Homework Packet

Homework Packet Contents

- Grading Policy
- Schedule
- ZIPS Module Concept
- Assignments
  - Exercise 1 : Simple program/worksheet to practice pointers
  - Exercise 2 : string parsing, structures, pointers
  - Exercise 3 : parsing, structures, typedefs, pointers
  - Exercise 4 : pointers, ASCII streams, binary streams
  - Exercise 5 : pointers to pointers, binary streams
  - Exercise 6 : array of pointers, function pointers, qsort(), bsearch()
  - Exercise 7 : void pointers, function pointers, sorting
  - Exercise 8 : ADT, linked lists, void pointers
  - Exercise 9 : signal handling, long jump

Grading Policy

General:

Solutions are due at the beginning of class. No late homework will be accepted. Source solutions are submitted prior to each class through the course web page. Hard copy only for exercise 1 and exercise 2. Solutions are expected to be ANSI compliant. Some exceptions will be made for non-ANSI library functions. Each student is expected to do his or her own work. Use of solutions from previous classes is strictly forbidden and will result in no credit. Challenges to grades are welcome.

Points:

5 Points Assignment is received on time and meets specified requirements.
Assignment correctly executes test case(s) specified in exercise handout.

3 Points Assignment is received on time and meets specified requirements.
Assignment correctly executes test case(s) specified in exercise handout.
Assignment does not implement the algorithm or code as specified in the exercise handout.

0 Points Assignment not received or received late.
Assignment does not correctly execute test case(s) specified in exercise handout.

Pass Criteria:

80% of 10 classes = 8 classes required for Satisfactory Completion.
80% of 45 points = 36 total points required for Satisfactory Completion.

Schedule

Class 1	March 29, 2006	
Assigned:	Exercise	**1**
	and Exercise	**2**
Class 2	April 5, 2006	
Assigned:	Exercise	**3**
Due:	Exercise	**1**
Class 3	April 12, 2006	
Assigned:	Exercise	**4**
Due:	Exercise	**2**
Class 4	April 19, 2006	
Assigned:	Exercise	**5**
Due:	Exercise	**3**
Class 5	April 26, 2006	
Assigned:	Exercise	**6**
Due:	Exercise	**4**
Class 6	May 3, 2006	
Assigned:	Exercise	**7**
Due:	Exercise	**5**
Class 7	May 10, 2006	
Assigned:	Exercise	**8**
Due:	Exercise	**6**
Class 8	May 17, 2006	
Assigned:	Exercise	**9**
Due:	Exercise	**7**
Class 9	May 24, 2006	
Assigned:	No Assignment	
Due:	Exercise	**8**
Class 10	May 31, 2006	
Assigned:	No Assignment	
Due:	Exercise	**9**

Zip Code Reporting System (ZIPS)

Exercises 3 - 9 are part of a project called the Zip Code Reporting System. The purpose of ZIPS is to read and write records from files and to operate on data structures containing these records. The ZIPS project will consist of multiple source files and include files.

The following files can be found at the course web site:

zips.c	ZIPS print function for exercises.
zips.h	ZIPS include file
ex3.c	main function for exercise 3.
zips3.c	modify this file to complete exercise 3.
qsort_of.h	include file for exercise 7
linklist.h	include file for exercise 8
linklist.c	double linked list functions for exercise 8
various data files	ASCII formatted test data input file

Do not modify these files except as noted in the exercise descriptions. For exercises 3 - 8, I will compile and test your program on my system. Since distributed files are not to be turned in, changing these files may mean that your exercise may not compile on my system.

Exercise 1

```
/*************************************************************************
 * Exercise 1
 * Following each operation in the short program below, please note the
 * values of inx, ptr, and *ptr. Initial examples are provided.
 *************************************************************************/
#include <stdio.h>

static void test( void )
{
int inx       = 0,
    * ptr    = NULL,
    values[] = { 1, 2, 3, 4, 5 };
                  /*****************************************/
                  /*  inx    |   ptr          |   *ptr     */
                  /*****************************************/
ptr = &values[1]; /*   0     *   &values[1]   *   2        */
                  /*****************************************/
++ptr;            /*   0     *   &values[2]   *   3        */
                  /*****************************************/
inx = *ptr;       /*         *                *            */
                  /*****************************************/
ptr -= 1;         /*         *                *            */
                  /*****************************************/
inx = *(ptr + 2); /*         *                *            */
                  /*****************************************/
*(ptr + 1) = -42; /*         *                *            */
                  /*****************************************/
inx = *(++ptr);   /*         *                *            */
                  /*****************************************/
inx = *(ptr++);   /*         *                *            */
                  /*****************************************/
*(ptr - 3) = -17; /*         *                *            */
                  /*****************************************/
inx = *(ptr - 3); /*         *                *            */
                  /*****************************************/
ptr -= 3;         /*         *                *            */
                  /*****************************************/
ptr += 5;         /*         *                *            */
                  /*****************************************/
}
int main( void )
{
  test();
}
/*************************************************************************
 *  Regarding pre-fix and post-fix operator:
 *      The pre-fix increments before the value is used
 *      The post-fix increments after the value is used
 *************************************************************************/
```

Exercise 2

OBJECTIVES: To understand string parsing concepts, difficulties, and stdlib support.
To understand structures, pointers, and pointers to structures.

Assignment: Complete ex2.c parse functions per functional descriptions provided with each function.

Turn in the following hard copy results at the start of class:

- ex2.c source file
- program execution output.

Considerations:

- Parse (per Webster): to separate a sentence into it's parts.

Exercise 3

OBJECTIVES: To understand string parsing concepts, difficulties, and stdlib support.
To understand structures, typedefs, pointers, and pointers to structures.
To create a function to be used in subsequent exercises.

Assignment:

1. Setup your compiler to compile each source file and to link the two objects into an executable. The program is made up of four source files: ex3.c, zips.c, zips3.c, and zips.h.
2. Create the *ZIPS_parse_zips_rec()* function found in zips3.c. This function parses an input string and initializes the fields of a structure. A sample test string is included in ex3.c.
3. Add appropriate tests to ex3.c test driver.

Submit the following to course web site:

- ex3.c source file.
- zips3.c source file.

Considerations:

- Your test driver should exercise the limits of each parameter (including the fields of the structure) and should test for valid and invalid scenarios.

Exercise 4

OBJECTIVES: To understand stream based file I/O. To understand the difference between ASCII and binary file I/O. To create a binary data file to be used in subsequent exercises.

Assignment: For this assignment you must create two new source files: zips4.c and ex4.c.

In zips4.c:

1. Write the *ZIPS_create_bin_from_ascii()* function. The functional description and prototype for this function are found in *zips.h*.

For ex4.c, write a main function that:

1. Makes on call to the *ZIPS_create_bin_from_ascii()* function to create zips.bin binary file from zips.txt ASCII file.
2. Reports success/failure to create binary file.
3. Reports number of records written to binary file.
4. Reports number of records skipped.

Submit the following to course web site:

- ex4.c source file
- zips4.c source file

Turn in the following hardcopy results at the start of class:

- a paragraph describing the tests you executed.

Considerations:

- *ZIPS_create_bin_from_ascii()* must robust enough to handle similarly formatted input files.
- The zips.txt file provided on the course web site must be processed without errors and without skipping any records.
- You should create other data files for you own use that test cases such as input files without any records, without any valid records, with mostly valid records, with some records whose field lengths exceed the target field length, more records then zips.txt, and less records then zips.txt.
- Your program should link object files create from ex4.c, zips4.c, zips3.c, and zips.c.
- *fwrite* return the number of objects written.

Exercise 5

OBJECTIVES: To understand stream based file I/O. To create a function to read and test the data binary file created in exercise 4.

Assignment: For this assignment you must create two new source files: zips5.c and ex5.c.

In zips5.c:

1. Write the *ZIPS_read_recs_from_bin()* function. The functional description and prototype for this function are found in *zips.h*.

In ex5.c, write a main function that:

1. reads the data from the binary file, *zips.bin* by calling *ZIPS_read_recs_from_bin().*
2. displays the data using the *ZIPS_print_recs().*
3. frees any allocated space.

Submit the following to course web site:

- ex5.c source file.
- zips5.c source file.

Considerations:

- *fread* returns the number of objects read.
- The data is read into a pointer to an array that has been allocated on the heap.
- No credit if memory is not successfully returned to calling functions.
- Your ZIPS_read_recs_from_bin() function must be able to read similarly formatted binary files (i.e. make no assumption about the length of the test data file).
- Your program should link ex5.o, zips5.o and zips.o.

Exercise 6

OBJECTIVES: To understand binary file I/O. To understand function pointers, void pointers, and pointers to pointers. To practice the implementation of *qsort* and *bsearch*.

Assignment: For this assignment you must create one new source files: ex6.c.

In ex6.c, write a main function that:

1. reads the data from the binary file, *zips.bin* by calling *ZIPS_read_recs_from_bin()*.
2. creates an array of pointers to ZIPS_data_t structures on the heap (i.e. using malloc) and based on the number of records read from the binary file. Initializes pointers to data read from binary file.
3. writes the data from the array of pointers to stdout using *ZIPS_print_recs()*.
4. sorts the array of pointers on the **zip_code** field using *qsort* and a sort comparison function of your own design
5. writes the data from the array of pointers to stdout using *ZIPS_print_recs()*.
6. report the distance between the following zip codes …

98101 (Seattle)	97217 (Portland)
55767 (Moose Lake)	99362 (Walla Walla)
33147 (Miami, FL)	87729 (Miami, NM)
99999 (Not in data)	36026 (Equality)
36026 (Equality)	99999 (Not in data)

   … by searching the array of pointers using *bsearch* and *ZIPS_compute_distance0*.
7. frees any allocated space.

Submit the following to course web site:

- ex6.c source file.

Exercise 6 is continued on the next page …

Exercise 6

Considerations:

- Call *ZIPS_read_recs_from_bin()* to read data from file in step 1.
- An array of pointers cannot be passed directory in to *ZIPS_print_recs()*.
- Your sort comparison function and key comparison should probably be *static* functions in ex6.c.
- *bsearch* returns a pointer to the object in the input array that matches key or NULL if none found and passes pointers to array objects to the comparison function. The key is passed as a pointer. Since your program will use an array of pointers, expect a pointer to pointer in your key comparison function.
- *qsort* returns void and passes pointers to array objects to the comparison function. Since your program will use an array of pointers, expect a pointer to pointer in your comparison function.
- Your program should link ex6.o, zips5.o, and zips.o.

Exercise 7

OBJECTIVES: To understand function pointers, void pointers, and pointers to pointers. To practice the implementation of a recursive function.

Assignment: For this assignment you must create one new source file: qsort_of.c

Implement your own version of the ANSI C library function *qsort*; call it *qsort_of*. Please consult your class handout (see example less11-3.c) for a quick sort algorithm. This function should mimic the ANSI qsort function and should have the same prototype and behavior as qsort.

To show that your *qsort_of* functions properly, write a test driver (ex7.c) that tests sorting various arrays including the following: array of integers, array of pointers to structures (as in exercise 6), array of structures, and array of doubles.

Submit the following to course web site:

- qsort_of.c source file
- ex7.c source file

Considerations:

- Any functions local to qsort_of() should be declared as static functions in the qsort_of.c source file.
- Use the qsort_of() prototype found in qsort_of.h.
- Generally, four tasks to convert less11-3 qsort_of() to ex7 qsort_of():
    1. Convert swap() to swap objects of *any* data types,
    2. Compare data with function referenced by function pointer,
    3. Convert *int* * to *void* * and *char* * + *size*
    4. Modify parameter lists
- Your test driver should function equally using qsort() or qsort_of().

Exercise 8

OBJECTIVES: To understand pointers, void pointers, abstract data types, and linked lists.

Assignment: Using the abstract linked list concepts and source in linklist.c and linklist.h, write a program called ex8.c that:

1. Calls ZIPS_read_recs_from_bin to get an array of ZIPS_data_t records.
2. Creates an linked list and enqueue each record from the array in the list.
3. prints the records to stdout by cycling the list.
4. frees any allocated space

Submit the following to course web site:

- ex8.c source file.

Considerations:

- use LIST_operate_on_list() to print the records.

Exercise 9

OBJECTIVES: to understand the implementation of signal handling, long jumps, and the assert macro.

ASSIGNMENT: Compile, link and execute the signal handling/long jmp example provided in class (less20.c). Write a paragraph describing the program.

Submit the following to course web site:

- ex9.txt: a paragraph describing behavior