Pointers and Modular Programming

SDC OSW CSE 3541

Department of Computer Science & Engineering
ITER, Siksha 'O' Anusandhan Deemed To Be University
Jagamohan Nagar, Jagamara, Bhubaneswar, Odisha - 751030

Text Book(s)

Jeri R. Hanly & Elliot B. Koffman

Problem Solving and Program Design in C

Seventh Edition, Pearson Education

Robert Love

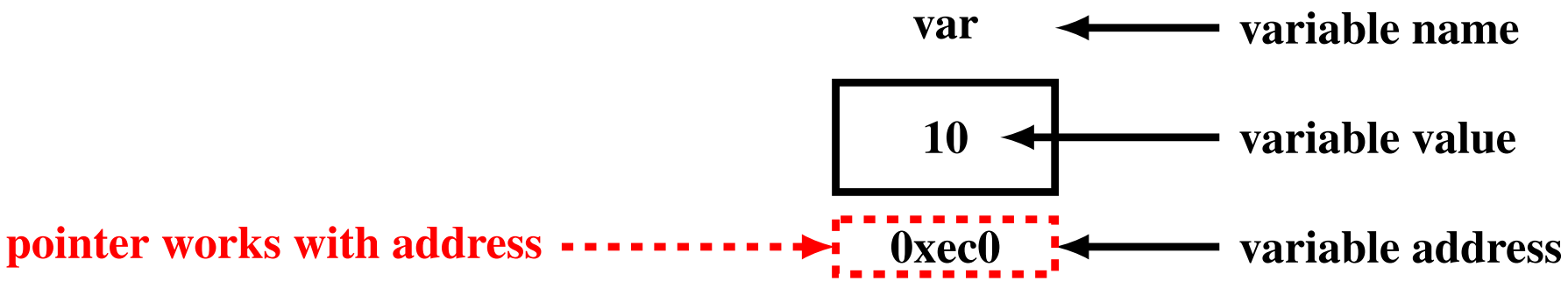
LINUX System Programming

Second Edition, SPD, O'REILLY

Talk Flow

Pointers in C

- When C was developed, computers were much less powerful and had much less memory to work with. The ability to work directly with particular memory locations was beneficial. Pointers in C allow you to effectively represent complex data structures, to change values passed as arguments to functions, to work with memory that has been dynamically allocated, and to more concisely and efficiently deal with arrays.
- Pointers are variables whose values are memory addresses.
- A variable name directly references a value, and a pointer indirectly references a value.
- Referencing a value through a pointer is called indirection.



Pointers and the Indirection Operator

Defining a Pointer Variable

- Lets assume we create a integer variable count as follows.

```
int count = 10;
```

- In above line of code the variable is created with value 10. We want to declare a pointer variable to store th address of the variable. The general syntax of pointer declaration is as follows:

```
<type> *<pointer variable name>;
```

- Hence, for integer the declaration will be as follows:

```
int *ptr;
```

- Here, **ptr** is the pointer varibale that can store the address of a integer memory location and can ***ptr** will represent the value in that location.

Pointers and the Indirection Operator (contd.)

Intializing a Pointer Variable

- In the above example we want to assign address of variable **count** to the pointer **ptr**.
- The ptr can be initialized in two ways as follows:
  1. At the time of declaration

```
int count = 10;
int *ptr = &count; // intialization
```

  2. Any where else in code block

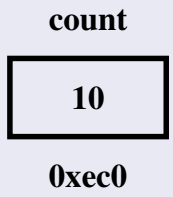
```
int count = 10;
int *ptr;
...
ptr = &count; // intialization
...
```

- Here, **&** operator extract the address of a variable. We have used it in **scanf()** function.

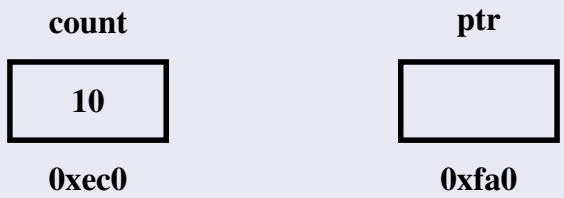
Pointers and the Indirection Operator (contd.)

Visualization

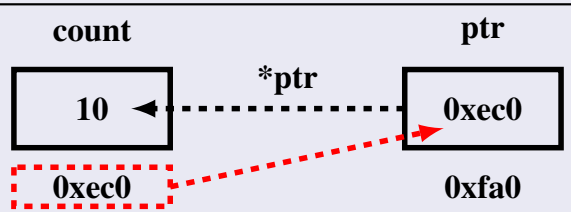
```
int count=10; // creation of count
```




```
int *ptr; // creation of the pointer
```




```
ptr = &count; // intialization
```



NB: only **ptr** refers to data stored in ptr which is the address of the variable **count**. ***ptr** means the data stored in the address location available in **ptr** (i.e. value available in variable **count**).

Pointers and the Indirection Operator (contd.)

Example code

```
#include<stdio.h>
int main(){
  int count = 10;
  int *ptr = &count;     // same as
                         // int *ptr;
                         // ptr = &count;
  printf("Value available in variable count = %d\n", count); // value of count
  printf("Address of the variable count = %p\n", &count); // address of count
  printf("Value available in variable ptr = %p\n", ptr);
                         // value in ptr will be same as address of count
  printf("Address of the variable ptr = %p\n", &ptr);  // address of ptr
  printf("value indirection of ptr is = %d\n", *ptr);
                         // value that ptr is pointing to will be same as value of count
  return(0);
}
```

Output

```
Value available in variable count = 10
Address of the variable count = 0x7ffe7bb20adc
Value available in variable ptr = 0x7ffe7bb20adc
Address of the variable ptr = 0x7ffe7bb20ad0
value indirection of ptr is = 10
```

Pointers and the Indirection Operator (contd.)

Indirect Reference

- In the above example the ***ptr** can be used to manipulate the variable **count**.
- When the unary indirection operator * is applied to a pointer variable, it has the effect of following the pointer referenced by its operand. This provides an **indirect reference** to the cell that is selected by the pointer variable.

Example (try yourself)

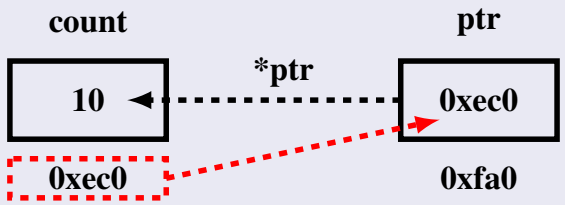
```
#include<stdio.h>
int main(){
        int count = 10; int *ptr = &count;
        *ptr = 20;
        printf("current value on count = %d\n", count);
        *ptr = 2 + *ptr;
        printf("current value on count = %d\n", count);
        *ptr = ++(*ptr);
        printf("current value on count = %d\n", count);
        return(0);
}
```

Pointers and the Indirection Operator (contd.)

Referencing and indirect referencing

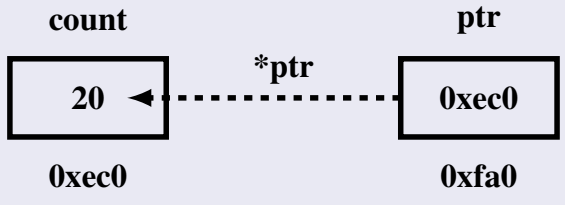
Initialization

```
int count = 10;
int *ptr;
ptr = &count;
```



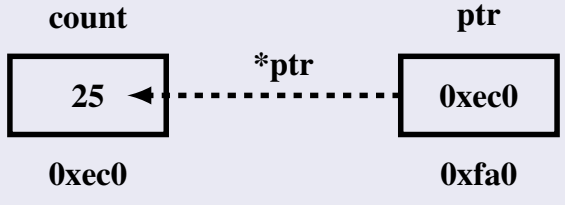
Direct reference

```
count = 20;
```

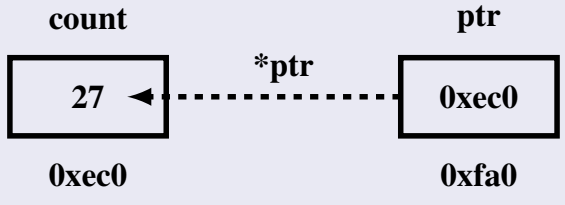


Indirect reference

```
*ptr = 25;
```




```
*ptr = 2 + *ptr;
```



Pointers and the Indirection Operator (contd.)

Pointers to file

- C allows a program to explicitly name a file from which the program will take input or write output.

```
FILE *inp; /* pointer to input file */
FILE *outp; /* pointer to ouput file */
```

- The calls to function fopen in the statements.

```
inp = fopen("distance.txt", "r");
outp = fopen("distout.txt", "w");
```

"r" in the first call to fopen indicates that we wish to read.
"w", indicating our desire to write to file.

- For the functions fscanf and fprintf, file equivalents of functions scanf and printf.

```
fscanf(inp, "%lf", &item);
fprintf(outp, "%.2f\n", item);
```

- Close file when no further use by calling function fclose with the file pointers.

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

Pointers and the Indirection Operator (contd.)

Example program

```
#include <stdio.h>
int main(void) {
  FILE *inp, *outp; /* pointer to input file */
  double item;
  int input_status; /* status value returned by fscanf */
  inp = fopen("indata.txt", "r"); // Prepare files for input or output
  outp = fopen("outdata.txt", "w");
  input_status = fscanf(inp, "%lf", &item); // Read each item, format it, and write it
  while (input_status == 1) {
        fprintf(outp, "%.2f, ", item);
        input_status = fscanf(inp, "%lf", &item);
  }
  fclose(inp);/* Close the files */
  fclose(outp);
  return (0);
}
```

Output

```
File indata.txt
344 55 6.3556 9.4

File outdata.txt
344.00, 55.00, 6.36, 9.40,
```

Scope Names

- The scope of a name refers to the region of a program where a particular meaning of a name is visible or can be referenced.
- Let's consider the following 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 */
```

Scope Names (contd.)

- Scope of Names in above program:

Name	Visible in one	Visible in fun_two	Visible in main
MAX	yes	yes	yes
LIMIT	yes	yes	yes
main	yes	yes	yes
localvar (in main)	no	no	yes
one (the function)	yes	no	yes
anarg (int)	yes	no	no
second	yes	no	no
onelocal	yes	no	no
fun_two	yes	yes	yes
one (formal parameter)	no	yes	no
anarg (char)	no	yes	no
localvar (in fun_two)	no	yes	no

Functions with Input/Output Parameters

Formal and actual parameter

- **Function Arguments :** If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function. Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.
- **Parameter**
  - A parameter is a special kind of variable, used in a function to refer to one of the pieces of data provided as input to the function to utilize.
  - These pieces of data are called arguments.
  - Parameters are Simply Variables.
- **Formal Parameter**
  - Parameter Written in Function Definition is Called "Formal Parameter".
  - Formal parameters are always variables, while actual parameters do not have to be variables.
- **Actual Parameter**
  - Parameter Written in Function Call is Called "Actual Parameter".
  - One can use numbers, expressions, or even function calls as actual parameters.

Functions with Input/Output Parameters (contd.)

Need of pointer

Let's assume the swapping of variable value in function as follows:

```
#include<stdio.h>
void swap(int var1, int var2)
{
        int temp;
        temp = var1;
        var1 = var2;
        var2 = temp;
        printf("After swapping the value in var1 :  %d (in function)\n", var1);
        printf("After swapping the value in var2 :  %d (in function)\n", var2);
}
int main(){
        int var1 = 5, var2=6;
        printf("Before swapping the value in var1 :  %d (in main)\n", var1);
        printf("Before swapping the value in var2 :  %d (in main)\n", var2);
        swap(var1, var2);
        printf("After swapping the value in var1 :  %d (in main)\n", var1);
        printf("After swapping the value in var2 :  %d (in main)\n", var2);
        return(0);
}
```

Functions with Input/Output Parameters (contd.)

Output

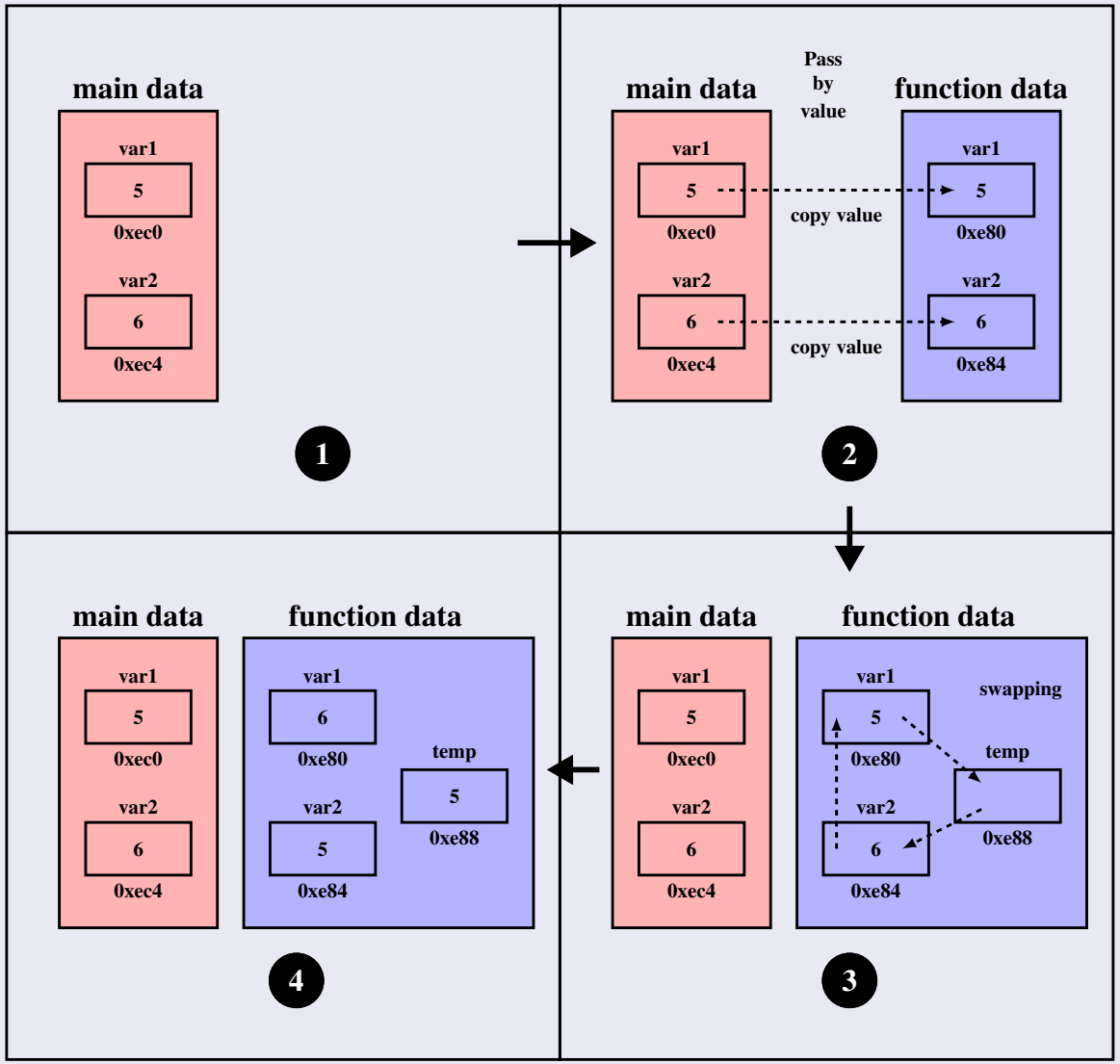
```
Before swapping the value in var1 :  5 (in main)
Before swapping the value in var2 :  6 (in main)
After swapping the value in var1 :  6 (in function)
After swapping the value in var2 :  5 (in function)
After swapping the value in var1 :  5 (in main)
After swapping the value in var2 :  6 (in main)
```

Explanaion

- In the above example. it can be observed that the values of variable in function have been swapped. However, the values available in variables of main have not been changed.
- This is caused due to locality of reference. We copy the variable of main to the local variable of function, hence the operation take place in terms of local variable. The model is called as **pass by value** model for function calling. The following figure illustrate the working steps o the above program.

Functions with Input/Output Parameters (contd.)

Illustration



- When a function is called it creates its local variable.
- The pass by value model copy the original data to the newly created variables which are local to function.
- The operation done in function affects the variables local to it.
- In such a case a single result can be returned. (A function can only return a single value.)
- After function call is completed the memory location is made free.

Functions with Input/Output Parameters (contd.)

Pass by reference model

```
#include<stdio.h>
void swap(int *ptr1, int *ptr2)
{
        int temp;
        temp = *ptr1;
        *ptr1 = *ptr2;
        *ptr2 = temp;
        printf("After swapping the value in var1 :  %d (in function)\n", *ptr1);
        printf("After swapping the value in var2 :  %d (in function)\n", *ptr2);
}
int main(){
        int var1 = 5, var2=6;
        printf("Before swapping the value in var1 :  %d (in main)\n", *ptr1);
        printf("Before swapping the value in var2 :  %d (in main)\n", *ptr2);
        swap(&var1, &var2);
        printf("After swapping the value in var1 :  %d (in main)\n", *ptr1);
        printf("After swapping the value in var2 :  %d (in main)\n", *ptr2);
        return(0);
}
```

Functions with Input/Output Parameters (contd.)

Output

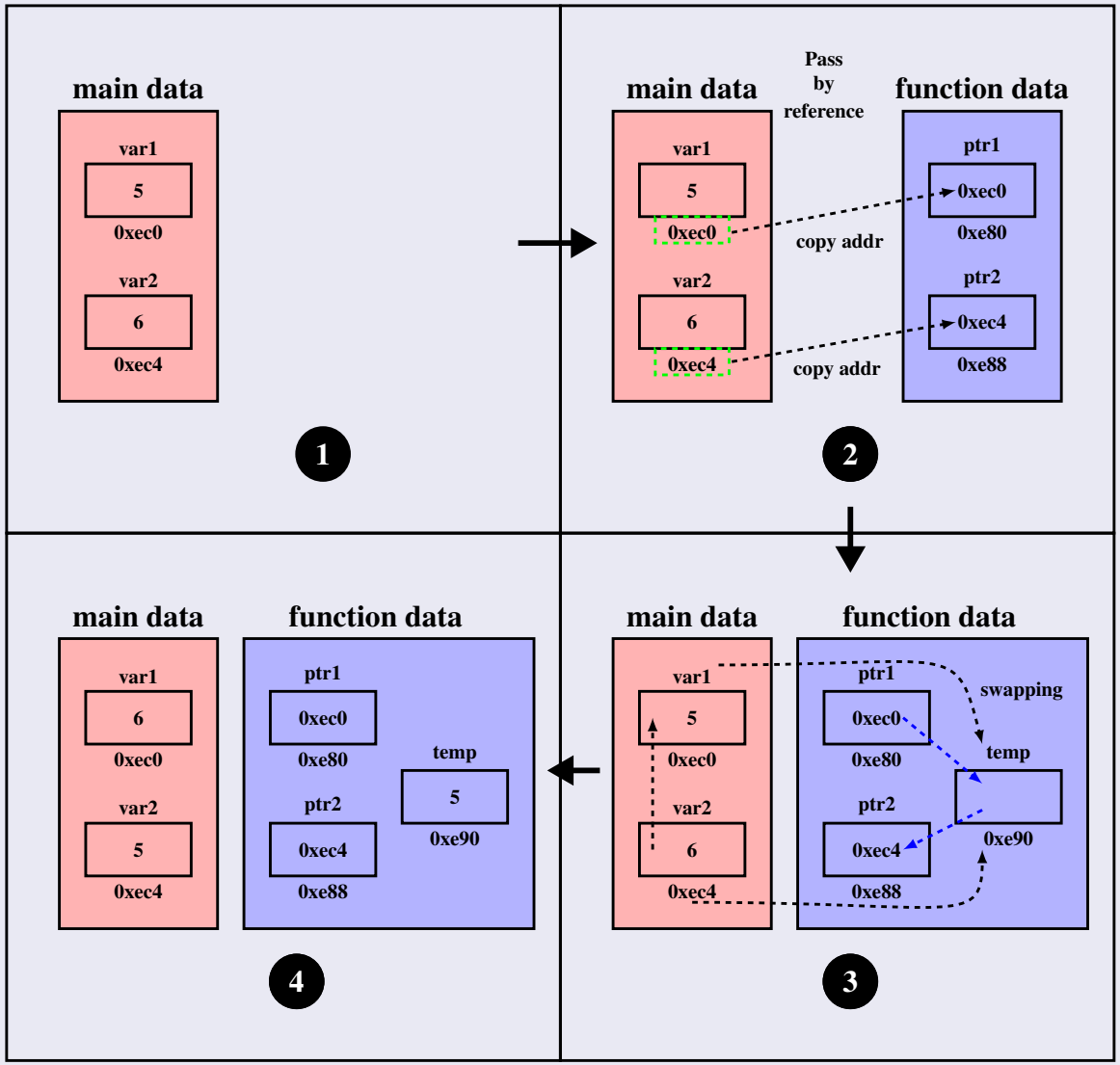
```
Before swapping the value in var1 :  5 (in main)
Before swapping the value in var2 :  6 (in main)
After swapping the value in var1 :  6 (in function)
After swapping the value in var2 :  5 (in function)
After swapping the value in var1 :  6 (in main)
After swapping the value in var2 :  5 (in main)
```

Explanaion

- In case of pass by reference model the address of the variables is passed.
- The manipulation in function takes place in terms of address indirection.
- Hence, any operation in function pointer reflected on the local variable of the main function.
- The following illustration shows the working of pass by reference model.

Functions with Input/Output Parameters (contd.)

Illustration



- Call to function passes the address instead of variable value.
- The *'s in the declarations of the function's formal parameters are part of the names of the parameters' data types. These *'s should be read as "pointer to".
- In function call & used to get the address of the variable.
- The pass by reference can be considered as both input and output to a function.

void pointers

- void pointers are pointers that point to a value that has no type (and thus also an undetermined length and undetermined de-reference properties).
- Use of void pointer example.

```
#include<stdio.h>
int main()
{
        int a=8;
        float b = 10.24;
        char c ='A';
        void *ptr;
        ptr = &a;
        printf("int value uing void ptr = %d\n", *(int *)ptr);
        ptr = &b;
        printf("float value uing void ptr = %f\n", *(float *)ptr);
        ptr = &c;
        printf("char value uing void ptr = %c\n", *(char *)ptr);
}
```

- Output:

```
int value uing void ptr = 8
float value uing void ptr = 10.240000
char value uing void ptr = A
```

Common Programming Errors

De-referencing pointer variable

"*" de-reference operator is missing in printf statement. This causes error and some junk values are getting displayed in output screen.

```
int *ptr;
int m = 4000;
ptr = &m;
printf("%d",ptr);
```

Wild Pointer

An uninitialized pointer is referred to as a wild pointer.

```
int * p;
//Pointer p is created and it contains garbage address
*p= 10;
//p will set the data value of garbage address to 10.
```

Common Programming Errors (contd.)

Assigning value to pointer variable

```
int *ptr;
int m = 100;
ptr = m;
printf("%d",*ptr);
```

An address should be assigned to the pointer. In the code, variable âĂŸmâĂŹ is assigned to pointer variable, which results in segmentation fault.

Address of uninitialized variable âĂŞ assigned to pointer

```
int *ptr;
int m;
ptr = &m;
printf("%d",*ptr);
```

In the code, variable 'm' is not initialized and address of variable 'm' is assigned to the pointer variable where the output is garbage value.

Common Programming Errors (contd.)

Comparing pointers

```
char str1[10],str2[10];
char *ptr1 = str1;
char *ptr2 = str2;
if(ptr1 > ptr2)
{
        printf("comparing pointers");
}
```

Pointer variables comparison is not valid, since they are stored in random memory locations. In the example provided below, if condition itself is error. Comparison is not possible between pointers.

De-referencing NULL pointer

```
int *ptr = NULL;
*ptr = 10;
```

In the code, pointer ptr is declared and initialized to NULL and "*ptr" which is the value in the address pointed by pointer is supposed to get overwritten as 10. Since ptr is not pointing to any location, we will get segmentation fault.

Practice Questions

Q1. Determine the following information about each value in a list of positive integers.

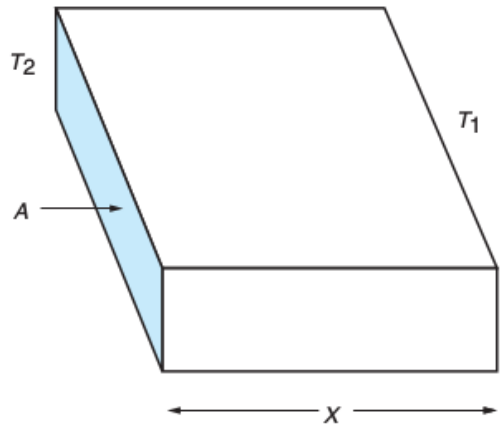
(a) Is the value a multiple of 7, 11, or 13?
(b) Is the sum of the digits odd or even?
(c) Is the value a prime number?

You should write a function with three type int output parameters that send back the answers to these three questions. Some sample input data might be: `104 3773 13 121 77 30751`

Q2. Develop a collection of functions to solve simple conduction problems using various forms of the formula

$$H = \frac{kA(T_2 - T_1)}{X}$$

where H is the rate of heat transfer in watts, k is the coefficient of thermal conductivity for the particular substance, A is the cross-sectional area in m^2 (square meters), T_2 and T_1 are the kelvin temperatures on the two sides of the conductor, and X is the thickness of the conductor in meters.



Practice Questions (contd.)

Define a function for each variable in the formula. For example, function calc_h would compute the rate of heat transfer, calc_k would figure the coefficient of thermal conductivity, calc_a would find the cross-sectional area, and so on.
Develop a driver function that interacts with the user in the following way:

```
Respond to the prompts with the data known. For the
unknown quantity, enter a question mark (?).
Rate of heat transfer (watts) >> 755.0
Coefficient of thermal conductivity (W/m-K) >> 0.8
Cross-sectional area of conductor (m^2) >> 0.12
Temperature on one side (K) >> 298
Temperature on other side (K) >> ?
Thickness of conductor (m) >> 0.003
          kA (T2 - T1)
H = ---------------
              X
Temperature on the other side is 274 K.
H = 755.0 W
T2 = 298 K
k = 0.800 W/m-K
T1 = 274 K
A = 0.120 m^2
X = 0.0003 m
```

Practice Questions (contd.)

Q3. The square root of a number N can be approximated by repeated calculation using the formula;

$$NG = 0.5(LG + N/LG)$$

where *NG* stands for next guess and *LG* stands for last guess. Write a function that calculates the square root of a number using this method. The initial guess will be the starting value of *LG*. The program will compute a value for *NG* using the formula given. The difference between *NG* and *LG* is checked to see whether these two guesses are almost identical. If they are, *NG* is accepted as the square root; otherwise, the next guess (*NG*) becomes the last guess (*LG*) and the process is repeated (another value is computed for *NG*, the difference is checked, and so on). The loop should be repeated until the difference is less than 0.005. Use an initial guess of 1.0.
Write a driver function and test your square root function for the numbers 4, 120.5, 88, 36.01, 10,000, and 0.25

Practice Questions (contd.)

Q4. InternetLite Corporation is an Internet service provider that charges customers a flat rate of $7.99 for up to 10 hours of connection time. Additional hours or partial hours are charged at $1.99 each. Write a function charges that computes the total charge for a customer based on the number of hours of connection time used in a month. The function should also calculate the average cost per hour of the time used (rounded to the nearest cent), so use two output parameters to send back these results. Write a second function round_money that takes a real number as an input argument and returns as the function value the number rounded to two decimal places. Write a main function that takes data from an input file usage.txt and produces an output file charges.txt . The data file format is as follows:
Line 1: current month and year as two integers
Other lines: customer number (a 5-digit number) and number of hours used
Here is a sample data file and the corresponding output file:

```
Data file  usage.txt
10 2009
15362 4.2
42768 11.1
11111 9.9
```

```
Output file charges.txt
Charges for 10/2009

          | Hours | Charge   | Average
Customer  | used  | per hour | cost
-----------------------------------
15362     | 4.2   | 7.99     | 1.90
42768     | 11.1  | 11.97    | 1.08
11111     | 9.9   | 7.99     | 0.81
```

Practice Questions (contd.)

Q5. When an aircraft or an automobile is moving through the atmosphere, it must overcome a force called drag that works against the motion of the vehicle. The drag force can be expressed as

$$F = \frac{1}{2} CD \times A \times \rho \times A^2$$

where F is the force (in newtons), CD is the drag coefficient, A is the projected area of the vehicle perpendicular to the velocity vector (in m^2), is the density of the gas or fluid through which the body is traveling (kg/m^3), and V is the body's velocity. The drag coefficient CD has a complex derivation and is frequently an empirical quantity. Sometimes the drag coefficient has its own dependencies on velocities: For an automobile, the range is from approximately 0.2 (for a very streamlined vehicle) through about 0.5. For simplicity, assume a streamlined passenger vehicle is moving through air at sea level (where $\rho = 1.23 kg/m^3$). Write a program that allows a user to input A and CD interactively and calls a function to compute and return the drag force. Your program should call the drag force function repeatedly and display a table showing the drag force for the input shape for a range of velocities from 0 m/s to 40 m/s in increments of 5 m/s.

Q6. Write a program to model a simple calculator. Each data line should consist of the next operation to be performed from the list below and the right operand. Assume the left operand is the accumulator value (initial value of 0). You need a function **`scan_data`** with two output parameters that returns the operator and right operand scanned from a data line. You need a function **`do_next_op`** that performs the required operation. **`do_next_op`** has two input parameters (the operator and operand) and one input/output parameter (the accumulator). The valid operators are:

Practice Questions (contd.)

```
+    add
-    subtract
*    multiply
/    divide
^    power (raise left operand to power of right operand)
q    quit
```

Your calculator should display the accumulator value after each operation. A sample run follows.

```
+ 5.0
result so far is 5.0
^ 2
result so far is 25.0
/ 2.0
result so far is 12.5
q 0
final result is 12.5
```

THANK YOU