>111</td>
<td> </td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>012</td>
<td>LF (NL line feed, new line)</td>
<td>42</td>
<td>2A</td>
<td>052</td>
<td> </td>
<td> </td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td> </td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>013</td>
<td>VT (vertical tab)</td>
<td>43</td>
<td>2B</td>
<td>053</td>
<td> </td>
<td> </td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td> </td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>014</td>
<td>FF (NP form feed, new page)</td>
<td>44</td>
<td>2C</td>
<td>054</td>
<td> </td>
<td> </td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td> </td>
<td>L</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>015</td>
<td>CR (carriage return)</td>
<td>45</td>
<td>2D</td>
<td>055</td>
<td> </td>
<td> </td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td> </td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>016</td>
<td>SO (shift out)</td>
<td>46</td>
<td>2E</td>
<td>056</td>
<td> </td>
<td> </td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td> </td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>017</td>
<td>SI (shift in)</td>
<td>47</td>
<td>2F</td>
<td>057</td>
<td> </td>
<td> </td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td> </td>
<td>O</td>
</tr>
<tr>
<td>16</td>
<td>G</td>
<td>020</td>
<td>DLE (data link escape)</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td> </td>
<td> </td>
<td>80</td>
<td>50</td>
<td>120</td>
<td> </td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>H</td>
<td>021</td>
<td>DC1 (device control 1)</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td> </td>
<td> </td>
<td>81</td>
<td>51</td>
<td>121</td>
<td> </td>
<td>Q</td>
</tr>
<tr>
<td>18</td>
<td>I</td>
<td>022</td>
<td>DC2 (device control 2)</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td> </td>
<td> </td>
<td>82</td>
<td>52</td>
<td>122</td>
<td> </td>
<td>R</td>
</tr>
<tr>
<td>19</td>
<td>J</td>
<td>023</td>
<td>DC3 (device control 3)</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td> </td>
<td> </td>
<td>83</td>
<td>53</td>
<td>123</td>
<td> </td>
<td>S</td>
</tr>
<tr>
<td>20</td>
<td>K</td>
<td>024</td>
<td>DC4 (device control 4)</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td> </td>
<td> </td>
<td>84</td>
<td>54</td>
<td>124</td>
<td> </td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>L</td>
<td>025</td>
<td>NAK (negative acknowledge)</td>
<td>53</td>
<td>35</td>
<td>065</td>
<td> </td>
<td> </td>
<td>85</td>
<td>55</td>
<td>125</td>
<td> </td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>M</td>
<td>026</td>
<td>SYN (synchronous idle)</td>
<td>54</td>
<td>36</td>
<td>066</td>
<td> </td>
<td> </td>
<td>86</td>
<td>56</td>
<td>126</td>
<td> </td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>N</td>
<td>027</td>
<td>ETB (end of trans. block)</td>
<td>55</td>
<td>37</td>
<td>067</td>
<td> </td>
<td> </td>
<td>87</td>
<td>57</td>
<td>127</td>
<td> </td>
<td>W</td>
</tr>
<tr>
<td>24</td>
<td>O</td>
<td>030</td>
<td>CAN (cancel)</td>
<td>56</td>
<td>38</td>
<td>070</td>
<td> </td>
<td> </td>
<td>88</td>
<td>58</td>
<td>130</td>
<td> </td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>P</td>
<td>031</td>
<td>EM (end of medium)</td>
<td>57</td>
<td>39</td>
<td>071</td>
<td> </td>
<td> </td>
<td>89</td>
<td>59</td>
<td>131</td>
<td> </td>
<td>Y</td>
</tr>
<tr>
<td>26</td>
<td>Q</td>
<td>032</td>
<td>SUB (substitute)</td>
<td>58</td>
<td>3A</td>
<td>072</td>
<td> </td>
<td> </td>
<td>90</td>
<td>5A</td>
<td>132</td>
<td> </td>
<td>Z</td>
</tr>
<tr>
<td>27</td>
<td>R</td>
<td>033</td>
<td>ESC (escape)</td>
<td>59</td>
<td>3B</td>
<td>073</td>
<td> </td>
<td> </td>
<td>91</td>
<td>5B</td>
<td>133</td>
<td> </td>
<td>[</td>
</tr>
<tr>
<td>28</td>
<td>S</td>
<td>034</td>
<td>FS (file separator)</td>
<td>60</td>
<td>3C</td>
<td>074</td>
<td> </td>
<td> </td>
<td>92</td>
<td>5C</td>
<td>134</td>
<td> </td>
<td>\</td>
</tr>
<tr>
<td>29</td>
<td>T</td>
<td>035</td>
<td>GS (group separator)</td>
<td>61</td>
<td>3D</td>
<td>075</td>
<td> </td>
<td> </td>
<td>93</td>
<td>5D</td>
<td>135</td>
<td> </td>
<td>{</td>
</tr>
<tr>
<td>30</td>
<td>U</td>
<td>036</td>
<td>RS (record separator)</td>
<td>62</td>
<td>3E</td>
<td>076</td>
<td> </td>
<td> </td>
<td>94</td>
<td>5E</td>
<td>136</td>
<td> </td>
<td>^</td>
</tr>
<tr>
<td>31</td>
<td>V</td>
<td>037</td>
<td>US (unit separator)</td>
<td>63</td>
<td>3F</td>
<td>077</td>
<td> </td>
<td> </td>
<td>95</td>
<td>5F</td>
<td>137</td>
<td> </td>
<td>`</td>
</tr>
</tbody>
</table>
ASCII Again

• Encoding scheme for characters
• For arithmetical operations, you can use the ASCII char.

//sets b to ‘B’
char b = ‘A’ + 1;

//sets e to 101
int e = ‘d’ + 1;
GCC and Compilers

GNU C Compiler (aka GNU Compiler Collection)

Compiling Commands:

gcc <program_name>.c
    produces a.out executable file

gcc -o <program_name> <program_name>.c
    produces an executable file with the name of your file
Common Compiling Errors and Warnings

undefined reference to function “GetString”

forgot to link in cs50 library (-lcs50)

implicit declaration of built in function ‘sqrt’

forgot to #include <math.h>

c control reaches end of non-void function

one of your non-void functions did not return a value.
Variables

Allow us to store information about the state of a program so we can access/change this information at a later time.

```c
int var1 = 5; // declares an integer with value 5
var1++; // increments var1
printf("%d", var1); // prints out 6
```
Be Careful!!

42 = int var;
Some types in C:

- **int**: 4 bytes goes from $-2^{31} \rightarrow 2^{31} - 1$
- **float**: 4 bytes (7-digit precision)
- **double**: 8 bytes (15-digit precision)
- **char**: 1 byte goes from $-2^7 \rightarrow 2^7 - 1$
- **long**: 4 bytes goes from $-2^{31} \rightarrow 2^{31} - 1$
- **long long**: 8 bytes goes from $-2^{63} \rightarrow 2^{63} - 1$

Signed vs. Unsigned?

Note: The sizes above are machine dependent, not C-Dependent, however these sizes are the most common.
Type Casting

Useful if you want to go between types:

Syntax:

```java
int oldNum = 42;
float newNum = (float) oldNum;
char c = (char) oldNum;
```
Conditionals

Based off Booleans or Predicates: A statement that returns true or false which must first be fulfilled for the code to executed.

Represented with if, else if and else statements.
int num = GetInt();

if (num > 0)
    printf("You entered a positive number!");
else if (num < 0)
    printf("You entered a negative number!");
else
    printf("You entered zero");
The Ternary Operator (aka The Sexy Statement)

Condense if and else into a 1 line statement!

Example:

```c++
int num = getInt();
string statement = (num < 0) ? "Error" : "Valid";
```

Syntax:

```c++
<variable_name> = (<condition>) ? <if true then> : <else then>;
```
For loops

for (<counter(s) initialization>; <condition(s)>; <change counter(s)>)
{
    //your code here
}

Example:

int i, j;

for (i = 0, j = 0; i < 3 && j < 10; i++, j+= 2) {
    printf("\ni:%d, j: %d", i, j);
}
While and Do-While Loops

while (<condition>)
{
    //do this
}

This loop checks then evaluates.

do {
    //do this
}

This loop evaluates then checks.

} while (<condition>)
Arrays

Simple data structure for storing objects of the same type.

Imagine them as lined up mailboxes, each with its own distinct number or shall we say index!
Declaring and Initializing Arrays

// declare an array of integers of length and fill it
int myArray[] = {4, 5, 6, 7};

// change the value in the 2\textsuperscript{nd} slot to be 3
myArray[2] = 3;
Using Loops with Arrays

Loops can be used to iterate through arrays!

```cpp
int myArray[4];

for (int i = 0; i < 4; i++)
    myArray[i] = i;
```
libraries

Standard Library
  printf
  ...
Math Library
  round
  ...

CS50 Library
  GetChar
  GetDouble
  GetFloat
  GetInt
  GetLongLong
  GetString
#include <cs50.h>
gcc foo.c -lcs50
functions
int main(void)
{
  <do stuff>
  return 0;
}
return-type

name([arg-type arg-name, ...])
{

    <do stuff>
    return value;

}
\[ f(x) = x^2 + 4x \]
f(x) = x^2 + 4x
f(2) = (2)^2 + 4(2)
f(2) = 4 + 8
f(2) = 12
int foo(int x)
{
    return x*x + 4*x;
}
command-line args
int main(int argc, char *argv[])
{
    // <do stuff>
    return 0;
}
./programname cmd line args
./programname cmd line args

argc =
./programname cmd line args

argc = 4
./programname cmd line args

argc = 4

argv[0] =
argv[1] =
argv[2] =
argv[3] =
./programname cmd line args

argc = 4

argv[0] = "../programname"
argv[1] = "cmd"
argv[2] = "line"
argv[3] = "args"
scope
// Swaps a and b. (lol jk)
void
swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
frames
int
bar(int i)
{
    return i + 1;
}

int
foo(int n)
{
    return bar(n) * 2;
}

int
main(void)
{
    int x = 5;
    x = foo(x);
    return 0;
}
int bar(int i)
{
    return i + 1;
}

int foo(int n)
{
    return bar(n) * 2;
}

int main(void)
{
    int x = 5;
    x = foo(x);
    return 0;
}
int
bar(int i)
{
    return i + 1;
}

int
foo(int n)
{
    return bar(n) * 2;
}

int
main(void)
{
    int x = 5;
    x = foo(x);
    return 0;
}
int
bar(int i)
{
    return i + 1;
}

int
foo(int n)
{
    return bar(n) * 2;
}

int
main(void)
{
    int x = 5;
    x = foo(x);
    return 0;
}
int
bar(int i)
{
    return i + 1;
}

int
foo(int n)
{
    return bar(n) * 2;
}

int
main(void)
{
    int x = 5;
    x = foo(x);
    return 0;
}
```c
int bar(int i)
{
    return i + 1;
}

int foo(int n)
{
    return bar(n) * 2;
}

int main(void)
{
    int x = 5;
    x = foo(x);
    return 0;
}
```
int
bar(int i)
{
    return i + 1;
}

int
foo(int n)
{
    return bar(n) * 2;
}

int
main(void)
{
    int x = 5;
    x = foo(x);
    return 0;
}
```c
int bar(int i)
{
    return i + 1;
}

int foo(int n)
{
    return bar(n) * 2;
}

int main(void)
{
    int x = 5;
    x = foo(x);
    return 0;
}
```
Part 2

Rose Cao
topics = topics -> next;

Hi! I’m Rose, for part 2. =) 

• Recursion
• Search
• Sort
• Asymptotic Notation
Recursive Functions
(as opposed to iterative)

- When a program repetitively calls on itself
- Performs a task by repeating smaller, similar tasks
- Ex: \(5! = 120\)
  - \(5 * 4! = 120\)
  - \(4 * 3! = 24\)
  - \(3 * 2! = 6\)
  - \(2 * 1! = 2\)
  - \(1 * 0! = 1\)
- Needs a base case to know when to stop!
A more interesting example:

Print the characters of a string.
(recursively, since you know the iterative version already)

```c
void print_chars(char str[], int spot) {
    // Base case: at end of string
    if (str[spot] == '\0')
        return;
    else {
        // Print the character
        printf("%c\n", str[spot]);

        // Call print_chars with next spot
        print_chars(str, spot + 1);
    }
}
```
How it happens:

void print_chars(char str[], int spot)
{
    // Base case: at end of string
    if (str[spot] == '\0')
        return;
    else
    {
        // Print the character
        printf("%c", str[spot]);
        // Call print_chars with next spot
        print_chars(str, spot + 1);
    }
}

Somewhere in main():
... print_chars(str, 0); ...
with str:

```
f t w /0
```

Printed:

```
ftw
```

Done with print_chars()!
main() goes on its merry way.
Fun Fact:

If you switch the two lines in else{}, you print the string backwards!

(Do you see why?)

```c
void print_chars(char str[], int spot)
{
    // Base case: at end of string
    if (str[spot] == '\0')
        return;
    else
    {
        // Call print_chars with next spot
        print_chars(str, spot + 1);
        // Print the character
        printf("%c\n", str[spot]);
    }
}
```

Will call itself before printing, stacking frames, and will print when the frames are returning!
On the return:

void print_chars(char str[], int spot)
{
    // Base case: at end of string
    if (str[spot] == '\0')
        return;
    else
    {
        // Call print_chars with next spot
        print_chars(str, spot + 1);
        // Print the character
        printf("%c", str[spot]);
    }
}

Somewhere in main():
... print_chars(str, 0); ...
with str:

```
f t w /0
```

Printed:
```
ftw
```

Done!

(ftw != wtf....)

```
p
```
Quiz[Recursion]

• “What’s this do?”
  – Think about call stack
    • Draw it out if need be
  – Remember where execution stopped on prev instance
    • (e.g. at the recursive call)

• “Write one.”
  – What’s repeating? What’s changing?
  – Base case!
Searching
(for the number 50)

• Linear: $O(n)$, $\Omega(1)$

| 3 | 7 | 17 | 42 | 50 | 61 | 171 |

• Binary: $O(\log n)$, $\Omega(1)$

| 3 | 7 | 17 | 42 | 50 | 61 | 171 |

Note: list needs to be pre-sorted for binary search—but it’s worth it!
Selection Sort

- \(O(n^2), \Omega(n^2)\)

1. Look for smallest # in unsorted part
2. Switch it with the first slot of the unsorted
3. Repeat until all sorted

\[\begin{array}{cccc}
3 & 17 & 42 & 50 \\
\end{array}\]
Bubble Sort

- $O(n^2), \Omega(n)$

1. Go down list
   - If two adjacent #’s are out of order, swap ‘em
2. When at end of list, repeat if swaps were made

(+ once more through to make sure everything’s in order, that there aren’t any swaps)
Merge Sort

- $O(n \log n)$, $\Omega(n \log n)$
- Recursive!

$mSort$ (list of $n$)

- If $n < 2$, return.
- Else
  - $mSort$ (left half).
  - $mSort$ (right half).
  - Merge sorted

<table>
<thead>
<tr>
<th>3</th>
<th>17</th>
<th>42</th>
<th>50</th>
<th>61</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>3</td>
<td>50</td>
<td>61</td>
<td>17</td>
</tr>
<tr>
<td>42</td>
<td>3</td>
<td>50</td>
<td>61</td>
<td>17</td>
</tr>
<tr>
<td>42</td>
<td>3</td>
<td></td>
<td>61</td>
<td>17</td>
</tr>
<tr>
<td>42</td>
<td>3</td>
<td></td>
<td>17</td>
<td>61</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
<td>50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Asymptotic Notation

- Algorithm Efficiency
- \( O(n) \) – upper bound
- \( \Omega(n) \) – lower bound
- \( \Theta(n) \) – \( O(n) =\Omega(n) \)

\( n = \) size of input

(best) Constant < Log < Linear < Quadratic / Polynomial < Exponential < Factorial (worst)
Quiz[AsymptoticNotation]

• Just memorize or cheatsheet it.
• Or...walk though algorithm & think about math.

Ex: Merge Sort, list length $n$

Length depends on how many times list was halved (steps)—
mathematically: $n \sim 2^{\text{steps}}$

So... steps $\sim \log_2 n$. (Done?!) But... at each halved level, have to walk through to compare & merge...
...so $\sim n$ additional steps per level.

steps $\sim n \times \log_2 n$
Part 3

Tian Feng
Fun With Pointers

Notation:
• \&var returns the address of var
  – \&tian == eliot
• *ptr returns the value referenced by ptr
  – *eliot == tian

Pointer Arithmetic
• Move around addresses
  – Incidentally, array[i] = *(array + i), we’ll discuss this more later in the semester
Malloc and Heaps

Malloc
• Dynamic memory allocation
• Syntax:
  – type *array = malloc(size);
• Memory created on the heap
• Used in conjunction with free()
  – free(array);

Heap
• Dynamically allocated memory with variable length duration
• Exists until released by user
Arrays and Strings

• The name of an array is just a pointer to the first value in the array

• Strings are just arrays of chars
  – End with ‘\0’, the null character
  – Thus the name of a string is a reference to the location of the first char of the string
Structs: Custom object of aggregated data types
• struct name
  {
    _______;  
  }

When referencing data in a struct:
• Struct.field
  – tian.name
• Ptr->field
  – eliot->name

Typedef: Custom naming of data types/objects
• typedef ________ name;

When using typedef and structs in conjunction:
• typedef struct
  {
    int id;
    char name[30];
  } student;
## Linked Lists vs. Arrays

<table>
<thead>
<tr>
<th>Linked Lists</th>
<th>Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Organized collections</td>
<td>• Organized collections</td>
</tr>
<tr>
<td>• Traversable</td>
<td>• Traversable</td>
</tr>
<tr>
<td>• Variable Length</td>
<td>• Fixed Length</td>
</tr>
<tr>
<td>• Variable Order</td>
<td>• Fixed Order</td>
</tr>
<tr>
<td>• Non Index-able</td>
<td>• Index-able</td>
</tr>
<tr>
<td>• Non-contiguous blocks of memory</td>
<td>• Contiguous blocks of memory</td>
</tr>
</tbody>
</table>

- Non-contiguous blocks of memory: 
  - Linked Lists: 
    - Data stored at different memory locations.
    - No fixed structure.
  - Arrays: 
    - Data stored in a continuous block of memory.
    - Fixed structure.

- Contiguous blocks of memory:
  - Linked Lists: 
    - No contiguous memory blocks.
  - Arrays: 
    - Data stored in a single contiguous block.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Linked Lists vs. Arrays Pros & Cons

**Linked Lists**
- **Pros**
  - Arbitrary insertions and deletions
  - Easy to reorder
  - No need for a contiguous block of memory
- **Cons**
  - Linear time access
  - Overhead for pointer data

**Arrays**
- **Cons**
  - Need to reallocate memory and transfer array
  - Need to “bump” every other value
  - Ahh! Contiguous block
- **Pros**
  - Constant time access (index)
  - Minimal storage overhead
Linked List Construction

typedef struct _node
{
    struct _node *next;
    ____________;
} student;
Stacks and Queues

Stacks
• LIFO
  – “last in first out”
• Real life applications:
  – Box of saltines
• Like the stack memory construct

Queues
• FIFO
  – “first in first out”
• Real life applications:
  – Lines in restaurants
  – Printer queues
File I/O

File related calls:

• fopen and fclose
  – Open and close file

• Fgetc
  – Gets a char from the file

• fprintf
  – Prints in file in stated format
questions?