# Chapter Nine

# Characters and Strings

1

## Text Data

- These days, computers work less with *numeric data* than with *text data*
- To unlock the full power of text data, you need to know how to manipulate *strings* in more sophisticated ways
- Because a string is composed of individual *characters*, it is important for you to understand how character work and how they are represented inside the computer

2

## Enumeration Types

- There are many types of useful data that are *neither numeric* data *nor text* data
- The days of a week: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
- The classes of students in the school: freshman, sophomore, junior, senior

3

## Enumeration Types

- The process of listing all the elements in the domain of a type is called *enumeration*
- A type defined by listing all of its elements is called an *enumeration type*
- *Characters* are similar in structure to enumeration types

4

## Representing Enumeration Types

- *How* do computers *represent internally* the values of enumeration types
- Computers are good at manipulating *numbers*
- To represent a finite set of values of any type, all you have to do is to *give each value a number*
- The process of assigning an integer to each element of an enumeration type is called *integer encoding*

5

## An Example

```
#define    Sunday      0
#define    Monday      1
#define    Tuesday     2
#define    Wednesday   3
#define    Thursday    4
#define    Friday      5
#define    Saturday    6
```

```
int  weekday;
```

```
#define    Freshman    1
#define    Sophomore   2
#define    Junior      3
#define    Senior      4
```

```
int  class;
```

6

## Defining Enumeration Types

- A new *enumeration type* can be defined as

```
typedef  enum {
        list of elements
} type-name;
```

For example,

```
typedef  enum {
        FALSE, TRUE
} bool;
```

7

## An Example

```
typedef  enum {
   Sunday, Monday, Tuesday, Wednesday,
   Thursday, Friday, Saturday
} weekdayT;

typedef  enum {
   Freshman, Sophomore, Junior, Senior
} classT;
```

8

## Advantages

- The compiler is *able to choose the integer codes*, thereby freeing the programmer from the responsibility
- A *separate* and *meaningful* type name instead of int makes the program *easier to read*
- Explicitly defined enumeration types are *easier to debug*

9

## Integer Encoding

- You can specify *explicitly* the integer codes associated with the elements of an enumeration type as part of the definition
- If an element is *not explicitly* assigned an integer code, a *consecutive* integer code next to the previous element is assigned
- By default, the integer codes for the elements start with 0

10

## An Example

```
typedef  enum {
   Sunday, Monday, Tuesday, Wednesday,
   Thursday, Friday, Saturday
} weekdayT;

typedef  enum {
   Freshman = 1, Sophomore, Junior, Senior
} classT;

typedef  enum {
   FALSE, TRUE
} bool;
```

11

## Operations on Enumeration

- C compilers *automatically* convert *values of an enumeration type* to *integers* whenever the values are used in an expression
- All *arithmetic* for enumeration types works the same way as it does for *integers*
- However, compilers do not check if the value of an expression is still a *valid* value of an enumeration type

```
weekday = (weekday + 1) % 7;
```

12

## An Example

```
typedef enum { North, East, South, West }  directionT;

directionT OppositeDirection(directionT dir)
{
   switch (dir) {
      case  North: return  South;
      case  East:   return West;
      case  South: return North;
      case  West:  return East;
      default: printf("Illegal direction value.");
   }
}
```

13

## Characters

- In C, single characters are represented using the type char
- The type char is a *built-in* enumeration type
- The domain of *values* of char is the set of *symbols* that can be displayed on the screen or typed on the keyboard
- The set of *operations* for char is the *same* as that for int

14

## ASCII Character Set

- To allow *effective communication* among computers, *standard integer encoding* systems for characters have been proposed
- The most commonly used system is the ASCII (American Standard Code for Information Interchange) *character set*

15

## ASCII Character Set

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | \000 | \001 | \002 | \003 | \004 | \005 | \006 | \a | \b | \t |
| 10 | \n | \v | \f | \r | \016 | \017 | \020 | \021 | \022 | \023 |
| 20 | \024 | \025 | \026 | \027 | \030 | \031 | \032 | \033 | \034 | \035 |
| 30 | \036 | \037 | space | ! | " | # | $ | % | & | ' |
| 40 | ( | ) | * | + | , | - | . | / | 0 | 1 |
| 50 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; |
| 60 | < | = | > | ? | @ | A | B | C | D | E |
| 70 | F | G | H | I | J | K | L | M | N | O |
| 80 | P | Q | R | S | T | U | V | W | X | Y |
| 90 | Z | [ | \ | ] | ^ | _ | ` | a | b | c |
| 100 | d | e | f | g | h | i | j | k | l | m |
| 110 | n | o | p | q | r | s | t | u | v | w |
| 120 | x | y | z | { | \| | } | ~ | \177 | | |

16

## Character Constants

- *Character constant* is written by enclosing the desired character in single quotation marks
  'A' => 65
  '9' => 57
- *Avoid using integer constants* to refer to ASCII characters within a program

17

## Properties of ASCII Set

- The codes for the *digits* 0 *through* 9 are *consecutive*
- The codes for the *uppercase letters* are *consecutive*
- The codes for the *lowercase letters* are *consecutive*

18

## Special Characters

- The characters that can be displayed on the screen are called *printing characters*
- The other characters that are used to perform a particular operation are called *special characters*
- Special characters are represented as *escape sequences* that consist of a backslash '\' followed by a letter or an octal numeric value

19

## Escape Sequence

| | |
|---|---|
| \a | Audible alert (beeps or rings a bell) |
| \b | Backspace |
| \f | Formfeed (starts a new page) |
| \n | Newline (moves to the beginning of the next line) |
| \r | Return (returns to the beginning of the current line) |
| \t | Tab (moves horizontally to the next tab stop) |
| \v | Vertical tab (moves vertically to the next tab stop) |
| \0 | Null character (the character whose ASCII code is 0) |
| \\ | The character \ itself |
| \′ | The character ′ (only in character constants) |
| \″ | The character ″ (only in string constants) |
| \ddd | The character whose ASCII code is octal number ddd |

20

## Character Arithmetic

- *Adding an integer to a character*
  '0' + 5 => '5', 'A' + 5 => 'F'
- *Subtracting an integer from a character*
  '5' – 5 => '0', 'F' – 5 => 'A'
- *Subtracting one character from another*
  'X' + ('a' – 'A') => 'x'
- *Comparing two characters against each other*
  'F' > 'A' => TRUE, 'F' > 'f' => FALSE

21

## Types of Characters

- The ctype.h interface declares several predicate functions for determining the type of a character
  - islower(ch) TRUE if ch is a lowercase
  - isupper(ch) TRUE if ch is a uppercase
  - isalpha(ch) TRUE if ch is a letter
  - isdigit(ch) TRUE if ch is a digit
  - isalnum(ch) TRUE if ch is a letter or digit
  - ispunct(ch) TRUE if ch is a punctuation
  - isspace(ch) TRUE if ch is ' ', '\f', '\n', '\t', or 'v'

22

## An Example

```
bool islower(char ch)
{
    return (ch >= 'a' && ch <= 'z');
}

bool isdigit(char ch)
{
    return (ch >= '0' && ch <= '9');
}
```

23

## Conversion of Letters

- The ctype.h interface also declares two extremely useful conversion functions
  - tolower(ch): If ch is an uppercase letter, returns its lowercase equivalent; otherwise returns ch unchanged
  - toupper(ch): If ch is an lowercase letter, returns its uppercase equivalent; otherwise returns ch unchanged

24

## An Example

```
char tolower(char ch)
{
   if (ch >= 'A' && ch <= 'Z') {
      return ch + ('a' – 'A');
   } else {
      return ch;
   }
}
```

25

## Reasons for Using Libraries

- Because the library functions are standard, it is *easier* for other programmers *to read* library functions than your own
- It is *easier* to rely on library functions *for correctness* than on your own
- The library implementation of functions are often *more efficient* than your own

26

## Characters in Switch

```
bool isVowel(char ch)
{
   switch (tolower(ch)) {
      case 'a': case 'e': case 'i': case 'o': case 'u':
         return TRUE;
      default:
         return FALSE;
   }
}
```

27

## Character Input & Output

- Character input is performed using
  `int getchar(void);`
  in stdio.h. It returns the character read or EOF if end of file or error occurs
- Character output is performed using
  `int putchar(ch);`
  in stdio.h. It returns the character written or EOF if error occurs

28

## An Example

*A cyclic letter-substitution cipher*:

Cipher code = 4

I am a student from Taiwan.

⇩

M eq e wxyhirx jvsq Xemaer.

29

## An Example

```
main() {
   int  k, ch;
   printf("Key in cipher code? "); scanf("%d", &k);
   while ((ch = getchar()) != EOF) {
      if (isupper(ch)) {
         ch = (ch – 'A' + k) % 26 + 'A';
      } else if (islower(ch)) {
         ch = (ch – 'a' + k) % 26 + 'a';
      }
      putchar(ch);
   }
}
```

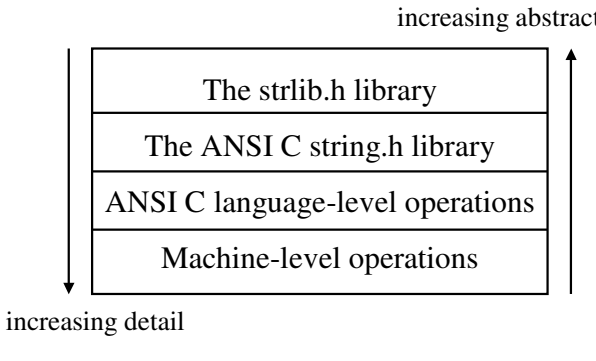
30

## Strings

- A *string* is a *sequence of characters*
- In this chapter, you will learn the *abstract behaviors* of strings by using a string library that defines a type string and hides the *internal representation* of strings and many *manipulations* of strings, just like int and double
- You will learn those *complex details* in the later chapters

31

## Layered Abstraction




32

## Abstract Types

- An *abstract* type is a type defined only by its *behavior* and not in terms of its *representation*
- The *behavior* of an abstract type is defined by the *operations* that can be performed on objects of that type. These operations are called *primitive operations*

33

## The strlib.h Library

- This library contains the following functions

| | |
|---|---|
| getLine() | read a line as a string |
| stringLength(s) | length of a string |
| ithChar(s, i) | ith character of a string |
| concat($s_1$, $s_2$) | concatenates two strings |
| copyString(s) | copy a string |
| substring(s, $p_1$, $p_2$) | extract a substring |
| stringEqual($s_1$, $s_2$) | Are two strings equal |
| stringCompare($s_1$, $s_2$) | compare two strings |
| charToString(ch) | convert char to string |

34

## The strlib.h Library

- This library contains the following functions

| | |
|---|---|
| findChar(ch, str, p) | find a character |
| findString(s, str, p) | find a substring |
| convertToLowerCase(s) | converts to lowercase |
| convertToUpperCase(s) | converts to uppercase |
| intToString(i) | converts integer to string |
| realToString(ch) | converts real to string |
| stringToInt(s) | converts string to integer |
| stringToReal(s) | converts string to real |

35

## getLine & stringLength

```
main()
{
   string  str;

   printf("Key in a string: ");
   str = getLine();
   printf("The length of %s is %d.\n",
                  str, stringLength(str));
}
```

36

## ithChar

```
/* "student" => 't' */

char lastChar(string  str)
{
   return (ithChar(str, stringLength(str) - 1);
}

/* The positions within a string are numbered
    starting from 0 */
```

37

## concat

```
string concatNCopies(int n, string  str)
{
   string result;
   int i;
   result = "";
   for (i = 0; i < n; i++)
      result = concat(result, str);
   return result;
} /* (4, "*") => "****" */
```

38

## charToString

```
string reverseString(string  str)
{
   string result, temp;
   int i;
   result = "";
   for (i = 0; i < stringLength(str); i++) {
      temp = charToString(ithChar(str, i));
      result = concat(temp, result);
   }
   return result;
} /* "student" => "tneduts" */
```

39

## subString

```
string secondHalf(string  str)
{
   int len;

   len = stringLength(str);
   return subString(str, len / 2, len - 1);
}
```

40

## subString

- If p1 is negative, it is set to 0 so that it indicates the first character in the string
- If p2 is greater than stringLength(s) - 1, it is set to stringLength(s) – 1 so that it indicates the last character
- If p1 ends up being greater than p2, subString returns the *empty string*

41

## stringEqual

```
main()
{
   string  answer;
   while (TRUE) {
      playOneGame();
      printf("Would you like to play again? ");
      answer = getLine();
      if (stringEqual(answer, "no")) break;
   }
}
```

42

## stringCompare

- If s1 precedes s2 in lexicographic order, stringCompare returns a *negative integer*
- If s1 follows s2 in lexicographic order, stringCompare returns a *positive integer*
- If the two string are exactly the same, stringCompare returns 0
- The *lexicographic* order is different from the *alphabetical* order used in dictionaries

43

## findChar

```
string Acronym(string  str) {
   string acronym;  int pos;
   acronym = charToString(ithChar(str, 0));
   pos = 0;
   while (TRUE) {
      pos = findChar(' ', str, pos + 1);
      if (pos == -1) break;
      acronym = concat(acronym,
         charToString(ithChar(str, pos + 1)));
   }
   return acronym;
} /* "Chung Cheng University" => "CCU" */
```

44

## findString

replaceFirst("a plan", "a", "a nice") => "a nice plan"

```
string replaceFirst(string  str, string pat, string replace) {
   string head, tail;  int pos;

   pos = findString(pat, str, 0);
   if (pos == -1) return str;
   head = subString(str, 0, pos - 1);
   tail = subString(str, pos + stringLength(pat),
                            stringLength(str) - 1);
   return concat(concat(head, replace), tail);
}
```

45

## convertToLowerCase

```
string convertToLowerCase(string  str)
{
   string result;  char ch;  int i;

   result = "";
   for (i = 0; i < stringLength(str); i++) {
      ch = ithChar(str, i);
      result = concat(result, charToString(tolower(ch)));
   }
   return result;
}
```

46

## Numeric Conversion

- The function intToString(n) converts the integer n into a string of digits, preceded by a minus sign if n is negative
  intToString(123) => "123"
  intToString(-4) => "-4"
- The function realToString(d) converts the floating point d into the string that would be displayed by printf using the %G format code
  realToString(3.14) => "3.14"
  realToString(0.00000000015) => "1.5E-10"

47

## protectedIntegerField

*****123

```
string protectedIntegerField(int  n, int places)
{
   string numstr, fill;

   numstr = intToString(n);
   fill = concatNCopies(places -
                        stringLength(numstr), "*");
   return concat(fill, numstr);
}
```

48