

Character and String Processing

CSE 130: Introduction to C Programming
Spring 2005

1

The char Data Type

- A char value can be thought of as either a character or a small integer
 - `printf("%d", 'a');` /* prints 97 */
 - `printf("%c", 97);` /* prints 'a' */
- Backslash ('\') is an escape character
 - Ex. `\n, \t, \', \\`

2

char Arithmetic

- Remember that characters are integers
 - `('a' + 1)` is 'b'
 - `('Z' - 'A')` is 25
 - `('A' - 'a')` has the same value as `('B' - 'b')`
- If `c` holds a lowercase letter, then `(c + 'A' - 'a')` is the corresponding capital

3

Another Example

```
/* Capitalize all lowercase letters */
while ((c = getchar()) != EOF)
{
    if ('a' <= c && c <= 'z')
        putchar(c + 'A' - 'a');
    else
        putchar(c);
}
```

4

getchar() and putchar()

- These are macros defined in `<stdio.h>`
- `getchar()` reads a character from the keyboard
- `putchar()` writes a character to the screen
- EOF is a constant that means "end-of-file"

5

An Example

```
int c; /* c can hold any character value */
while ((c = getchar()) != EOF)
{
    putchar(c);
}
```

6

ctype.h Macros

- isalpha(c), isdigit(c), isalnum(c), isxdigit(c)
- isupper(c), islower(c)
- isspace(c), ispunct(c)
- isprint(c), isgraph(c), iscntrl(c)
- isascii(c)
- toupper(c), tolower(c), toascii(c)

7

Strings

- A string is a one-dimensional array of characters
- Strings are terminated by the null character '\0'
- "%s" denotes a string in printf/scanf:
char w[100];
scanf("%s", w);

8

String Initialization

- The following are equivalent:
 - char s[] = {'a', 'b', 'c', '\0'};
 - char s[] = "abc";
- We can also use pointers:
 - char *p = "abc";
 - Here, the compiler stores the constant in memory

9

An Example

```
/* Build a string character by character */
char name[100];
int i, c;
for (i = 0; (c = getchar()) != '\n'; i++)
    name[i] = c;
```

10

Pointer Example

```
char new_string[100], *p = new_string;
int c;
while ((c = getchar()) != '\n')
{
    *p = c;
    p++;
}
*p = '\0'; /* terminate the string */
```

11

Counting Words (I)

```
int word_cnt(const char * s)
{
    int cnt = 0;
    /* word-counting code (next slide) */
    return cnt;
}
```

12

Counting Words (2)

```
while (*s != '\0')
{
    while (isspace(*s)) /* skip whitespace */
        s++;
    /* word-counting code (next slide) */
}
```

13

Counting Words (3)

```
if (*s != '\0') /* found a word */
{
    cnt++;
    while (!isspace(*s) && *s != '\0') /* skip word */
        s++;
}
```

14

String-Handling Functions

- Found in <string.h>
- `char *strcat(char *s1, const char *s2);`
 - Concatenates s1 and s2, stores in/returns s1
- `int strcmp(const char *s1, const char *s2);`
 - returns -1, 0, or 1 if s1 is <, ==, or > s2

15

More String Functions

- `char *strcpy(char *s1, const char *s2)`
 - Copies s2 into s1 until '\0' is encountered
 - Returns s1
- `char *strncpy(char *s1, const char *s2, int n);`
 - Copies n characters into s1, plus padding ('\0') if n < (length of s1)
 - Use this instead of strcpy!

16

More String Functions

- `unsigned strlen(const char *s)`
 - Returns # of characters in s before '\0'
- `char *strtok(char *str, const char *set);`
 - Separates str into tokens
 - tokens are delimited by characters in set
 - For subsequent calls, first argument is NULL

17

Arguments to main()

- To pass command-line arguments to main(), we need to use arrays of pointers
 - Ex. gcc <filename>
- main() can take two arguments:
 - argc (an int)
 - argv[] (an array of char *)

18

argc and argv

- argc provides a count of the command-line arguments
- argv[0] contains the name of the command
- argc is always at least 1
- argv contains the command-line arguments themselves

19

An Example

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

20

Sample Output

- Input:
my_echo
- Output:
argc = 1
argv[0] = my_echo

21

Another Example

- Input: my_echo try this
- Output:
argc = 3
argv[0] = my_echo
argv[1] = try
argv[2] = this

22

More on argv

- argv could also have been declared as:
 - char **argv;
- This is a pointer to pointer to char
- Alternately, argv is an array of pointers to char (an array of strings)
- The system allocates space for argv

23

The sizeof Operator

- sizeof returns an integer indicating the number of bytes required to store an object
 - Ex. sizeof (object)
- An object can be a type (int, float, double), an expression (a + b), an array, etc.
- Parentheses are only required if sizeof is applied to a type
 - Ex. sizeof (int)

24

Dynamic Memory Allocation

- `<stdlib.h>` contains the `calloc()` and `malloc()` functions
 - `calloc()` initializes allocated memory to 0
- Returns a void pointer to new memory
- Usage:
 - `calloc(<n>, <object size>);`
 - Allocates ($n * \text{<object size>}$) bytes

25

Allocation Example

```
/* Goal: allocate an array of n ints */  
int * a = calloc(n, sizeof(int));  
  
/* a points to a suitable block of memory, or  
NULL if the allocation was unsuccessful */
```

26

free()

- Dynamically-allocated memory is not released automatically (except when the program ends)
- To release unused space, use `free()`:

```
int *a = calloc(n, sizeof(int));  
  
...  
free(a);
```

27