Characters, Strings,
Larger C Program organization

Jinyang Li
Characters
How to represent text characters?

- How to associate bit patterns to integers?
  - base 2
  - 2’s complement

- How to associate bit patterns to floats?
  - IEEE floating point representation (based on normalized scientific notation)

- How to associate bit patterns to characters?
  - by convention
  - ASCII, UTF
ASCII: American Standard Code for Information Exchange

- Developed in 60s, based on the English alphabet
- use one byte (with MSB=0) to represent each character
- How many unique characters can be represented?

128
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>[NULL]</td>
<td>32</td>
<td>20</td>
<td>[SPACE]</td>
<td>64</td>
<td>40</td>
<td>@</td>
<td>96</td>
<td>60</td>
<td>`</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>[START OF HEADING]</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>[START OF TEXT]</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>[END OF TEXT]</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>[END OF TRANSMISSION]</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>(ENQUIRY)</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>[ACKNOWLEDGE]</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>[BEL]</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>[BACKSPACE]</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>[HORIZONTAL TAB]</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>[VERTICAL TAB]</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>[FORM FEED]</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>[CARRIAGE RETURN]</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>[SHIFT OUT]</td>
<td>46</td>
<td>2E</td>
<td>,</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>[SHIFT IN]</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>[DATA LINK ESCAPE]</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>[DEVICE CONTROL 1]</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>[DEVICE CONTROL 2]</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>[DEVICE CONTROL 3]</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>[DEVICE CONTROL 4]</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>[NEGATIVE ACKNOWLEDGE]</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>[SYNCHRONOUS IDLE]</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>[ENG OF TRANS. BLOCK]</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>[CANCEL]</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>[END OF MEDIUM]</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>[SUBSTITUTE]</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>[ESCAPE]</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>{</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>[FILE SEPARATOR]</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>[GROUP SEPARATOR]</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>}</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>[RECORD SEPARATOR]</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>[UNIT SEPARATOR]</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>_</td>
<td>127</td>
<td>7F</td>
<td>[DEL]</td>
</tr>
</tbody>
</table>
C exercise 1: tolower

// tolower returns the corresponding
// lowercase character for c if c is an
// uppercase letter. Otherwise, it returns c.
char tolower(char c) {

}

int main() {
    char c = tolower('A');
    printf("resulting c is %c\n", c);
}

C exercise 1: tolower

// tolower returns the corresponding lowercase character for c if c is an uppercase letter. Otherwise, it returns c.
char tolower(char c) {

    // test if c is an uppercase letter
    if (c < 'A' || c > 'Z') {
        return c;
    }

}
C exercise 1: tolower

// tolower returns the corresponding lowercase character for c if c is an uppercase letter. Otherwise, it returns c.
char tolower(char c) {

    // test if c is an uppercase letter
    if (c < 'A' || c > 'Z') {
        return c;
    }

    return c + 'a' - 'A';
}

C’s standard library includes tolower, toupper
C exercise 2: toDigit

// toDigit returns the corresponding integer for c
// if c is a valid digit character, e.g ‘1’, ‘2’,
// Otherwise, it returns -1.
int toDigit(char c) {

}

int main() {
    int d = toDigit(‘8’);
    printf("int is %d, multiply-by-2 %d\n", d, 2*d);
}
C exercise 2: toDigit

// toDigit returns the corresponding integer for c
// if c is a valid digit character, e.g ‘1’, ‘2’,
// Otherwise, it returns -1.
int toDigit(char c) {
    // test if c is a valid character
    if (c < ‘0’ || c > ‘9’) {
        return -1;
    }
}

int main() {
    int d = toDigit(‘8’);
    printf("int is %d, multiply-by-2 %d\n", d, 2*d);
}
C exercise 2: toDigit

// toDigit returns the corresponding integer for c
// if c is a valid digit character, e.g ‘1’, ‘2’,
// Otherwise, it returns -1.
int toDigit(char c) {
    // test if c is a valid character
    if (c < ‘0’ || c > ‘9’) {
        return -1;
    }
    return c – ‘0’;
}

int main() {
    int d = toDigit(‘8’);
    printf(“int is %d, multiply-by-2 %d\n”, d, 2*d);
}
The Modern Standard: UniCode

• ASCII can only represent 128 characters
  – How about Chinese, Korean, all of the world's languages? Symbols? Emojis?

• Unicode standard represents >135,000 characters
UTF-8

- UTF-8 is one encoding form for Unicode
  - use 1, 2, or 4 byte to represent a character
  - Unicode for ASCII characters have the same ASCII value \(\rightarrow\) UTF-8 one byte code is the same as ASCII

- C has no primitive support for Unicode
C Strings
Strings

• String is represented as an array of chars.
  – Array has no space to encode its length.
• How to determine string length?
  – explicitly pass around an integer representing length

// tolower_string turns every character in character array s
// into lower case
void tolower_string(char *s, int len) {
    for (int i = 0; i < len; i++) {
        s[i] = tolower(s[i]);
    }
}
Strings

• String is represented as an array of chars.
  – Array has no space to encode its length.

• How to determine string length?
  – explicitly pass around an integer representing length
  – C string stores a NULL character to mark the end (by convention)

```c
void tolower_string(char *s) {
}
```
Strings

• String is represented as an array of chars.
  – Array has no space to encode its length.
• How to determine string length?
  – explicitly pass around an integer representing length
  – C string stores a NULL character to mark the end (by convention)

```c
void tolower_string(char *s) {
    int i = 0;
    while (s[i] != '\0') {
        s[i] = tolower(s[i]);
        i++;
    }
}
```
Copying string?

does this make a copy of “hi”?

```c
char s[4] = “hi”;
char *h;
h = s;
h[0] = ‘H’;

printf(“s=%s h=%s\n”, s, h);
```

![Diagram showing memory allocation and values]

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0xdeadbefef12345678</td>
</tr>
<tr>
<td>h</td>
<td>0x78</td>
</tr>
</tbody>
</table>
```

Does this make a copy of “hi”?
Copying string?

does this make a copy of “hi”?

```c
char s[4] = “hi”;  
char h[4];  
h = s;  
h[0] = ‘H’;  

printf(“s=%s h=%s\n”,s,h);
```
void strcpy(char *dst, char *src)
{
}

int main()
{
    char s[4] = "hi";
    char h[4];
    strcpy(h, s);
    h[0] = 'H';

    printf("s=%s h=%s\n", s, h);
}
void strcpy(char *dst, char *src) {
    int i = 0;
    while (src[i] != '\0') {
        dst[i] = src[i];
        i++;
    }
}

int main() {
    char s[4] = "hi";
    char h[4];
    strcpy(h, s);
    h[0] = 'H';

    printf("s=%s h=%s\n", s, h);
}
Copying string

```c
void strcpy(char *dst, char *src) {
    int i = 0;
    while (src[i] != '\0') {
        dst[i] = src[i];
        i++;
    }
}

int main() {
    char s[4] = "hi";
    char h[2];
    strcpy(h, s);
    h[0] = 'H';
    printf("s=%s h=%s\n", s, h);
    // Results in out-of-bound write!
    // Buffer overflow!
}
```
void strncpy(char *dst, char *src, int n) {
  int i = 0;
  while (src[i] != '\0' && i < n) {
    dst[i] = src[i];
    i++;
  }
}

int main() {
  char s[4] = "hi";
  char h[2];
  strncpy(h, s, 2);
  h[0] = 'H';

  printf("s=%s h=%s\n", s, h);
}
A different way of initializing string

```c
char s1[4] = "hi";
char *s2 = "bye";
s1[0] = 'H';          // OK
s2[0] = 'B';          // Segmentation fault (bus error)
printf("s1=%s s2=%s\n", s1, s2);
```
A different way of initializing string

```c
char s1[4] = "hi";
char *s2 = "bye";
s1[0] = 'H';
s2[0] = 'B';

printf("s1=%s  s2=%s\n",s1,s2);
```

```
s1:
  0x00
  'h'
  'h'
  ...
  0xdeadbefef12345678

s2:
  0x21
  0x00
  'e'
  'y'
  'b'
  ...
  0x00000000087654321

read-only memory
```
The Atoi function

// atoi returns the integer
// corresponding to the string of digits
int atoi(char *s)
{

}

int main()
{
    char *s = "123";
    printf("integer is %d\n", atoi(s));
}
# The Atoi function

// atoi returns the integer
// corresponding to the string of digits
int atoi(char *s) {
    int result = 0;
    int i = 0;
    while (s[i] >= '0' && s[i] <= '9') {
        result = result * 10 + (s[i] - '0');
        i++;
    }
    return result;
}

<table>
<thead>
<tr>
<th>'1'</th>
<th>'2'</th>
<th>'3'</th>
<th>'\0'</th>
</tr>
</thead>
<tbody>
<tr>
<td>result = 1</td>
<td>result = 1*10+2</td>
<td>result = 12*10+3</td>
<td>end loop</td>
</tr>
</tbody>
</table>
Array of pointers

```c
char* names[3] = {
    "alice",
    "bob",
    "clark"
};

char **namep;
namep = names;

printf("name is %s", namep[1]);
```
The most commonly used array of pointers: argv

```c
int main(int argc, char **argv)
{
    for (int i = 0; i < argc; i++) {
        printf("%s\n", argv[i]);
    }
}
```

$ ./a.out 1 2 3
./a.out 1 2 3

argv[0] is the name of the executable
Organization of large C programs
typedef struct {
    int val;
    struct node *next;
} node;

node* insert(node *head, int val) {
    node *new_head = (node *)malloc(sizeof(node));
    new_head->next = head;
    new_head->val = val;
}

int main() {
    node *head = NULL;
    for (int i = 0; i < 3; i++)
        head = insert(head, i);
}

What if another program also wants to use this linked list implementation?
linked list: multiple files

typedef struct {
  int val;
  struct node *next;
} node;

node *insert(node *head, int val) {
  node *new_head = (node *)malloc(sizeof(node));
  new_head->next = head;
  new_head->val = val;
}

#include "list.h"

node* insert(node *head, int val) {
  insert(node *head, int val) {

$ gcc -c list.c$
$ gcc list.c$
linked list: multiple files

```c
#include "list.h"
int main() {
    node *head = NULL;
    for (int i = 0; i < 3; i++)
        head = insert(head, i);
}
```

```c
#include "list.h"
int main() {
    node *head;
    for (int i = 0; i < 3; i++)
        head = insert(head, i);
}
```

```
gcc -c test1.c
$ gcc test1.o list.o
$ ./a.out
```

- generate object file test1.o
- link test1.o and list.o to form executable a.out
Exporting global variables

typedef struct {
    int val;
    struct node *next;
}node;

node *insert(node *head, int val);

#include "list.h"
int debug;
node* insert(node *head, int val) {
    ...
    if (debug > 0)
        printf("inserted val %d\n", val);
}

#include "list.h"
int main() {
    debug = 1;
    ...
}

Exporting global variables

typedef struct {
    int val;
    struct node *next;
} node;

extern int debug;

node *insert(node *head, int val);

#include "list.h"

int debug;

node * insert(node *head, int val) {
    ...
    if (debug > 0)
        printf("inserted val %d\n", val);
}

#include "list.h"

int main() {
    debug = 1;
    ...
}
C does not have explicit namespace

• Scope of a global variable / function by default is across all files (linked together)
• To restrict the scope of a global variable / function to this file only, prefix with “static” keyword

```c
#include "list.h"
static int debug;
static node* insert(node *head, int val) {
    ...
    if (debug > 0)
        printf("inserted val %d\n", val);
}
```

No other files can use the debug variable and insert function
static prefixing local variables means different things

• Normal local variables are de-allocated upon function exit

• Static local variables are not de-allocated
  – offers private, persistent storage across function invocation

```c
node* insert(node *head, int val) {
    static int n_inserts = 0;
    ...
    n_inserts++;
    printf("number of inserts %d\n", n_inserts);
}
```

initialized once, never deallocated
(like a global variable, except with local scope)
C standard library

<assert.h> assert
<ctype.h> isdigit(c), isupper(c), isspace(c), tolower(c), toupper(c) ...
<math.h> log(f) log10(f) pow(f, f), sqrt(f), ...
<stdio.h> fopen, fclose, fread, fwrite, printf, ...
<stdlib.h> malloc, free, atoi, rand
<string.h> strlen, strcpy, strcat, strcmp

To read manual, type man 3 strlen

Section 3 of manpage is dedicated to C std library
The C pre-processor

• All the hashtag directives are processed by C pre-processor **before** compilation

• `#include <stdio.h>`
  – insert text of included file in the current file
  – with `<...>` , preprocessor searches system path for specified file
  – with “...”, preprocessor searches local directory as well as system path
C Macros

• `#define name replacement_text`

```c
#define NITER 10000

int main()
    for (int i = 0; i < NITER; i++) {
        ....
    }
}
```

It’s better to write:
```c
static const int niter = 10000;
```
C Macros

• Macro can have arguments
• Macro is NOT a function call

#define SQUARE(X) X*X

a = SQUARE(2);

b = SQUARE(i+1);

c = SQUARE(i++);

a = 2*2;
b = i+1*i+1;
C Macros

- Macros can have arguments
- Macro is NOT a function call

#define SQUARE(X) (X)*(X)

a = SQUARE(2);  
a = (2)*(2);

b = SQUARE(i+1);  
b = (i+1)*(i+1);

c = SQUARE(i++);  
c = (i++)*(i++);

Macro is hard to debug, avoid it if you can