

(3-2) File Processing with Functions

Instructor – Beiyu Lin
CptS 121 (May 13th, 2019)
Washington State University

Review – Functions

- What is a function?
 - Functions w/o input arguments
 - Functions w/ input arguments
-
- What is a pointer?
 - Read in file functions with pointers



Review – Functions w/o input

```
#include<stdio.h>  /* starting with including libraries*/
#include<stdlib.h>
double get_grade_point(void); /*declare a function*/

double get_grade_point(void)
{
    double grade_point = 0.0;
    printf("Please enter your grade point for your course:");
    scanf("%lf", &grade_point);
    return grade_point;
}
```



Review – Functions w/ input

```
#include<stdio.h> /* starting with including libraries*/
#include<stdlib.h>
double get_grade_point(void); /*declare a function*/

int sum_credit(int cred_class1, int cred_class2, int cred_class3)
{
    int sum_all_credit = 0;
    sum_all_credit = cred_class1 + cred_class2 + cred_class3;
    return sum_all_credit;

}
```



Review – Pointer

- What is a pointer?
- Example

```
int i, k;  
int *ip;
```

```
ip = &i;  
i = 100;  
k = *ip;  
k = k + 2;  
i = *(&k);  
*(&k) = 200;
```

- ip is a variable name
- ip is type “pointer to type int”
 - e.g. char *charPtr;
- &: reference operator -> returns pointer (address)
- *: dereference operator -> returns contents at address



Review – File Pointers

- File Pointers:
 - To read data from and write data to files
 - Data does not disappear when the program stops running
- C uses the data structure FILE for working with files
 - Working with files, use pointers to them, FILE *
- Most common file input/output (I/O) functions:
 - fopen()

For example:

```
FILE* infile = NULL;  
infile = fopen("gpa_file.txt", "r"); /*open the file and read the file*/
```



- fclose(), fgetc(), fputc(), fread(),
- fwrite()

Operation: “r” => read;
“w” => write (over write);
“a” => append at the end;

Why Files?

- Need to store data and information outside of a program
- Most real applications need to create, update, and/or delete data and information
- Easy to process and manipulate



Files and Streams in C (1)

- C views each file as a sequential stream of bits (1's and 0's) or bytes
- Each file ends with an end-of-file marker (EOF)
- Once a file is opened a stream is associated with it



Files and Streams in C (2)

- When a program starts execution, three files and associated streams are automatically opened
 - standard input (allows for us to get data from keyboard)
 - standard output (allows for us to write to the screen)
 - standard error
- Streams provide communication channels between files and programs



File Processing Algorithm

- Step 1: open the desired file
 - Opening is based on filename and permissions (read, write, or append)
 - Creates a new stream
- Step 2: process the file
 - Read data from the file
 - Does not affect file
 - Write data to the file
 - Completely overwrites existing file
 - Add data to the end of the file
 - Retains previous information in file
- Step 3: close the file
 - Destroys the stream



How to Get Started with Files in C?

- Before files may be manipulated, they must first be opened
 - Opening a file creates a communication channel between the file and the program
- Once a file is opened, several standard library functions are available to process file data and information
- Once all information and data associated with the file is no longer needed, it should be closed



File Functions in C

- Located in `<stdio.h>`
- Open a file:
 - `fopen ()` – returns a file handle to opened file
- Read from a file:
 - `fscanf ()`
- Write to a file:
 - `fprintf ()`
- Close a file:
 - `fclose ()` – closes file based on file handle



Review – File Pointers

- Most common file input/output (I/O) functions:
 - `fopen()`
e.g. `FILE* infile = fopen("gpa_file.txt", "r"); /* infile is a file pointer*/`
 - `fgetc()`
File pointer must be open for reading
e.g. `char ch = fgetc(infile);`
 - `fputc()`
Writes or appends the specified character to the pointed-to file.
e.g. `fputc("A", infile); /*write character A to the file*/`
 - `fscanf()`
Reads data from the the file
 - `fwrite()`
 - `fclose()`



Review – Read in file functions with pointers

/*define the function*/

```
void get_grade_point_infile(FILE* infile, int* class_id, int* class_credit, double* class_grade)
{
```

2

```
    fscanf(infile, "%d", class_id); /*similar as scanf*/
    fscanf(infile, "%d", class_credit); /*similar as scanf*/
    fscanf(infile, "%lf", class_grade); /*similar as scanf*/
```

```
}
```

```
int main(void)
```

```
{
```

1

```
    FILE* infile = NULL;
    infile = fopen("gpa_file.txt", "r"); /*open the file and read the file*/
```

```
    int class_id = 0, class_credit = 0;
    double class_grade = 0.0;
```

```
    /*call the function*/
```

```
    get_grade_point_infile(infile, &class_id, &class_credit, &class_grade);
```

3

```
    fclose(infile);
```

```
}
```

C. Hundhausen, A. O'Fallon, B. Lin



Problem Solving Example Revisited (1)

- Problem Statement: Write a program that computes your grade point average after completion of 3 courses.
- Inputs from a file:
 - Grade point and number of credits for course 1
 - Grade point and number of credits for course 2
 - Grade point and number of credits for course 3
- Outputs to a file:
 - Grade point average (GPA)
- Relevant formula:
$$\text{GPA} = ((\text{grade_point1} * \text{num_credits1}) + (\text{grade_point2} * \text{num_credits2}) + (\text{grade_point3} * \text{num_credits3})) / \text{total_num_credits}$$



Problem Solving Example (2)

- Initial algorithm
 - Open the data files
 - Get the grade points earned for each class from input file
 - Get the credit hours for each class from input file
 - Compute the average of the grade points
 - Write the results to output file
 - Close the files



Problem Solving Example (3)

- Refined algorithm

- Open the data files
 - Open one file with read permissions (input file)
 - Open one file with write permissions (output file)
- Get the grade points earned for each class from input file
- Get the credit hours for each class from input file
- Compute the total number of credits
 - $\text{total_num_credits} = \text{num_credits1} + \text{num_credits2} + \text{num_credits3};$
- Compute the credits hours earned
 - $\text{weighted_credits} = (\text{grade_point1} * \text{num_credits1}) + (\text{grade_point2} * \text{num_credits2}) + (\text{grade_point3} * \text{num_credits3});$
- Compute the average of the grade points
 - $\text{gpa} = \text{weighted_credits} / \text{total_num_credits};$
- Write the results to output file
 - Write `total_num_credits`
 - Write `weighted_credits`
 - Write `gpa`
- Close the files
 - Close the input file
 - Close the output file



Problem Solving Example (4)

- The GPA example revisited (now with file processing)

```
double get_grade_point (FILE *infile);
int get_credits (FILE *infile);
int compute_total_num_credits (int num_credits1, int num_credits2, int num_credits3);
double compute_weighted_credits (double grade_point1, double grade_point2, double grade_point3,
                                int num_credits1, int num_credits2, int num_credits3);
double compute_gpa (double weighted_credits, int total_num_credits);
void display_gpa (FILE * outfile, double weighted_credits, int total_num_credits, double gpa);

int main (void)
{
    int num_credits1 = 0, num_credits2 = 0, num_credits3 = 0;
    double grade_point1 = 0.0, grade_point2 = 0.0, grade_point3 = 0.0,
           weighted_credits = 0.0, total_num_credits = 0.0, gpa = 0.0;
    FILE * infile = NULL, *outfile = NULL; /* Variables that will allow for manipulation of our file streams */

    /* Need to open an input file and output file */
    infile = fopen ("input.txt", "r"); /* Input file opened with read permissions "r" */
    outfile = fopen ("output.txt", "w"); /* Output file opened with write permissions "w" */

    /* Get the grade points and credits from the input file */
    /* The input file, "input.txt", stores the grade point and number of credits for a class on separate lines */
    grade_point1 = get_grade_point (infile);
    num_credits1 = get_credits (infile);

    grade_point2 = get_grade_point (infile);
    num_credits2 = get_credits (infile);

    grade_point3 = get_grade_point (infile);
    num_credits3 = get_credits (infile);
```



Problem Solving Example (5)

```
/* Sum up the credits for each course */
total_num_credits = compute_total_num_credits (num_credits1, num_credits2,
                                                num_credits3);

/* Compute credit hours earned */
weighted_credits = compute_weighted_credits (grade_point1, grade_point2, grade_point3,
                                              num_credits1, num_credits2, num_credits3);

/* Compute gpa */
gpa = compute_gpa (weighted_credits, total_num_credits);

/* Write the results to output file, "output.txt" */
display_gpa (outfile, weighted_credits, total_num_credits, gpa);

/* Don't forget to close your files! */
fclose (infile);
fclose (outfile);

return 0;
```

```
}
```



Problem Solving Example (6)

- Definition of `get_grade_point ()`
/* Reads a grade point earned for a class from a file */

```
double get_grade_point (FILE *infile)
{
    double grade_point = 0.0;

    fscanf (infile, "%lf", &grade_point);

    return grade_point;
}
```



Problem Solving Example (7)

- Definition of `get_credits` ()

/* Reads the number of credits earned for a class from a file.

Precondition: the file referred to by `infile` must already be open.*/

```
int get_credits (FILE *infile)
{
    int num_credits = 0;

    fscanf (infile, "%d", &num_credits);

    return num_credits;
}
```



Problem Solving Example (8)

- Definition of `compute_total_num_credits` ()

/ Sums up the total number of credits earned for 3 courses */*

```
Int compute_total_num_credits (int num_credits1, int num_credits2,  
                               int num_credits3)  
{  
    int total_num_credits = 0;  
  
    total_num_credits = num_credits1 + num_credits2 + num_credits3;  
  
    return total_num_credits;  
}
```



Problem Solving Example (9)

- Definition of
`compute_weighted_credits ()`

```
double compute_weighted_credits (double grade_point1, double grade_point2,  
    double grade_point3, int num_credits1, int num_credits2, int num_credits3)  
{  
    double weighted_credits = 0.0;  
  
    weighted_credits = (grade_point1 * num_credits1) + (grade_point2 *  
        num_credits2) + (grade_point3 * num_credits3);  
  
    return weighted_credits;  
}
```



Problem Solving Example (10)

- Definition of `compute_gpa` ()

```
double compute_gpa (double weighted_credits, int total_num_credits)
{
    double gpa = 0.0;

    gpa = weighted_credits / total_num_credits;

    return gpa;
}
```



Problem Solving Example (11)

- Definition of `display_gpa` ()
/* Outputs the calculated values to a file */

```
void display_gpa (FILE *outfile, double weighted_credits,  
                 int total_num_credits, double gpa)  
{  
    fprintf (outfile, "Weighted Credits: %.2lf\n"  
            "Total Credits: %d\n"  
            "GPA: %.2lf\n", weighted_credits,  
            total_num_credits, gpa);  
}
```



Closing Thoughts on Files

- Files are required for many applications
- C has no direct support for random-access to data in files
 - Must handle data in files sequentially
- Files may be created and manipulated in any manner appropriate for an application



References

- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (8th Ed.)*, Addison-Wesley, 2016



Collaborators

- Chris Hundhausen

