ICS-103

Computer Programming in C

Chapter 6:

Pointers and Modular Programming

Dr. Tarek Ahmed Helmy El-Basuny

Outline of Ch. 06 Topics

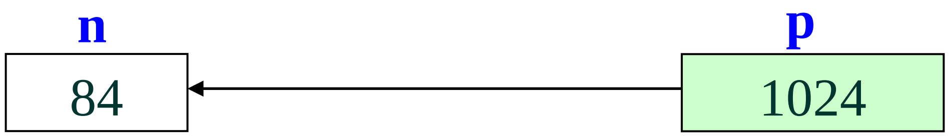
- Variables Vs. Pointers
- Pointer Variable Definition and Declaration
- Direct (&) and Indirect (*) Reference Operators
- Why Data Files?
- Declaring FILE Pointer Variable
- Opening data files for input/output
- Scanning from and printing to data files
- Handling File not found error
- EOF-controlled Loops
- Closing input and output files
- Functions with one Output Parameter,
  - Example of Call to Function with one Output Parameter,
- Functions with more than one Output Parameters,
  - Examples of Calls to Functions with many Output Parameters,
- Scope of Names,
  - Example of Names Scope,
- Common Programming Errors

What is a Pointer Variable?

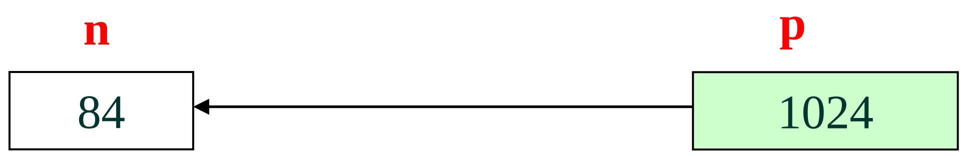
- **Variables** are simply names used to refer to some locations in memory.
- A **normal variable** directly contains a specific value.
- A **pointer variable** is a special variable that stores an address of a variable.
- If a pointer variable stores the address of a char variable, we call it a character pointer and so on.
- The **address/reference operator** & gives the "address of a **variable**" while the **indirection operator** * gives the "content of an address pointed to by a **pointer**"
- Pointers like any other variables must be declared before they can be used.
- A pointer variable is declared by preceding its name with an asterisk *.
  Example: int *p;
- How can we initialize p?
- First we must have an integer variable, then we use the & operator to get the address of the variable and assign it to p.

```
int n = 84;
p = &n;              (address operator &)
```

- Suppose that the int variable n is stored in the memory cell # 1024, then the following figure shows the relationship between n (variable) and p (pointer).



What is a Pointer Variable?

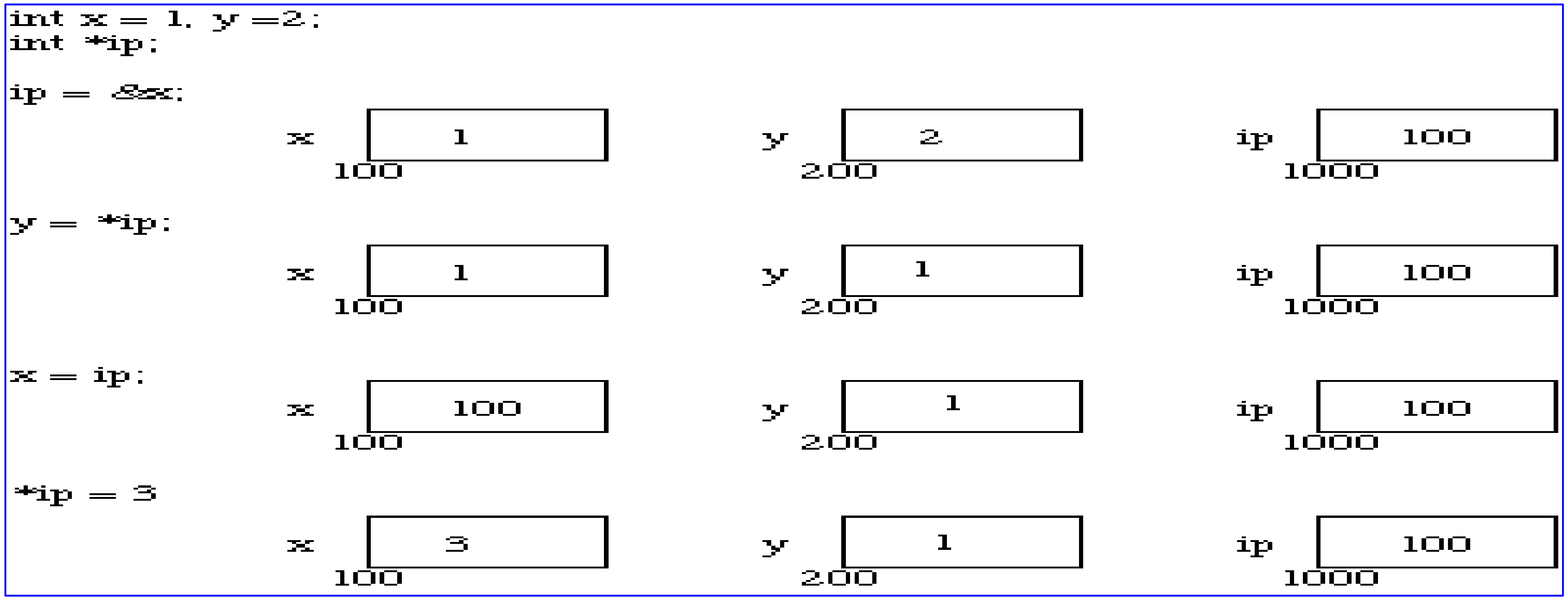


- A pointer variable such as p above, has two associated values:
- Its direct/reference value, which is referenced by using the name of the variable, &n.
  - If the address of the variable n in this example is 1024, then the direct value of the pointer will be 1024.
  - We can print the direct value of a pointer variable using printf by using %p as the place holder.
- Its indirect value, which is referenced by using the the **indirection/ Dereference** operator (*). So the indirect value of *p is 84.
- Multiple pointers require using a * before each variable declaration.

```
    int *myPtr1, *myPtr2;
```

- It is always a good practice to assign a NULL value to a pointer variable in case you **do not have an exact address to be assigned**.
- A pointer that is assigned NULL is called a **null** pointer.
- You may Initialize pointers to `0` **but** `NULL` is recommended.

Pointer & Variables: Example 1



```
int x = 1, y =2;
int *ip;

ip = &x;
```

```
y = *ip;
```

```
x = ip;
```

```
*ip = 3
```

- The assignments **x = 1** and **y = 2** obviously load these values into the variables.
- **ip** is declared to be a **pointer to an integer** and is assigned to the address of **x** (**&x**). So ip gets loaded with the value 100.
- Next **y** gets assigned to the **contents of ip**.
- Because **ip** points to memory location 100 (the location of **x)**.
- So **y** gets assigned to the values of **x,** which is 1.

Direct and Indirect value of a Pointer Variable

- The direct value of a pointer variable is the address of the variable it points to.
- The direct value of a pointer can be accessed by using address operator **&.**
  - *Syntax: &Variable-name.*
- The indirect value of a pointer variable is the content of the address it points to (the value of the variable it points to).
- The indirect value of a pointer can be accessed by using the indirection operator (*****). Syntax: **PointerVariable.*

  if int V = 101; and *P=&V;

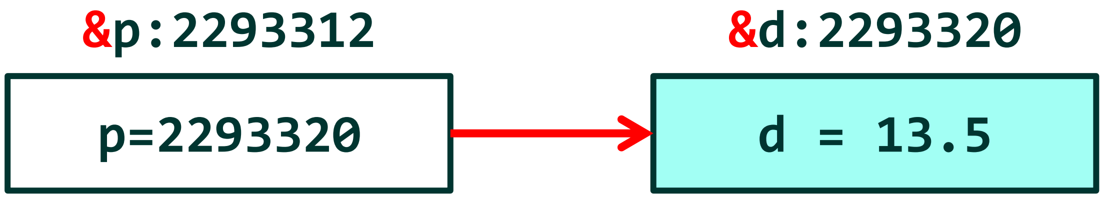
  then int *P = 101;

/* i.e. *P refers to the contents of the variable V (in this case, the integer 101) */

- **Pointers are used to:**
  - Point to input/output data files.
  - Change variables inside a function (reference parameters).
  - Remember a particular member of a group (such as an array).
  - Dynamic memory allocation (especially of arrays).
  - Build complex data structures (linked lists, stacks, queues, trees, etc.)

Example 1: Direct and Indirect value of a Pointer Variable

```
double d;
int *p;
d=13.5;
```



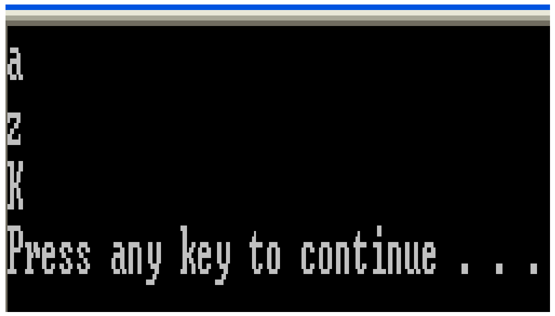
- Pointers contain address of a variable that has a specific value (indirect reference).
- Using a pointer variable p, one can access:

1. Its direct value: the value of pointer variable p.
   - In the above example, the value of p is 2293320.
     - It is the address of variable d (&d is 2293320)
2. Its indirect value: using the indirection operator *
   - In the example, *p is the value of d, which is 13.5.
3. Its address value: using the address operator &
   - In the example, &p is 2293312.

Example 2: Direct and Indirect value of a Pointer Variable

- This example demonstrates the relationship between direct and indirect value of a pointer variable.

```
#include<stdio.h>
int main(void) {
    char g='z';
    char c='a';
    char  *p;
    p=&c;   // p is pointing to the variable c
    printf("%c\n",*p);   // printing to the value of variable c
    p=&g;    // p is pointing to the variable g
    printf("%c\n",*p);   // printing to the value of variable g
    *p='K'; // changing the value of variable g to be K
    printf("%c\n",g);
    system("pause");
    return 0;            }
```

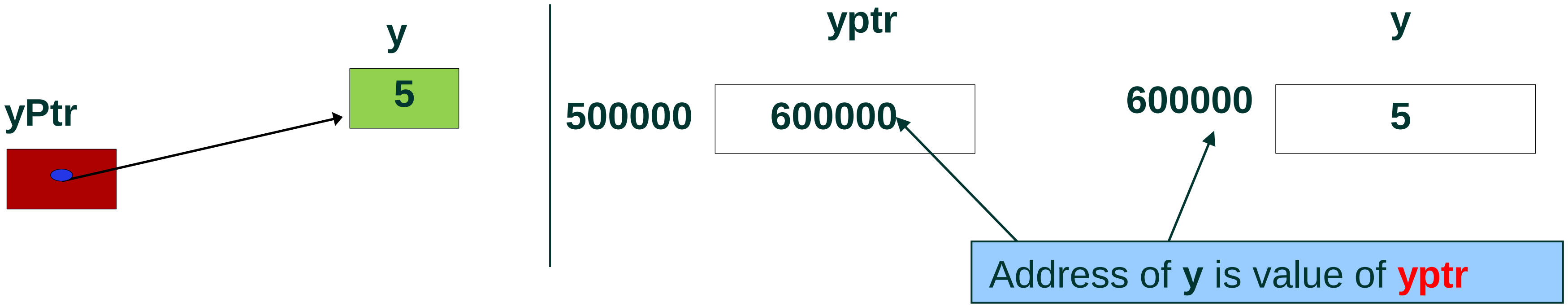


```
a
z
K
Press any key to continue . . .
```

Example 3: Direct and Indirect value of a Pointer Variable

- **& (address operator)**
  - Returns address of operand

```
int y = 5; /*declaring a variable y and assign 5 to it*/
int *yPtr; /*declaring yPtr a pointer to variable y */
yPtr = &y;       // yPtr gets address of y
```



- *** (indirection/dereferencing operator)**
  - Returns a name of what its operand points to.
  - `*yptr` returns `y` (because `yptr` points to `y`)
  - `*` can be used for assignment
    - Returns alias to an object

```
*yptr = 7;  // changes y to 7
```

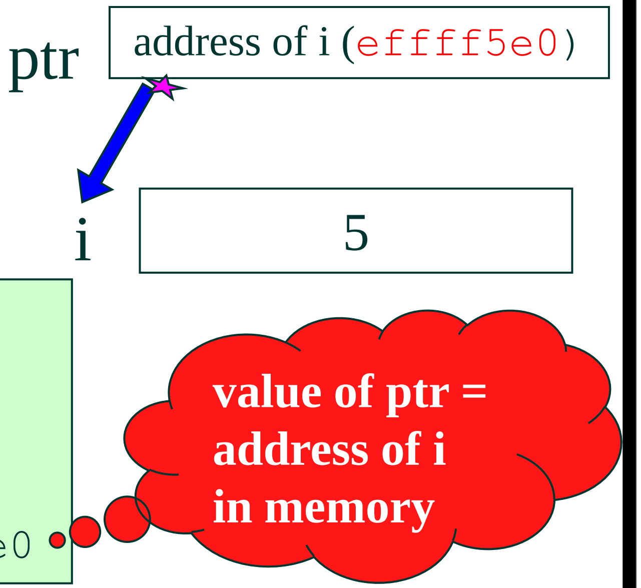
Example 4: Direct and Indirect value of a Pointer Variable

- ptr is a pointer variable storing an address of i
- ptr is NOT storing the actual value of i

```c
int i = 5;
int *ptr;
ptr = &i;
printf("i = %d\n", i);
printf("*ptr = %d\n", *ptr);
printf("ptr = %p\n", ptr);
```

Output:

```
i = 5
*ptr = 5
ptr = effff5e0
```



```c
/* Using the & and * operators */
#include <stdio.h>

int main()
{
   int a;         /* a is an integer */
   int *aPtr;     /* aPtr is a pointer to an integer */

   a = 7;
   aPtr = &a;     /* aPtr set to address of a */

   printf( "The address of a is %p"
           "\nThe value of aPtr is %p", &a, aPtr );

   printf( "\n\nThe value of a is %d"
           "\nThe value of *aPtr is %d", a, *aPtr );

   printf( "\n\nShowing that * and & are inverses of "
           "each other.\n&*aPtr = %p"
           "\n*&aPtr = %p\n", &*aPtr, *&aPtr );

   return 0;
}
```

The address of **a** is the value of **aPtr**.

The * operator returns an alias to what its operand points to. **aPtr** points to **a**, so ***aPtr** returns **a**.

Notice how * and & are inverses

```
The address of a is 0012FF88
The value of aPtr is 0012FF88
The value of a is 7
The value of *aPtr is 7
Proving that * and & are complements of each other.
&*aPtr = 0012FF88
*&aPtr = 0012FF88
```

- Declare variables
- Initialize variables
- Print
- Program Output

Example 6: Direct and Indirect value of a Pointer Variable

- We can access and modify a variable:
  - Either directly using the variable name
  - Or indirectly, using a pointer to the variable
- To refer to the *contents* of the variable that the pointer points to, we use indirection operator
  - Syntax: **PointerVariable*

    int V = 101;

    /* i.e. *P refers to the contents of the variable V (in this case, the integer 101) */
- Example:

```
double d = 13.5;

double *p = &d;     /* p = address of d */

*p = -5.3;          /* d = -5.3 */

printf("%.2f", d);    /* -5.30 */
```

Triple Use of * (Asterisk)

1. As a multiplication operator:

```
z  = x * y ;        /* z = x times y */
```

2. To declare pointer variables:

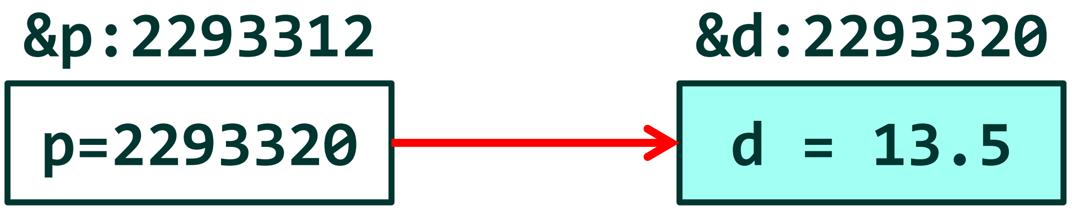
```
char ch;       /* ch is a character */
char *p;       /* p is pointer to char */
```

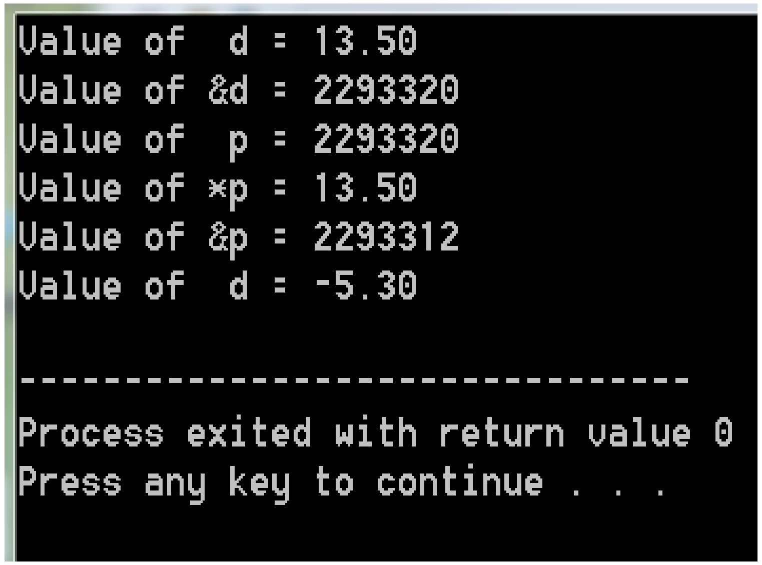
3. As an indirection operator:

```
 p = &ch;      /* p = address of ch */
*p = 'A';      /* ch = 'A' */
*p = *p + 1;  /* ch = 'A' + 1 = 'B' */
```

Example 7: Direct and Indirect value of a Pointer Variable

```c
#include <stdio.h>
int main(void)  {
  double d = 13.5;
  double *p;  /* p is a pointer to double variable*/
  p = &d;     /* p = address of d */
  printf("Value of  d = %.2f\n", d);
  printf("Value of &d = %d\n", &d);
  printf("Value of  p = %d\n", p);
  printf("Value of *p = %.2f\n", *p);
  printf("Value of &p = %d\n", &p);
  *p = -5.3;  /* d = -5.3 */
  printf("Value of  d = %.2f\n", d);
  return 0;
}
```





```
Value of  d = 13.50
Value of &d = 2293320
Value of  p = 2293320
Value of *p = 13.50
Value of &p = 2293312
Value of  d = -5.30


--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Why Data Files?

- So far, all our coded programs obtained their input from the keyboard by using scanf and displayed their output on the screen by using printf.
  - When a program is terminated, the entire data is lost. **If you need to get the data again then you need to run it again**.
    - Storing the results in a file will save your data even if the program terminates.
  - If you have to enter a large number of data, it will take a lot of time to enter them all. i.e. Processing large number of employees or student data.
  - However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.
  - Moreover, you can easily move your data from one computer to another without any changes.
  - There are applications where the output will be more useful if it is stored in a file for later processing.
- The good news is that C allows the programmer to use **data files**, both for input and output.
- **Data files** allow us to store information permanently and to access later on and alter that information whenever necessary.

Using Data Files

- **The process of using data files for input/output involves four steps as follows**:
  1. Declare input and output pointer variables of type `FILE *`.
  2. Open the files for **reading** or **writing** using **`fopen`** function.
  3. **Read** from the files using **`fscanf`** or **write** into the file using **`fprintf`**.
  4. Close the files after processing the data using **`fclose`**.
- Next, we explain how each of these steps will be implemented.

1- Declaring FILE Pointer Variables

- Declare FILE pointer variables to point to files as follows:

```
FILE *inp;  /* inp is a pointer to an input file */
FILE *outp; /* outp is a pointer to an output file */
```

- Note that the type **FILE** is in upper case.
  - The type FILE stores information about an opened file.
- Also note the use of * before a pointer variable.
  - inp and outp are pointer variables.
  - Recall that pointer variables store memory addresses.

2- Opening Data Files for Input/Output

- The second step is to open a file for reading or writing.
- Suppose our input data exists in file: `"data.txt"`
- **To open a file for reading, use the following:**

```
inp = fopen("data.txt", "r");
```

- The `"r"` indicates the purpose of reading from a file.
- Suppose we want to output data to: `"results.txt"`
- **To open a file for writing, use the following syntax**:

```
outp = fopen("results.txt", "w");
```

- The `"w"` indicates the purpose of writing to a file.
- You may define where the file will be created. i.e.

```
fopen("E:\\cprogram\\newprogram.txt","w");
fopen("E:\\cprogram\\oldprogram.bin",“r");
```

Opening Data Files for Input/output

- **Types of Files: T**here are two types of files you should know about:
  - Text files: The normal .txt files that you can easily create using Notepad or any simple text editors.
  - Binary files: Instead of storing data in plain text, they store it in the binary form (0's and 1's).

Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e., Data is added to end of file.	If the file does not exists, it will be created.
ab	Open for append in binary mode. i.e., Data is added to end of file.	If the file does not exists, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exists, it will be created.

Handling File NOT Found Error

- `inp = fopen("data.txt", "r");`
- **If the above `fopen` operation succeeds:**
  - It returns the **address of the open** `FILE` into `inp.`
  - The `inp` pointer can be used in all file read operations.
- **If the above `fopen` operation fails:**
  - For example, if the file `data.txt` is not found on disk.
  - It returns the **NULL** pointer value and assign it into `inp.`
- **We check the pointer `inp` immediately after `fopen`**

```
if (inp == NULL)
   printf("Cannot open file: data.txt\n");
```

2- Creating a File for Writing

- `outp = fopen("results.txt", "w");`
- If the above `fopen` operation succeeds:
  - It returns the address of the open `FILE` into `outp` pointer.
  - The `outp` pointer can be used in all file write operations
- If file `results.txt` does not exist on the disk
  - The OS typically creates a new file `results.txt` on disk.
- If file `results.txt` already exists on the disk
  - The OS typically clears its content to make it a new file.
- If `fopen` fails to create a new file for writing, then
  - It returns the NULL into `outp` pointer.

3- Input from & Output to Data Files

- Once we opened a file for reading or writing, The third step is:
  - To scan data from an input file, or
  - To print results into an output file.
- **To input a double value from file** `data.txt`, use:

```
fscanf(inp, "%lf", &data);
```

- The `fscanf` function works the same way as `scanf`.
  - Except that its first argument is an input `FILE` pointer
- **To output a double value to** `results.txt`, use:

```
fprintf(outp, "%f", data);
```

- Again, `fprintf` works similar to `printf`.
  - Except that its first argument is an output `FILE` pointer.

4- Closing Input and Output Files

- The final step in using data files is to close the files after you finish using them.
- The `fclose` function is used to close both input and output files as shown below:

```
fclose(inp);
fclose(outp);
```

- Warning: Do not forget to close files, especially output files.
  - This is necessary if you want to re-open a file for reading after writing data to it.
  - The OS might delay writing data to a file until it is closed.

Example: Program of File Input & Output

```
/* This program reads numbers from an input file “indata.txt”, formats and
writes each number on a separate line in an output file “outdata.txt”*/

#include <stdio.h>

int main(void) {
  FILE *inp;     /* pointer to input file */
  FILE *outp;    /* pointer to output file */
  double num;    /* number read */
  int status;    /* status of fscanf */

  /* Prepare files for input and output */
  inp = fopen("indata.txt", "r");
  outp = fopen("outdata.txt", "w");

  /* read each number, and then write it */
  status = fscanf(inp, "%lf", &num);
  while (status == 1) {
    fprintf(outp, "%.2f\n", num);
    status = fscanf(inp, "%lf", &num);
  }

  /* close the files */
  fclose(inp);
  fclose(outp);
  return 0;
}
```

Sample Run

- If the file: indata.txt contains

  344 55   6.3556   9.4   43.123   47.596

- Then the output file: outdata.txt will contain.

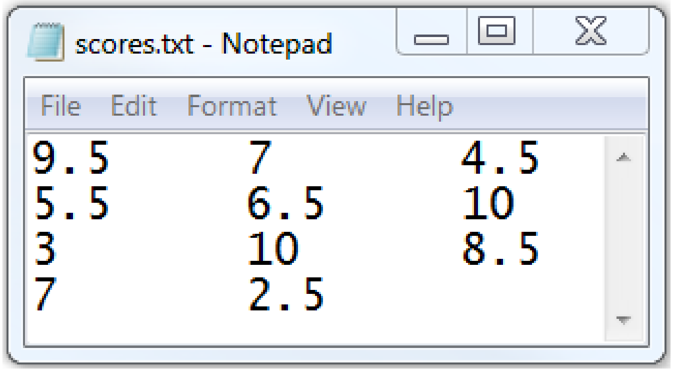
```
344.00
55.00
6.36
9.40
43.12
47.60
```

End-Of-File Controlled Loops

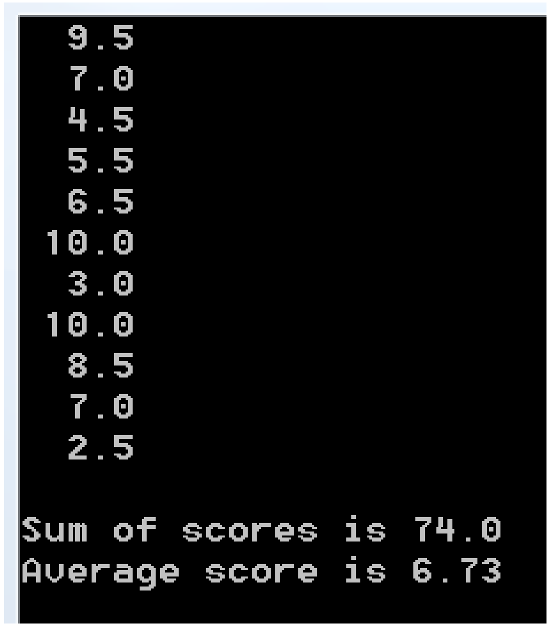
- When reading the input from a data file, **the program does not know how many data items to read.**
- Example: finding class average from student grades.
- The grades are read from an input file one at a time in a loop, until the end of file is reached.
- The question here is how to detect the end of file?
- **The good news is that**:
  - fscanf returns a special value, named EOF, when it encounters End-Of-File.
- We can take advantage of this by using EOF as a condition to control the termination of a loop.

Example: Reading from a file

```c
/* This program computes the average score of a class,
   The scores are read from an input file, scores.txt */
#include <stdio.h>
int main (void) {
   FILE *infile;
   double score, sum=0, average;
   int count=0, status;
   infile = fopen("scores.txt", "r");
   status = fscanf(infile, "%lf", &score);
   while (status != EOF)
   {
      printf("%5.1f\n", score);
      sum += score;
      count++;
      status = fscanf(infile, "%lf", &score);
   }
   average = sum / count;
   printf("\nSum of scores is %.1f\n", sum);
   printf("Average score is %.2f\n", average);
   fclose(infile);
   return 0;
}
```



```
9.5       7        4.5
5.5       6.5      10
3         10       8.5
7         2.5
```



```
  9.5
  7.0
  4.5
  5.5
  6.5
 10.0
  3.0
 10.0
  8.5
  7.0
  2.5

Sum of scores is 74.0
Average score is 6.73
```

Functions with one Output Parameter

❑ **So far, we know how to:**

- Call a function and pass input parameters to it. This is a call by value.
- Use the functions that returned zero or one value through the return statement
- See the following Example:

```c
#include <stdio.h>
/* function prototype*/
int max(int num1, int num2);
int main () {     /* main function */
/* local variable definition */
int a = 100;
int b = 200;
int ret;
/* calling a function to get max value */
ret = max(a, b);
printf( "Max value is : %d\n", ret );
return 0;
}
```

```c
/* function returning the max between
two numbers */
int max(int num1, int num2)
{
/* local variable declaration */
int result;
if (num1 > num2)
result = num1;
else
result = num2;
return result;
}
```

Functions with Many Output Parameters

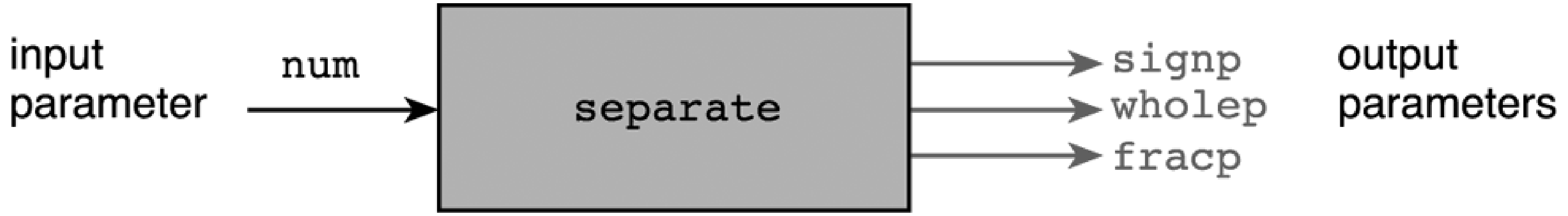
- Functions can also return more than one value:
  - To return more than one value, we need to **declare the** output parameters of the function **as pointers**.
  - When we use output parameters in functions, we declare those functions as void returning functions because they are not returning any value **by return statement** but they are going to update the values of the parameters in memory.
- Output parameters are pointer variables.
  - The caller passes the addresses of variables in memory.
  - The function uses indirect reference to modify variables in the calling function (for output results).
  - The function call in this case will be known as call by reference.

Example: swap two number

```
/* C Program to swap two numbers using pointers and function. */
#include <stdio.h>
void swap(int *n1, int *n2);
int main() {
int num1 = 5, num2 = 10;
// address of num1 and num2 is passed to the swap function
swap( &num1, &num2);
printf("Number1 = %d\n", num1);
printf("Number2 = %d", num2);
return 0;
}
void swap(int * n1, int * n2) {
 // pointer n1 and n2 points to the address of num1 and num2 respectively
int temp;
temp = *n1;
*n1 = *n2;
*n2 = temp;
}
```

Example: Function Separate

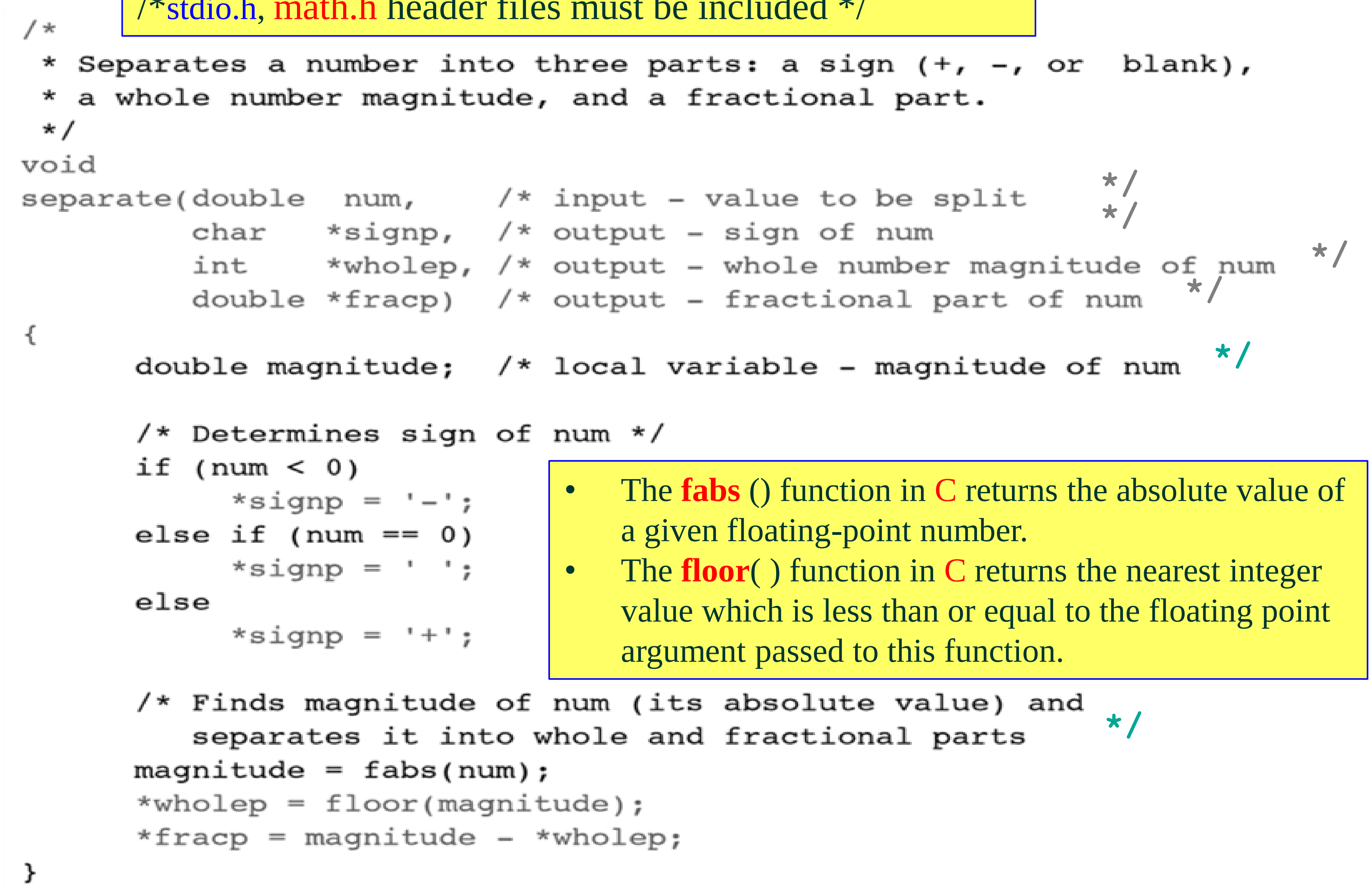
- Write a function that separates a number into a sign, a **whole**/Integer part, and a **fractional** part.
- The function has one input (a number) and returns many output values (sign, whole, fraction).
- In this case, we need to define the output parameters (sign, whole, fraction) as pointers.



```
void separate         /* function separate */
   (double num,       /* input number */
    char *signp,      /* sign pointer, output */
    int *wholep,      /* whole number pointer, output */
    double *fracp);   /* fraction pointer, output */
```

Example: Function Separate

/*stdio.h, math.h header files must be included */



```
/*
 * Separates a number into three parts: a sign (+, -, or  blank),
 * a whole number magnitude, and a fractional part.
 */
void
separate(double  num,     /* input - value to be split        */
         char    *signp,  /* output - sign of num             */
         int     *wholep, /* output - whole number magnitude of num  */
         double *fracp)   /* output - fractional part of num      */
{
      double magnitude;   /* local variable - magnitude of num    */

      /* Determines sign of num */
      if (num < 0)
           *signp = '-';
      else if (num == 0)
           *signp = ' ';
      else
           *signp = '+';

      /* Finds magnitude of num (its absolute value) and
         separates it into whole and fractional parts         */
      magnitude = fabs(num);
      *wholep = floor(magnitude);
      *fracp = magnitude - *wholep;
}
```

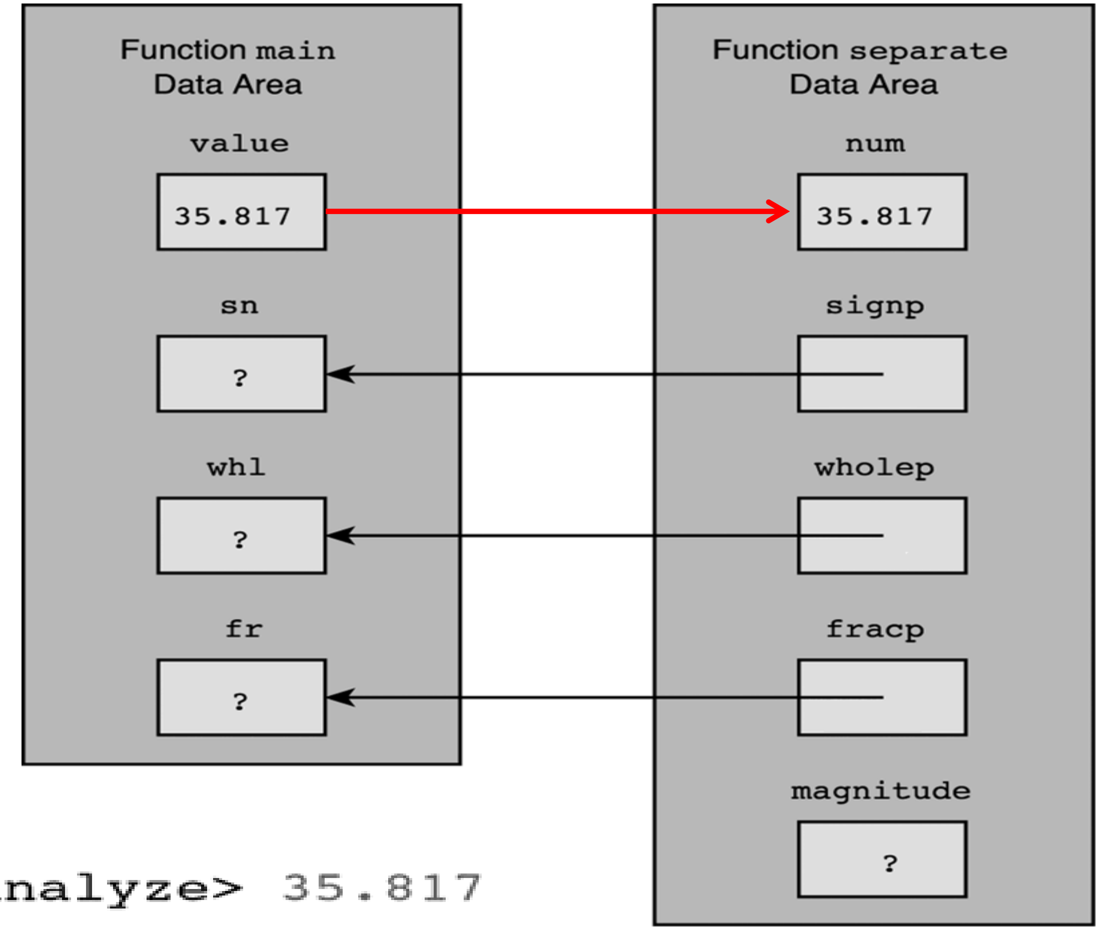
- The **fabs** () function in C returns the absolute value of a given floating-point number.
- The **floor**( ) function in C returns the nearest integer value which is less than or equal to the floating point argument passed to this function.

Calling the Function Separate

```c
int
main(void)
{
        double value; /* input - number to analyze     */
        char   sn;    /* output - sign of value        */
        int    whl;   /* output - whole number magnitude of value */
        double fr;    /* output - fractional part of value  */

        /* Gets data */
        printf("Enter a value to analyze> ");
        scanf("%lf", &value);

        /* Separates data value into three parts  */
        separate(value, &sn, &whl, &fr);

        /* Prints results  */
        printf("Parts of %.4f\n  sign: %c\n", value, sn);
        printf("  whole number magnitude: %d\n", whl);
        printf("  fractional part:  %.4f\n", fr);

        return (0);
}
```

Call the separate function here

Parameter Passing for Function Separate



```
Enter a value to analyze> 35.817
Parts of 35.8170
  sign: +
  whole number magnitude: 35
  fractional part: 0.8170
```

Programming Example

```
/* computes the area and circumference of a circle, given its radius */
#include <stdio.h>
void area_circum (double radius, double *area, double *circum);

int main (void) {
    double radius, area, circum ;
    printf ("Enter the radius of the circle > ") ;
    scanf ("%lf", &radius) ;

    area_circum (radius, &area, &circum) ;
    printf ("The area is %f and circumference is %f\n", area, circum) ;
    system("pause");
    return 0;
}

void area_circum (double radius, double *area, double *circum) {
    *area = 3.14 * radius * radius ;
    *circum = 2 * 3.14 * radius ;
}
```

Programming Example

```
/* Takes three integers and returns their sum, product and average */
#include<stdio.h>
void myfunction(int a,int b,int c,int *sum, int *prod, double *average);

int main (void) {
   int n1, n2, n3, sum, product;
   double av_g;
   printf("Enter three integer numbers > ");
   scanf("%d %d %d",&n1, &n2,&n3);
   myfunction(n1, n2, n3, &sum, &product, &av_g);
   printf("\nThe sum = %d\nThe product = %d\nthe avg = %f\n",sum, product, av_g);
            system("pause");
            return 0;
}

void myfunction(int a,int b,int c,int *sum,int *prod, double *average) {
     *sum=a+b+c;
     *prod=a*b*c;
     *average=(a+b+c)/3.0;
}
```

Programming Example

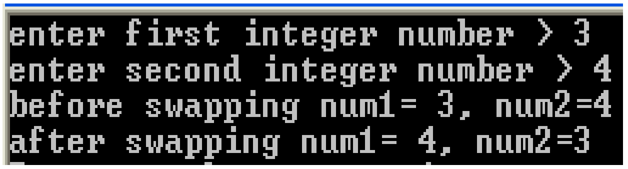
```
/* takes the coefficients of quadratic equation  a, b and c and returns its roots */
#include<stdio.h>
#include<math.h>

void quadratic(double a,double b, double c, double *root1, double *root2);

int main(void) {
    double a,b,c,r1,r2;
    printf("Please enter coefficients of the equation: [ a b c] > ");
    scanf("%lf%lf%lf",&a,&b,&c);

    quadratic(a,b,c,&r1,&r2);

    printf("\nThe first root is : %f\n",r1);
    printf("The second root is : %f\n", r2);
    system("pause");
    return 0;
}
```

```
void quadratic(double a,double b, double c,   double *root1, double *root2) {
    double desc;

    desc =b*b-4*a*c;
    if(desc < 0) {
      printf("No real roots\n");
      system("pause");
      exit(0);
    }
    else {
       *root1=(-b+sqrt(desc))/(2*a);
       *root2=(-b-sqrt(desc))/(2*a);
    }
}
```

Programming Example

```
/* swaps the values between 2 integer variables */
#include <stdio.h>
void readint(int *a, int * b);
void swap (int *a, int *b);
int main (void ) {
    int num1,num2;
    readint(&num1,&num2);
    printf("before swapping num1= %d,    num2=%d\
n",num1,num2);
    swap(&num1,&num2);
    printf("after swapping num1= %d, num2=%d\n",n
um1,num2);
    system("pause");
    return 0;
}
void readint (int *a, int *b) {
    printf("enter first integer number > ");
    scanf("%d",a);
    printf("enter second integer number > ");
    scanf("%d",b);
}
```

```
void swap (int *a, int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```



```
enter first integer number > 3
enter second integer number > 4
before swapping num1= 3, num2=4
after swapping num1= 4, num2=3
```

Because ***a*** and ***b*** are p
ointer variables, we d
o not use the & operat
or for scanf.

Example 2: Function Order

```
/* Arranges arguments in ascending order */
/* smp and lgp are pointer parameters */
/* Order variables pointed by smp and lgp */

void order(double *smp, double *lgp) {

  double temp;          /* temporary variable */

  /* compare variables pointed by smp and lgp */
  if (*smp > *lgp) {
    temp = *smp;        /* swap variables */
    *smp = *lgp;        /* pointed by smp and */
    *lgp = temp;        /* pointed by lgp */
  }
}
```

Multiple Calls to a Function

```c
#include <stdio.h>

void order(double *smp, double *lgp);

int
main(void)
{
        double num1, num2, num3; /* three numbers to put in order */

        /* Gets test data */
        printf("Enter three numbers separated by blanks> ");
        scanf("%lf%lf%lf", &num1, &num2, &num3);

        /* Orders the three numbers */
        order(&num1, &num2);
        order(&num1, &num3);
        order(&num2, &num3);

        /* Displays results  */
        printf("The numbers in ascending order are: %.2f %.2f %.2f\n",
               num1, num2, num3);

        return (0);
}
```

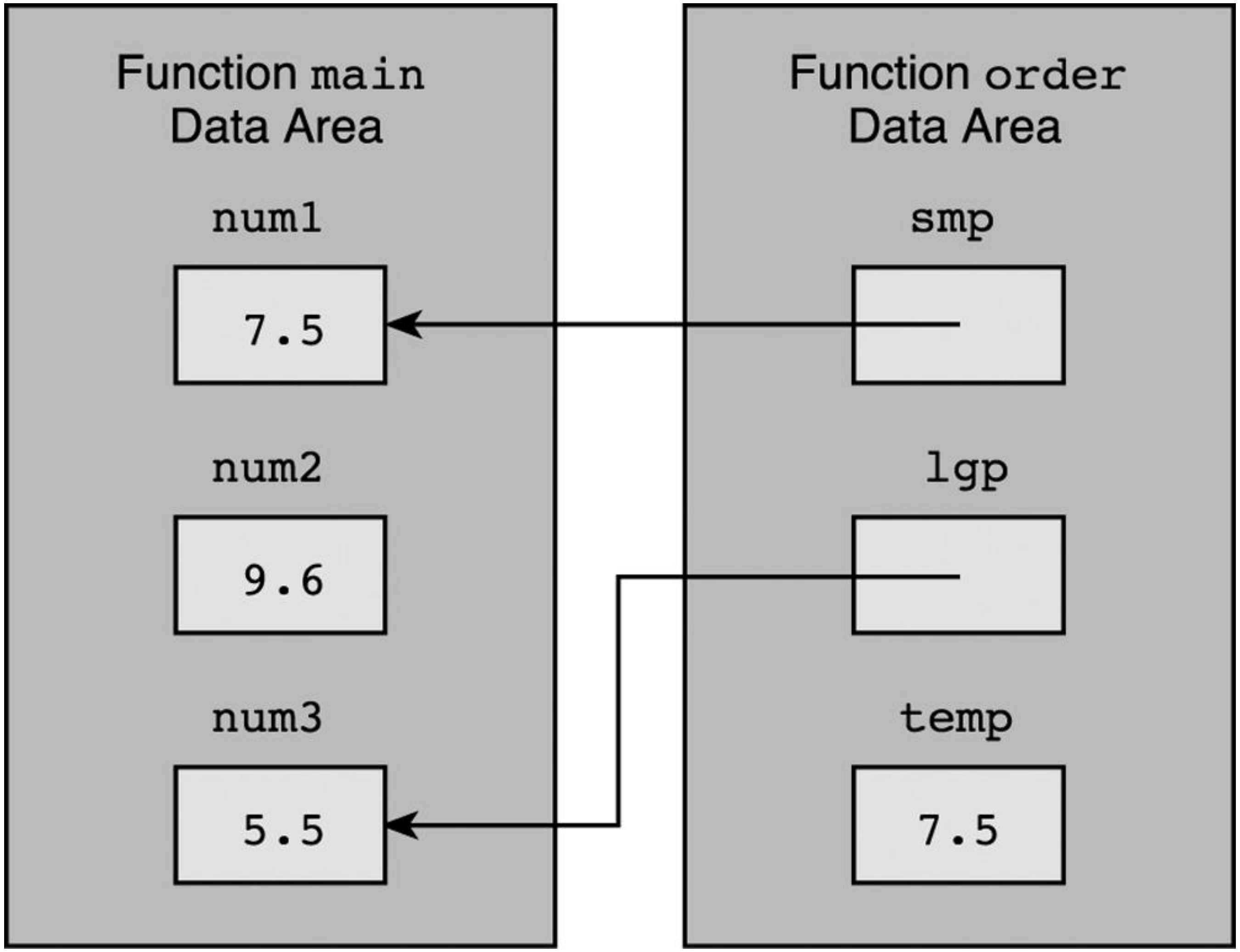
$\}$ SORTS 3 NUMBERS

Tracing Program: Sort 3 Numbers

Statement	num1	num2	num3	Effect
`scanf(. . .);`	7.5	9.6	5.5	Input Data
`order(&num1, &num2);`	7.5	9.6	5.5	No change
`order(&num1, &num3);`	5.5	9.6	7.5	swap num1, num3
`order(&num2, &num3);`	5.5	7.5	9.6	swap num2, num3
`printf(. . .);`				5.50 7.50 9.60

TRACE: order(&num1,&num3);

Data areas after: temp = *smp;



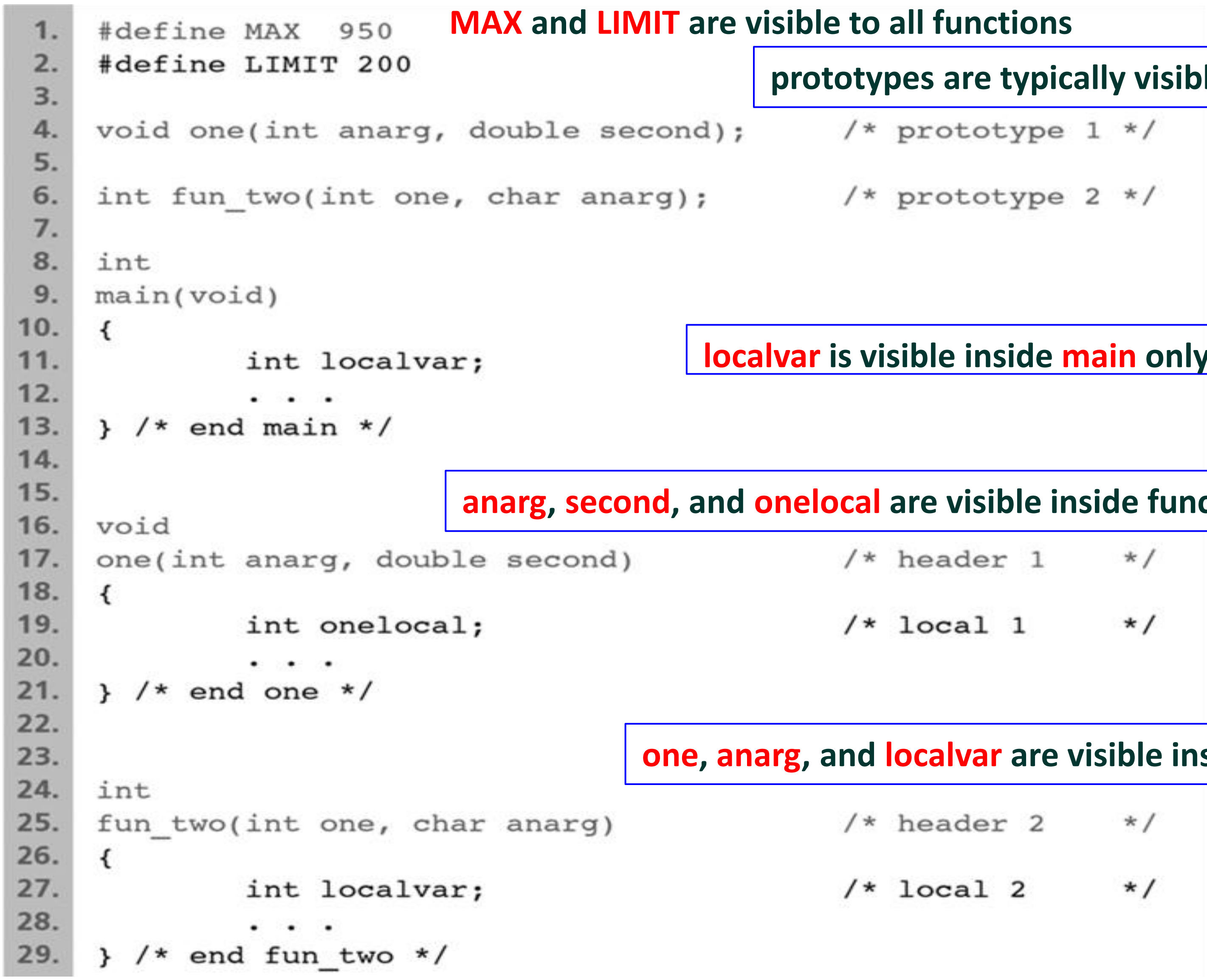
Outline of Ch. 06 Topics

- Variables Vs. Pointers
- Pointer Variable Definition and Declaration
- Direct (&) and Indirect (*) Reference Operators
- Why Data Files?
- Declaring FILE Pointer Variable
- Opening data files for input/output
- Scanning from and printing to data files
- Handling File not found error
- EOF-controlled Loops
- Closing input and output files
- Functions with one Output Parameter,
  - Example of Call to Function with one Output Parameter,
- Functions with more than one Output Parameters,
  - Examples of Calls to Functions with many Output Parameters,
- Scope of Names,
  - Example of Names Scope,
- Common Programming Errors

Scope of a Name

- Scope means the region of program where a name is visible/accessible.
  - Region of program where a name can be referenced or accessed.
- Scope of a name in: `#define NAME value`
  - From the definition line until the end of file.
  - Visible to all functions that appear after `#define`.
- Scope of a function prototype
  - Visible to all functions defined after the prototype.
- Scope of a parameter and a local variable
  - Visible only inside the function where it is defined.
  - Same name can be re-declared in different functions.

Scope of Names: Example



```
#define MAX   950
#define LIMIT 200

void one(int anarg, double second);        /* prototype 1 */

int fun_two(int one, char anarg);          /* prototype 2 */

int
main(void)
{
        int localvar;
          . . .
} /* end main */


void
one(int anarg, double second)              /* header 1    */
{
        int onelocal;                      /* local 1     */
          . . .
} /* end one */


int
fun_two(int one, char anarg)               /* header 2    */
{
        int localvar;                      /* local 2     */
          . . .
} /* end fun_two */
```

MAX and LIMIT are visible to all functions

prototypes are typically visible to all functions

localvar is visible inside main only

anarg, second, and onelocal are visible inside function one only

one, anarg, and localvar are visible inside fun_two only

Common Programming Errors

- **Be careful when using pointer variables**
  - A pointer should be initialized to a valid address before use.
  - De-referencing an invalid/NULL pointer is a runtime error.
- **Calling functions with output parameters**
  - Remember that output parameters are pointers.
  - Pass the address of a variable to a pointer parameter.
- **Do not reference names outside their scope.**
- **Create a file before reading it in a program.**
  - Remember that `fopen` prepares a file for input/output
  - The result of `fopen` should not be a `NULL` pointer
  - Check the status of `fscanf` to ensure correct input
  - Remember to use `fclose` to close a file, when done

The End!!

Thank you

Any Questions?