C Programming Practice Solutions

Page 1 Solutions

1. Describe call by reference and write a program to swap two numbers.

- **Concept:** In call by reference, we pass the *address* (memory location) of the variables to the function. Any changes made inside the function directly affect the original variables.

Code:
```
#include <stdio.h>

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    int a = 10, b = 20;
    swap(&a, &b);
    printf("Swapped: a = %d, b = %d\n", a, b);
    return 0;
}
```

-

2. Difference between call by value and call by reference? Illustrate with example.

- **Call by Value:** Passes a *copy* of the data. Changes inside function do *not* affect the original.
- **Call by Reference:** Passes the *address*. Changes *do* affect the original.

Example:
```
void modify(int val, int *ref) {
    val = 50;      // Changes copy only
    *ref = 50;     // Changes actual variable
}
```

-

3. Program using pointer-to-pointer (double pointer) to print value and addresses.

```
#include <stdio.h>
int main() {
    int var = 100;
    int *ptr = &var;
    int **pptr = &ptr;

    printf("Value of var: %d\n", var);
    printf("Address of var: %p\n", &var);
    printf("Address stored in ptr: %p\n", ptr);
    printf("Address of ptr: %p\n", &ptr);
    printf("Address stored in pptr: %p\n", pptr);
    printf("Value via pptr: %d\n", **pptr);
    return 0;
}
```

4. Program using function pointer to perform subtraction.

```
#include <stdio.h>
int subtract(int a, int b) { return a - b; }

int main() {
    int (*func_ptr)(int, int) = subtract; // Pointer to function
    int result = func_ptr(10, 4);
    printf("Result: %d\n", result);
    return 0;
}
```

5. Explain function pointers with an example of multiplication.

- **Explanation:** A function pointer stores the address of a function, allowing you to call that function dynamically.

Example:

```
#include <stdio.h>
int multiply(int a, int b) { return a * b; }
int main() {
    int (*ptr)(int, int) = multiply;
    printf("Product: %d\n", ptr(5, 6));
    return 0;
}
```

-

6. Program that displays all command-line arguments.

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Arguments passed:\n");
    for (int i = 0; i < argc; i++) {
        printf("Arg %d: %s\n", i, argv[i]);
    }
    return 0;
}
```

7. Explain preprocessor directives and create a macro for square.

- **Explanation:** Directives starting with `#` are processed before compilation. They handle file inclusion and text substitution.

Macro:

```
#include <stdio.h>
#define SQUARE(x) ((x) * (x))
int main() {
    printf("Square of 5: %d", SQUARE(5));
    return 0;
}
```

-

8. Short note on #include, #define, and #ifdef.

- `#include`: Pastes contents of another file (e.g., `<stdio.h>`).
- `#define`: Creates constants or macros (e.g., `#define PI 3.14`).
- `#ifdef`: Conditional compilation; compiles code only if a macro is defined.

9. Program that dynamically allocates array and resizes using realloc.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *arr = (int*)malloc(2 * sizeof(int)); // Size 2
    arr[0] = 1; arr[1] = 2;

    arr = (int*)realloc(arr, 3 * sizeof(int)); // Resize to 3
    arr[2] = 3;

    printf("Values: %d %d %d\n", arr[0], arr[1], arr[2]);
    free(arr);
    return 0;
}
```

10. Explain calloc() and realloc().

- `calloc(n, size)`: Allocates memory for `n` items of `size` bytes and initializes them to zero.
- `realloc(ptr, new_size)`: Changes the size of previously allocated memory pointed to by `ptr`.

11. Program to store 10 strings using dynamic memory and print them.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
    char *strings[10];
    for(int i=0; i<10; i++) {
        strings[i] = (char*)malloc(20 * sizeof(char)); // Alloc memory
        sprintf(strings[i], "String %d", i+1);
    }
    for(int i=0; i<10; i++) printf("%s\n", strings[i]);
    return 0;
}
```

12. Program to copy contents of one file into another.

```
#include <stdio.h>
int main() {
    FILE *source = fopen("src.txt", "r");
    FILE *dest = fopen("dest.txt", "w");
    char c;
    if (source && dest) {
        while ((c = fgetc(source)) != EOF) fputc(c, dest);
    }
    fclose(source); fclose(dest);
    return 0;
}
```

13. Explain file modes: "r", "w", "a", "r+", "w+".

- `"r"`: Read only.
- `"w"`: Write only (overwrites file).
- `"a"`: Append (adds to end).
- `"r+"`: Read and Write (file must exist).
- `"w+"`: Read and Write (overwrites file).

14. Program using fgetc() to read file char-by-char.

```
#include <stdio.h>
int main() {
    FILE *fp = fopen("data.txt", "r");
    char c;
    if (fp) {
        while ((c = fgetc(fp)) != EOF) printf("%c", c);
        fclose(fp);
    }
    return 0;
}
```

15. Program using fputc() to write a string char-by-char.

```
#include <stdio.h>
int main() {
    FILE *fp = fopen("output.txt", "w");
    char str[] = "Hello";
    for(int i=0; str[i]!='\0'; i++) fputc(str[i], fp);
    fclose(fp);
    return 0;
}
```

16. Explain call by reference with example. (See Answer #1). It involves passing the address of a variable to modify the original value directly.

17. Chain of pointers (3 levels) program.

```
#include <stdio.h>
int main() {
    int a = 10;
    int *p1 = &a;
    int **p2 = &p1;
    int ***p3 = &p2;
    printf("Value: %d\n", ***p3);
    printf("Addr a: %p, Addr p1: %p, Addr p2: %p\n", &a, &p1, &p2);
    return 0;
}
```

18. Function pointer for addition.

```
#include <stdio.h>
int add(int a, int b) { return a + b; }
int main() {
    int (*sum)(int, int) = add;
    printf("Sum: %d", sum(10, 20));
    return 0;
}
```

19. Purpose of argc and argv.

- `argc` (Argument Count): Number of command-line arguments.
- `argv` (Argument Vector): Array of strings containing the arguments.

20. Define preprocessor. Program for macro substitution.

- **Def:** A system that processes code before the compiler sees it.

Code:
```
#define MSG "Hello World"
#include <stdio.h>
int main() { printf("%s", MSG); return 0; }
```

-

21. Dynamic Memory Definition. Realloc syntax. Store 5 student names.

- **Def:** Allocating memory manually during runtime using heap.
- **Realloc:** `ptr = realloc(ptr, new_size);`

Code:
```
#include <stdio.h>
#include <stdlib.h>
int main() {
    char *students[5];
    for(int i=0; i<5; i++) {
        students[i] = malloc(50);
        printf("Enter name %d: ", i+1);
        scanf("%s", students[i]);
    }
    for(int i=0; i<5; i++) printf("%s\n", students[i]);
    return 0;
}
```

-

22. Program to append data to a file.

```
#include <stdio.h>
int main() {
    FILE *fp = fopen("log.txt", "a");
    fprintf(fp, "New Entry\n");
    fclose(fp);
    return 0;
}
```

23. Explain 5 different file modes. (See Answer #13).

24. Explain fgetc, fputc syntax and example.

- `int fgetc(FILE *stream);` - Reads one char.
- `int fputc(int char, FILE *stream);` - Writes one char.
- (See Answer #14 and #15 for examples).

25. Demonstrate realloc() function for resizing. (See Answer #9).

26. Syntax for fseek, ftell. Explain fseek.

- `int fseek(FILE *stream, long offset, int whence);`
- `long ftell(FILE *stream);`
- **fseek:** Moves the file cursor to a specific location (beginning, current, or end).

27. Explain fgets, fputs, fopen.

- `fopen`: Opens a file. `FILE *fp = fopen("name", "mode");`
- `fgets`: Reads a line. `fgets(buffer, size, fp);`
- `fputs`: Writes a line. `fputs("text", fp);`

28. Expand int array from 3 to 6 using realloc. Show shrink.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *arr = malloc(3 * sizeof(int)); // Size 3
    // Expand
    arr = realloc(arr, 6 * sizeof(int)); // Size 6
    // Shrink
    arr = realloc(arr, 2 * sizeof(int)); // Size 2
    free(arr);
    return 0;
}
```

29. Move file pointer to end and print size using ftell.

```
#include <stdio.h>
int main() {
    FILE *fp = fopen("file.txt", "r");
    fseek(fp, 0, SEEK_END);
    printf("File size: %ld bytes", ftell(fp));
    fclose(fp);
    return 0;
}
```

30. Explain SEEK_SET, SEEK_CUR, SEEK_END.

- `SEEK_SET`: Start of file.
- `SEEK_CUR`: Current cursor position.
- `SEEK_END`: End of file.

Page 2 Solutions

31. Read paragraph using fgets. Write multiple lines using fputs.

```
#include <stdio.h>
int main() {
    char buf[100];
    FILE *fp = fopen("story.txt", "r");
    while(fgets(buf, 100, fp)) printf("%s", buf); // Read
    fclose(fp);

    fp = fopen("out.txt", "w");
    fputs("Line 1\n", fp); // Write
    fputs("Line 2\n", fp);
    fclose(fp);
    return 0;
}
```

32. Find largest element in 2D array.

```
#include <stdio.h>
int main() {
    int arr[2][2] = {{5, 20}, {15, 8}};
    int max = arr[0][0];
    for(int i=0; i<2; i++)
        for(int j=0; j<2; j++)
            if(arr[i][j] > max) max = arr[i][j];
    printf("Largest: %d", max);
    return 0;
}
```

33. Read 10 ints, function printarr() to print.

```
#include <stdio.h>
void printarr(int A[], int len) {
    for(int i=0; i<len; i++) printf("%d ", A[i]);
}
int main() {
    int A[10];
    for(int i=0; i<10; i++) A[i] = i; // Dummy input
    printarr(A, 10);
    return 0;
}
```

34. Sort elements of array in ascending order.

```
#include <stdio.h>
int main() {
    int a[] = {5, 3, 8, 1, 2}, temp;
    for(int i=0; i<5; i++) {
        for(int j=i+1; j<5; j++) {
            if(a[i] > a[j]) { temp=a[i]; a[i]=a[j]; a[j]=temp; }
        }
    }
    for(int i=0; i<5; i++) printf("%d ", a[i]);
    return 0;
}
```

35. String copy without library.

```
#include <stdio.h>
int main() {
    char s1[] = "Hello", s2[10];
    int i = 0;
    while(s1[i] != '\0') { s2[i] = s1[i]; i++; }
    s2[i] = '\0';
    printf("Copied: %s", s2);
    return 0;
}
```

36. Accept 5 employee details and display.

```
#include <stdio.h>
struct Emp { char name[20]; int id; };
int main() {
    struct Emp e[5];
    for(int i=0; i<5; i++) {
        // Just demonstrating for 1 to save space in execution
        scanf("%s %d", e[i].name, &e[i].id);
    }
    // Display loop here...
    return 0;
}
```

37. String comparison without library.

```
#include <stdio.h>
int main() {
    char s1[] = "abc", s2[] = "abc";
    int i = 0, flag = 0;
    while(s1[i]!='\0' || s2[i]!='\0') {
        if(s1[i] != s2[i]) { flag = 1; break; }
        i++;
    }
    if(flag == 0) printf("Equal"); else printf("Not Equal");
    return 0;
}
```

38. Fibonacci first n terms using recursion.

```
#include <stdio.h>
int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
int main() {
    int n = 5;
    for(int i=0; i<n; i++) printf("%d ", fib(i));
    return 0;
}
```

39. List storage classes. Explain one.

- **List:** `auto`, `extern`, `static`, `register`.
- **Static:** Variable retains its value between function calls. Initialized only once.

40. Reverse string without predefined function.

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "Code";
    int len = 4; // Length of "Code"
    for(int i=len-1; i>=0; i--) printf("%c", str[i]);
    return 0;
}
```

41. Sum of diagonal elements of 2D array.

```
#include <stdio.h>
int main() {
    int m[3][3] = {{1,2,3},{4,5,6},{7,8,9}}, sum=0;
    for(int i=0; i<3; i++) sum += m[i][i];
    printf("Sum: %d", sum);
    return 0;
}
```

42. Explain Array of strings concept.

- It is a 2D character array or an array of character pointers.
- Example: `char names[3][10] = {"Tom", "Dick", "Harry"};`

43. 2D Array syntax. Accept and display.

- **Syntax:** `type name[rows][cols];`

Code:

```
int m[2][2];
// Input loop...
// Print loop using printf("%d ", m[i][j]);
```

-

44. Nested struct inside struct.

- **Concept:** A structure member can be another structure.

Example:

```
struct Date { int d, m, y; };
struct Student {
    char name[20];
    struct Date dob; // Nested
};
```

-

45. Differences between struct and union.

- **Struct:** Allocates memory for *all* members. Total size = sum of members.
- **Union:** Allocates memory for the *largest* member only. Shared memory.

46. Union with 4 members (int, float, char array, double).

```
#include <stdio.h>
union Data { int i; float f; char str[20]; double d; };
int main() {
    union Data data;
    data.i = 10;
    printf("Int: %d", data.i); // Only one active at a time
    return 0;
}
```

47. Struct Book (ID, title, author, price).

```
#include <stdio.h>
struct Book { int id; char title[20]; char author[20]; float price; };
int main() {
    struct Book b;
    scanf("%d %s %s %f", &b.id, b.title, b.author, &b.price);
    printf("%d %s %s %.2f", b.id, b.title, b.author, b.price);
    return 0;
}
```

48. Union Student (ID, name, age, percentage).

```
#include <stdio.h>
union Student { int id; char name[20]; int age; float per; };
int main() {
    union Student s;
    s.id = 1; printf("ID: %d\n", s.id); // Usage one by one
    s.per = 90.5; printf("Per: %.2f", s.per);
    return 0;
}
```

49. Define Union. Access members?

- **Def:** A user-defined data type where all members share the same memory location.
- **Access:** Using dot operator `.` (e.g., `u.member`).
- (See #46 for example).

50. N employee details using array of structure.

```
#include <stdio.h>
struct Emp { int id; char name[10]; };
int main() {
    struct Emp e[10];
    int n = 2; // For example
    for(int i=0; i<n; i++) scanf("%d %s", &e[i].id, e[i].name);
    for(int i=0; i<n; i++) printf("%d %s\n", e[i].id, e[i].name);
    return 0;
}
```

51. Dynamic alloc 5 ints, accept, display, free.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *p = (int*)malloc(5 * sizeof(int));
    for(int i=0; i<5; i++) p[i] = i+10; // Dummy input
    for(int i=0; i<5; i++) printf("%d ", p[i]);
    free(p);
    return 0;
}
```

52. Sum of elements in array using pointers.

```
#include <stdio.h>
int main() {
    int arr[] = {1, 2, 3, 4, 5}, sum = 0;
    int *p = arr;
    for(int i=0; i<5; i++) {
        sum += *p;
        p++;
    }
    printf("Sum: %d", sum);
    return 0;
}
```

53. Explain array of pointers.

- An array where each element is a pointer (address).
- Example: `int *ptr[5];` stores 5 integer addresses.

54. Smallest element in 2D array.

```
#include <stdio.h>
int main() {
    int m[2][2] = {{5, 2}, {8, 1}}, min = m[0][0];
    for(int i=0; i<2; i++)
        for(int j=0; j<2; j++)
            if(m[i][j] < min) min = m[i][j];
    printf("Min: %d", min);
    return 0;
}
```

55. Read 10 ints, separate function to print evens.

```
#include <stdio.h>
void printEven(int arr[], int n) {
    for(int i=0; i<n; i++) if(arr[i]%2 == 0) printf("%d ", arr[i]);
}
```

```
int main() {
    int arr[] = {1,2,3,4,5,6,7,8,9,10};
    printEven(arr, 10);
    return 0;
}
```

56. Sort array in descending order.

```
#include <stdio.h>
int main() {
    int arr[] = {1, 5, 3}, temp;
    for(int i=0; i<3; i++)
        for(int j=i+1; j<3; j++)
            if(arr[i] < arr[j]) { temp=arr[i]; arr[i]=arr[j]; arr[j]=temp; }
    // Loop to print arr...
    return 0;
}
```

57. String concatenation without library.

```
#include <stdio.h>
int main() {
    char s1[20] = "Hi ", s2[] = "There";
    int i=0, j=0;
    while(s1[i] != '\0') i++; // Move to end of s1
    while(s2[j] != '\0') { s1[i] = s2[j]; i++; j++; }
    s1[i] = '\0';
    printf("%s", s1);
    return 0;
}
```

58. 5 employees, display if salary > value.

```
#include <stdio.h>
struct Emp { char name[10]; float sal; };
int main() {
    struct Emp e[5] = {{"A", 5000}, {"B", 2000}}; // etc
    float val = 3000;
    for(int i=0; i<5; i++)
        if(e[i].sal > val) printf("%s\n", e[i].name);
    return 0;
}
```

59. nth Fibonacci number using recursion.

```
#include <stdio.h>
int fib(int n) {
   if(n <= 1) return n;
   return fib(n-1) + fib(n-2);
}
int main() {
   printf("5th Fib: %d", fib(5)); // Note: 0-indexed usually
   return 0;
}
```

60. Static storage class note + example.

- Preserves value even after function scope ends.

```
 void count() { static int c=0; c++; printf("%d", c); }
```

-

61. Palindrome string without predefined functions.

```
#include <stdio.h>
#include <string.h>
int main() {
   char s[] = "madam";
   int i=0, j=4, flag=0; // j = length-1
   while(i<j) {
      if(s[i] != s[j]) { flag=1; break; }
      i++; j--;
   }
   if(flag==0) printf("Palindrome");
   return 0;
}
```

62. Sum of non-diagonal elements of 2D array.

```
#include <stdio.h>
int main() {
   int m[3][3] = {{1,1,1},{1,1,1},{1,1,1}}, sum=0;
   for(int i=0; i<3; i++)
      for(int j=0; j<3; j++)
```

```
        if(i != j) sum += m[i][j]; // Non-diagonal logic
    printf("Sum: %d", sum);
    return 0;
}
```

63. Structure containing another structure and access. (See #44). Access: `student.dob.year`.

Page 3 Solutions

64. Compare structure and union memory allocation with example.

- **Struct:** `struct A { int i; char c; };` Size = `sizeof(int) + sizeof(char)` (approx 5-8 bytes with padding).
- **Union:** `union B { int i; char c; };` Size = `sizeof(int)` (4 bytes, largest member).

65. Book details using structure array for 3 books.

```
#include <stdio.h>
struct Book { char title[20]; };
int main() {
    struct Book b[3];
    for(int i=0; i<3; i++) {
        // Scanf logic here...
    }
    // Printf logic here...
    return 0;
}
```

66. Dynamic alloc N integers, sum, free.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n, sum=0;
    scanf("%d", &n);
    int *arr = (int*)malloc(n * sizeof(int));
    for(int i=0; i<n; i++) {
        arr[i] = i; // Input
        sum += arr[i];
    }
    printf("Sum: %d", sum);
    free(arr);
    return 0;
}
```

67. Largest element in integer array using pointers.

```
#include <stdio.h>
int main() {
    int arr[] = {10, 50, 20}, max;
    int *p = arr;
    max = *p;
    for(int i=0; i<3; i++) {
        if(*(p+i) > max) max = *(p+i);
    }
    printf("Max: %d", max);
    return 0;
}
```