COMP1511 18s2 (webcms)

Code Examples on characters_and_strings

andrew_rocks0.c

Print "Andrew Rocks!" - using ASCII codes for the characters

Compare the 8 andrew_rocks?.c programs which are all equivalent to get a better understanding of how & why C encodes character sequences

```
#include <stdio.h>

int main(void) {
    putchar(65);     // printf("%c", 65) is equivalent
    putchar(110);
    putchar(100);
    putchar(114);
    putchar(101);
    putchar(119);
    putchar(32);
    putchar(82);
    putchar(111);
    putchar(99);
    putchar(107);
    putchar(115);
    putchar(33);
    putchar(10);
    return 0;
}
```

andrew_rocks1.c

Print "Andrew Rocks!" - using character constants to get the ASCII codes for the characters
'A' is the C shorthand for the ASCII code for the character A (65)

```
#include <stdio.h>

int main(void) {
    putchar('A');   // equivalent to putchar(65)
    putchar('n');
    putchar('d');
    putchar('r');
    putchar('e');
    putchar('w');
    putchar(' ');
    putchar('R');
    putchar('o');
    putchar('c');
    putchar('k');
    putchar('s');
    putchar('\n');
    return 0;
}
```

andrew_rocks2.c

Print "Andrew Rocks!" - using character constants to get the ASCII codes for the characters, store them in array, and print the array.
Note we have to track the array length.

```c
#include <stdio.h>

#define LENGTH 14

int main(void) {
    int asciiCodes[14];

    asciiCodes[0] = 'A';
    asciiCodes[1] = 'n';
    asciiCodes[2] = 'd';
    asciiCodes[3] = 'r';
    asciiCodes[4] = 'e';
    asciiCodes[5] = 'w';
    asciiCodes[6] = ' ';
    asciiCodes[7] = 'R';
    asciiCodes[8] = 'o';
    asciiCodes[9] = 'c';
    asciiCodes[10] = 'k';
    asciiCodes[11] = 's';
    asciiCodes[12] = '!';
    asciiCodes[13] = '\n';

    int i = 0;
    while (i < LENGTH) {
        putchar(asciiCodes[i]);
        i = i + 1;
    }

    return 0;
}
```

andrew_rocks3.c

Print "Andrew Rocks!" - using character constants to get the ASCII codes for the characters, initialize an array to the values , and print the array.
Note we have to track the array length.

```c
#include <stdio.h>

#define LENGTH 14

int main(void) {
    int asciiCodes[LENGTH] = {'A','n','d','r','e','w',' ','R','o','c','k','s','!','\n'};

    int i = 0;
    while (i < LENGTH) {
        putchar(asciiCodes[i]);
        i = i + 1;
    }

    return 0;
}
```

andrew_rocks4.c

Print "Andrew Rocks!" - using character constants to get the ASCII codes for the characters, and initialize the array to the vales array using , and print the array.

This version has a extra special value in the array (0) to indicate the end of the sequence. This means we no longer have to keep track of how many characters in the array (note the while loop condition)

```c
#include <stdio.h>

int main(void) {
    // if we don't specify the size of an array being initialized C will make
    // it big enough to hold all the initializing elements (15 in this case)
    int asciiCodes[] = {'A','n','d','r','e','w',' ','R','o','c','k','s','!','\n',0};

    int i = 0;
    while (asciiCodes[i] != 0) {
        putchar(asciiCodes[i]);
        i = i + 1;
    }

    return 0;
}
```

andrew_rocks5.c

Print "Andrew Rocks!" - using character constants to get the ASCII codes for the characters, and initialize the array to the vales array using , and print the array.

This version has switched to using the numeric type char. This type is almost always 8 bits and shouldn't be used for arithmetic. It is commonly used to hold ASCII encodings.

```c
#include <stdio.h>

int main(void) {
    // if we don't specify the size of an array being initialized C will make
    // it big enough to hold all the initializing elements (15 in this case)
    char asciiCodes[] = {'A','n','d','r','e','w',' ','R','o','c','k','s','!','\n',0};

    int i = 0;
    while (asciiCodes[i] != 0) {
        putchar(asciiCodes[i]);
        i = i + 1;
    }

    return 0;
}
```

andrew_rocks6.c

Print "Andrew Rocks!" - using character constants to get the ASCII codes for the characters, and initialize the array to the vales array using , and print the array.

C has a convenient shorthand for char arrays containing a sequence of
ASCII codes with an extra 0 value marking the end of the sequence.
Its "Andrew Rocks!";

Compare the 8 andrew_rocks?.c programs which are all equivalent to get a better understand of how & why C encodes character sequences

```c
#include <stdio.h>

int main(void) {
    char asciiCodes[] = "Andrew Rocks!\n";
    int i;

    i = 0;
    while (asciiCodes[i] != 0) {
        putchar(asciiCodes[i]);
        i = i + 1;
    }

    return 0;
}
```

andrew_rocks7.c

Print "Andrew Rocks!" - using character constants to get the ASCII codes for the characters, and initialize the array to the vales array using , and print the array.

C has a convenient shorthand for char arrays containing a sequence of
ASCII codes with an extra 0 value marking the end of the sequence.
Its "Andrew Rocks!";

A number of C library functions accept zero-terminated char arrays
For example printf with the "%s" specification (below)

```c
#include <stdio.h>

int main(void) {
    char asciiCodes[] = "Andrew Rocks!\n";
    printf("%s", asciiCodes);
    return 0;
}
```

ascii.c

Print the 128 ASCII character encodings

```c
#include <stdio.h>

int main(void) {
    int ascii;
    ascii = 0;
    while (ascii < 128) {
        printf("In ASCII %d prints as %c\n", ascii, ascii);
        ascii = ascii + 1;
    }
    return 0;
}
```

getchar_eof.c

Read characters until eof

```c
#include <stdio.h>

int main(void) {
    // getchar returns an int which will contain either
    // the ASCII code of the character read or EOF

    int ch = getchar();
    while (ch != EOF) {
        printf("you entered the character: '%c' which has ASCII code %d\n", ch, ch);
        ch = getchar();
    }
    return 0;
}
```

upper_case.c

Read characters until eof, printing them with lower case letters convert to upper case

```c
#include <stdio.h>

int make_upper_case(int character);

int main(void) {
    // getchar returns an int which will contain either
    // the ASCII code of the character read or EOF

    int character = getchar();
    while (character != EOF) {

        int new_character = make_upper_case(character);
        putchar(new_character);

        character = getchar();
    }
    return 0;
}

// return upper case letter corresponding to lower case letter
// e.g. given 'f' return 'F'
// other characters returned unchanged
//
// library function toupper() in <ctype.h> does this task

int make_upper_case(int character) {
    if (character >= 'a' && character <= 'z') {
        int alphabetPosition = character - 'a';
        return 'A' + alphabetPosition;
    } else {
        return character;
    }
}
```

char2int.c

convert a single character to an int

```c
#include <stdio.h>

#define MAX_LINE 4096

int main(int argc, char *argv[]) {
    char line[MAX_LINE];
    int i;

    printf("Enter a single digit number: ");
    fgets(line, MAX_LINE, stdin);
    if (line[0] >= '0' && line[0] <= '9') {
        i = line[0] - '0';
        printf("You entered %d\n", i);
    }
    return 0;
}
```

string2int_getchar.c

convert a string read from stdin using getchar to an int

```c
#include <stdio.h>

int main(void) {

    printf("Enter a number: ");
    int c = getchar();

    int n = 0;
    while (c >= '0' && c <= '9') {
        n = 10 * n + (c - '0');
        c = getchar();
    }
    printf("You entered %d\n", n);

    return 0;
}
```

last_argument.c

print last argument

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

int main(int argc, char *argv[]) {
    if (argc == 1) {
        printf("I have no arguments\n");
    } else {
        printf("my last argument is %s\n", argv[argc - 1]);
    }
    return 0;
}
```

print_arguments.c

print command line argument

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

int main(int argc, char *argv[]) {
    int i;
    printf("argc=%d\n", argc);
    i = 0;
    while (i < argc) {
        printf("argv[%d]=%s\n", i, argv[i]);
        i = i + 1;
    }
    return 0;
}
```

sum_arguments.c

Convert command-line arguments to integers and print their sum.

No check is made that the command-line arguments are actually integers.
See strol for a more powerful library function which would allow checking.

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

int main(int argc, char *argv[]) {

    int sum = 0;
    int argument = 1;
    while (argument < argc) {
       sum = sum + atoi(argv[argument]);
       argument= argument + 1;
    }
    printf("sum of command-line arguments = %d\n", sum);

    return 0;
}
```

last_character.c

Simple example reading a line of input and examining characters

```
#include <stdio.h>

#define SIZE 8000

int main(void) {
    char x[SIZE];

    printf("Enter some input: ");
    if (fgets(x, SIZE, stdin) == NULL) {
        printf("Could not read any characters\n");
        return 0;
    }

    // the built-in function strlen could be used here
    int nCharacters = 0;
    while (x[nCharacters] != '\n' && x[nCharacters] != '\0') {
        nCharacters = nCharacters + 1;
    }

    // if we don't find a newline - the whole line can't have been read
    if (x[nCharacters] != '\n') {
        printf("Could not read read entire line\n");
        return 0;
    }

    printf("That line contained %d characters\n", nCharacters);

    if (nCharacters > 0) {
       printf("The first character was %c\n", x[0]);
       printf("The last character was %c\n", x[nCharacters-1]);
    }
    return 0;
}
```

fgets_eof.c

Read lines until eof

```c
#include <stdio.h>

#define MAX_LINE 1024

int main(void) {
    char line[MAX_LINE];

    // fgets returns NULL if it can't read any characters
    while (fgets(line, MAX_LINE, stdin) != NULL) {
        printf("you entered the line: %s", line);
    }
    return 0;
}
```

string2int_atoi.c

convert a string read from stdin to an int using the atoi function from the standard C library note no error checking

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

#define MAX_LINE 4096

int main(int argc, char *argv[]) {
    char line[MAX_LINE] = {0};
    int n;

    printf("Enter a number: ");
    fgets(line, MAX_LINE, stdin);
    n = atoi(line);
    printf("You entered %d\n", n);

    return 0;
}
```

string2int_fgets.c

convert a string read from stdin using fgets to an int

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

#define MAX_LINE 4096


int main(void) {
    char line[MAX_LINE] = {0};

    printf("Enter a number: ");
    fgets(line, MAX_LINE, stdin);
    int n = 0;
    int i = 0;
    while (line[i] > '0' && line[i] < '9') {
        n = 10 * n + (line[i] - '0');
        i = i + 1;
    }
    printf("You entered %d\n", n);
    return 0;
}
```

scanf_eof.c

Read numbers until eof or non-number encountered

```c
#include <stdio.h>

int main(void) {
    int num;
    // scanf returns the number of items read
    while (scanf("%d", &num) == 1) {
        printf("you entered the number: %d\n", num);
    }
    return 0;
}
```

getchar_eof1.c

Read characters until eof

```c
#include <stdio.h>

int main(void) {
    // getchar returns an int which will contain either
    // the ASCII code of the character read or EOF

    // using an assignment in a loop/if condition is
    // not recommended for novice programmers
    // but is used widely by experienced C programmers

    int ch;
    while ((ch = getchar()) != EOF) {
        printf("you entered the character: '%c' which has ASCII code %d\n", ch, ch);
    }
    return 0;
}
```

snap_line0.c

Reads lines of input until end-of-input
Print snap! if two consecutive lines are identical

See snap_line1.c for how to use functions to produce simpler code

```c
#include <stdio.h>

#define MAX_LINE 4096

int main(int argc, char *argv[]) {
    char line[MAX_LINE];
    char lastLine[MAX_LINE];

    // read first line into array lastLine
    printf("Enter line: ");
    fgets(lastLine, MAX_LINE, stdin);

    printf("Enter line: ");
    while (fgets(line, MAX_LINE, stdin) != NULL) {
        int i = 0;

        // count how many characters differ
        // between line & lastLine

        int differences = 0;
        while (line[i] != '\0' && lastLine[i] != 0) {
            if (lastLine[i] != line[i]) {
                differences = differences + 1;
            }
            i = i + 1;
        }

        if (differences == 0) {
            // lines are identical
            printf("Snap!\n");
        }

        // arrays can't be assigned so copy elements
        // of lastLine to line using a loop
        int j = 0;
        while (line[j] != '\0') {
            lastLine[j] = line[j];
            j = j + 1;
        }
        lastLine[j] = '\0';

        printf("Enter line: ");
```

snap_line1.c

Reads lines of input until end-of-input
Print snap! if two consecutive lines are identical

See snap_line2.c to see how to replace compareArrays & copyArray calls to with (strcmp & strcpy) from <string.h>

```c
#include <stdio.h>

#define MAX_LINE 4096

int compareArrays(char array1[], char array2[]);
void copyArray(char destinationArray[], char sourceArray[]);


int main(int argc, char *argv[]) {
    char line[MAX_LINE];
    char lastLine[MAX_LINE];

    // read first line into array lastLine
    printf("Enter line: ");
    fgets(lastLine, MAX_LINE, stdin);

    printf("Enter line: ");
    while (fgets(line, MAX_LINE, stdin) != NULL) {

        if (compareArrays(line, lastLine) == 0) {
            // lines are identical
            printf("Snap!\n");
        }

        copyArray(lastLine, line);

        printf("Enter line: ");
    }

    return 0;
}

// return 1 if array1 & array2 differ in any element, 0 otherwise
// array1 & array2 must be null-terminated char arrays
// strcmp from  <string.h> performs similar function

int compareArrays(char array1[], char array2[]) {
    int i = 0;
    while (array1[i] != '\0') {
        if (array1[i] != array2[i]) {
            return 1;
        }
```

print_message.c

Simple example of using functions & fgets

```c
#include <stdio.h>
#define MAX_PERSON 4096

void printManyMessages(int n, char nam[]);
void printMessages(char name[]);

int main(int argc, char *argv[]) {
    char person[MAX_PERSON] = {0};

    printf("Person's name? ");
    fgets(person, MAX_PERSON, stdin);

    int x = 0;
    while (person[x] != '\n' && person[x] != '\0') {
        x = x + 1;
    }
    person[x] = '\0';

    printManyMessages(1, person);

    printf("Repeat this how many times? ");
    int n;
    scanf("%d", &n);

    printManyMessages(n, person);

    return 0;
}

void printManyMessages(int n, char message[]) {
    int i = 0;
    while (i< n) {
        printMessages(message);
        i = i + 1;
    }
}


void printMessages(char name[]) {
    printf("%s is good\n", name);
    printf("%s is great\n", name);
    printf("We all love %s\n", name);
```

my_fgets.c

Simple implementation of fgets

```c
#include <stdio.h>

#define MAX_LINE 4096

char *my_fgets(char array[], int arraySize);

int main(int argc, char *argv[]) {
    char line[MAX_LINE];
    printf("Enter line: ");
    if (my_fgets(line, MAX_LINE) != NULL) {
        printf("You entered: %s", line);
    }

    return 0;
}
// fgets returns a pointer (which we cover later)

char *my_fgets(char array[], int arraySize) {

    int i = 0;
    while (i < arraySize - 1) {
        int ch = getchar();
        if (ch == EOF) {
            if (i == 0) {
                // no characters read
                // signal end-of-input
                return NULL;
            } else {
                array[i] = '\0';
                return array;
            }
        }
        array[i] = ch;
        if (ch == '\n') {
            array[i + 1] = '\0';
            return array;
        }
        i = i + 1;
    }

    // array full
    array[i] = '\0';
```

snap_line2.c

Reads lines of input until end-of-input
Print snap! if two consecutive lines are identical

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

#define MAX_LINE 4096

int main(int argc, char *argv[]) {
    char line[MAX_LINE];
    char lastLine[MAX_LINE];

    // read first line into array lastLine
    printf("Enter line: ");
    fgets(lastLine, MAX_LINE, stdin);

    printf("Enter line: ");
    while (fgets(line, MAX_LINE, stdin) != NULL) {

        if (strcmp(line, lastLine) == 0) {
            // lines are identical
            printf("Snap!\n");
        }

        strncpy(lastLine, line, MAX_LINE);

        printf("Enter line: ");
    }

    return 0;
}
```

print_argument_letters.c

print command line argument letters

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

int main(int argc, char *argv[]) {
    int i, j;
    i = 0;
    while (i < argc) {
        j = 0;
        while (argv[i][j] != '\0') {
            printf("argv[%d][%d]=%c\n", i, j, argv[i][j]);
            j = j + 1;
        }
        i = i + 1;
    }
    return 0;
}
```