भारतीय सूचना प्रौद्योगिकी संस्थान गुवाहाटी

Indian Institute of Information Technology Guwahati

DATA STRUCTURES LAB (CS111)

ASSIGNMENTS–01

1. Realize the following program:

```c
#include <stdio.h>

void f(void); // function declaration

void g(void) { // function declaration and definition
    printf("Line: %2d, Function: %s\n", __LINE__, __func__);
    return;
}

int main() {
    printf("Line: %2d, Function: %s\n", __LINE__, __func__);
    printf("Line: %2d, Function: %s\n", __LINE__, __func__);
    f(); // function call
    printf("Line: %2d, Function: %s\n", __LINE__, __func__);
    int a = 2, b = 3, add(int x, int y);
    printf("Line: %2d, Function: %s\n", __LINE__, __func__);
    g(); // function call
    printf("Line: %2d, Function: %s\n", __LINE__, __func__);
    printf("%d + %d = %d\n", a, b, add(a, b)); // function call
    printf("Line: %2d, Function: %s\n", __LINE__, __func__);
    return; // Will return a garbage value; may cause warning
}

void f() { // function definition
    printf("Line: %2d, Function: %s\n", __LINE__, __func__);
    g(); // function call
    // return; // We may not use it.
}

int add(int a, int b) { // function definition
    printf("Line: %d, Function: %s\n", __LINE__, __func__);
    return a + b;
}
```

2. Realize the following program (line numbers are at the right of each statement as comments for better understanding):

```c
#include <stdio.h>                                                          // 01
                                                                            // 02
int x = 1;                                                                  // 03
                                                                            // 04
void f();                                                                   // 05
                                                                            // 06
int main() {                                                                // 07
    printf("x = %d, &x = %p\n", x, &x);                                     // 08
    //printf("i = %d, &i = %p\n", i, &i); /* error: 'i' undeclared */       // 09
    int i = 2;                                                              // 10
    printf("i = %d, &i = %p\n", i, &i);                                     // 11
    {                                                                       // 12
        int i = 3, j = 4;                                                   // 13
        printf("i = %d, &i = %p\n", i, &i);                                 // 14
        printf("j = %d, &j = %p\n", j, &j);                                 // 15
    }                                                                       // 16
                                                                            // 17
    for (int k = 1; k < 2; k++)                                             // 18
        printf("k = %d, &k = %p\n", k, &k);                                 // 19
                                                                            // 20
    printf("i = %d, &i = %p\n", i, &i);                                     // 21
    //printf("j = %d, &j = %p\n", j, &j); /* error: 'j' undeclared */       // 22
    //printf("y = %d, &y = %p\n", y, &y); /* error: 'y' undeclared */       // 23
                                                                            // 24
    f();                                                                    // 25
                                                                            // 26
    for (int k = 1; k < 3; k++)                                             // 27
        printf("k = %d, &k = %p\n", k, &k);                                 // 28
    //printf("k = %d, &k = %p\n", k, &k); /* error: 'k' undeclared */       // 29
                                                                            // 30
    return 0;                                                               // 31
}                                                                           // 32
                                                                            // 33
int y = 5;                                                                  // 34
                                                                            // 35
void f() {                                                                  // 36
    printf("x = %d, &x = %p\n", x, &x);                                     // 37
    printf("y = %d, &y = %p\n", y, &y);                                     // 38
    //printf("i = %d, &i = %p\n", i, &i); /* error: 'i' undeclared */       // 39
}                                                                           // 40
```

Hints:

i. `i` is declared and defined at line 10. Therefore, at line 9, the lifetime of `i` has not begun. That is why we cannot access `i` at line 9.

ii. The lifetime of `j` starts at line 13 and ends at line 16. That is why we cannot access `j` at line 22.

iii. `y` is declared and defined at line 34, which is after the `main` function. That is why at line 23, we cannot access `y` from the `main` function.

iv. The lifetime of `k` starts at line 18 and ends at line 19. Again, the lifetime of `k` starts at line 27 and ends at line 28. That is why we cannot access `k` at line 29.

v. When `f` is called at line 25, the variable `i` is within its lifetime, but the access (scope) of `i` is restricted to the function `main` only. This is so because `i` is a named memory location / variable / object residing in an activation record of the `main` function inside the stack segment. That is why at line 39, we cannot access `i` from the `f` function.

vi. All other cases (excluding the above) hold the following. The variables that we access are within their lifetime as well as within their scope.

vii. At line 14, there co-exist two objects with the same name `i`: the first one is declared and defined at line 10, the second one is declared and defined at line 13. However, at line 14, we can access only the object declared and defined at line 13 (the second one). It is so because the second one "hides" the first one. This phenomenon is called "variable shadowing". Sometimes, it is also referred to as "name hiding".

3. Realize the following program:

```c
#include <stdio.h>

int main() {
    int a = 4, f(int a);
    f(a);
}

int f(int a) {
    printf("Line: %2d, a = %d, &a = %p\n", __LINE__, a, &a);
    if (a > 0) {
        f(a - 1); // recursive call
    }
    printf("Line: %2d, a = %d, &a = %p\n", __LINE__, a, &a);
    return a;
}
```

4. Realize the following program:

```c
#include <stdio.h>

int x; // can be accessed from outside this file by declaring it using extern
static int y; // access restricted to this file

void f() { // can be accessed from outside this file
    static int count; // accessible only to this function; default value is 0
    count++;
```

```
    printf("%s is called %d time(s).\n", __func__, count);
    return;
}

static void g() { // access restricted to this file
    printf("Inside %s\n", __func__);
}

void main(void) {
    void f();
    printf("x = %d, y = %d\n", x, y); // default value is 0
    f();
    f();
    f();
}
```

5. Realize the following program:

```
#include <stdio.h>
#include <stdlib.h>

int x = 1, y;
static int a = 2, b;

void f(){}

static void g();

int main(int argc, char *argv[]) {
    static int u = 3, v;
    int s = 4, t;
    printf("CODE/TEXT SEGMENT (LOW MEMORY):\n");
    printf("f      = %p\n", f);
    printf("main   = %p\n", main);
    printf("g      = %p\n", g);
    printf("printf = %p (library function)\n", printf);
    printf("\n");
    printf("DATA SEGMENT (INITIALIZED):\n");
    printf("x = %d, &x = %p (external) \n", x, &x);
    printf("a = %d, &a = %p (static) \n", a, &a);
    printf("u = %d, &u = %p (static, local to main)\n", u, &u);
    printf("\n");
    printf("DATA SEGMENT (UNINITIALIZED):\n");
    printf("y = %d, &y = %p (external)\n", y, &y);
    printf("b = %d, &b = %p (static)\n", b, &b);
    printf("v = %d, &v = %p (static, local to main)\n", v, &v);
    printf("\n");
    printf("HEAP:\n");
    printf("address = %p\n", calloc(1, 1));
    printf("\n");
```

```
    printf("STACK SEGMENT (INITIALIZED/UNINITIALIZED):\n");
    printf("s = %d, &s = %p\n", s, &s);
    printf("t = %d, &t = %p\n", t, &t);
    g();
    printf("\n");
    printf(
        "+-------------------------+\n"
        "|          STACK          | (HIGH MEMORY)\n"
        "+-------------------------+\n"
        "|            |            |\n"
        "|            V            |\n"
        "|                         |\n"
        "|                         |\n"
        "|            ^            |\n"
        "|            |            |\n"
        "+-------------------------+\n"
        "|          HEAP           |\n"
        "+-------------------------+\n"
        "| UNINITIALIZED DATA (BSS)|\n"
        "+-------------------------+\n"
        "| INITIALIZED DATA (DATA) |\n"
        "+-------------------------+\n"
        "|    TEXT/CODE SEGMENT    | (LOW MEMORY)\n"
        "+-------------------------+\n\n"
    );
    printf("Block Starting Symbol (BSS) portion contains "
           "statically-allocated variables."
    );
    return 0;
}

static void g(){
    int i = 1;
    printf("i = %d, &i = %p (stack grows)\n", i, &i);
}
```

6. Write a function in C that takes two pointers to integers and swaps the variables.
Solution:

```
#include <stdio.h>

int main() {
    int a = 2, b = 3;
    void swap(int *, int *);
    swap(&a, &b);
    printf("a = %d, b = %d", a, b);
}

void swap(int *pa, int *pb) {
    *pa ^= *pb;
```

```
    *pb ^= *pa;
    *pa ^= *pb;
}
```

7. Write a program in C to create an array of type `int` of size 5. Use the `scanf` function to take user inputs initializing the array. Then, print the elements of the array.
Solution 1:

```
#include <stdio.h>

int main() {
    int array[5];
    for (int i = 0; i < 5; i ++) {
        scanf("%d", array + i);
    }
    for (int i = 0; i < 5; i ++) {
        printf("%d ", *(array + i));
    }
}
```

Solution 2:

```
#include <stdio.h>

int main() {
    int array[5];
    for (int i = 0; i < 5; i ++) {
        scanf("%d", &array[i]);
    }
    for (int i = 0; i < 5; i ++) {
        printf("%d ", array[i]);
    }
}
```

8. Write a program in C to dynamically alllocate an array of type `int`. The size of the array is an user input. Use the `scanf` function to take user inputs initializing the array. Then, print the elements of the array. Reuse the same memory to store an array of `char`. Now, free the allocated memory from the heap.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int size = 0, *array = NULL;
    printf("Enter the size of the array: ");
    scanf("%d", &size);
    array = (int *) calloc(size, sizeof(int));
    //array = (int *) malloc(size * sizeof(int)); // An alternative of previous line
    printf("Enter the elements of the int-array (size is %d): ", size);
```

```
    for (int i = 0; i < size; i ++) {
        scanf("%d", array + i);
    }
    printf("The elements of the int-array: ");
    for (int i = 0; i < size; i ++) {
        printf("%d ", *(array + i));
    }

    char *c_array = (char *) array; // reusing the same memory
    printf(
        "\nEnter the elements of the char-array (size is %d): ",
        ((int) (size * sizeof(int)))
    );
    for (int i = 0; i < size * sizeof(int); i ++) {
        scanf("%c", &c_array[i]);
    }
    printf("The elements of the char-array: ");
    for (int i = 0; i < size * sizeof(int); i ++) {
        printf("%c ", c_array[i]);
    }

    printf("\nThe modified elements of the int-array: ");
    for (int i = 0; i < size; i ++) {
        printf("%d ", array[i]);
    }

    free(array);
    array = NULL; // handling dangling/ wild pointer
    c_array = NULL; // handling dangling/ wild pointer
}
```

9. Realize the following multi-file program:

   The following code is stored in a file named `my_library.h`:

```
#include <stdio.h>

extern int global_1; // declaration; cannot initialize

void print_static_global();
extern void print_global(); // extern does not have any effect

/* If included, the following will cause error */
//void sayHello();

void sayHi(); // This is never defined
```

   The following code is stored in a file named `my_library.c`:

```
/* Use the following to compile:
gcc -c my_library.c
```

```c
*/
#include "my_library.h"

/* Can be accessed from anywhere */
int global_1 = 1; // definition; this is good
int global_2 = 2; // multiple-definition; undefined behavior
int global_3 = 3; //

/* Can be accessed from anywhere */
extern void print_global() { //the keyword extern does not have any effect
    printf("\n*** File: %s, Function: %s, Line: %d ***\n",
        __FILE__, __func__, __LINE__
    );
    printf(
        "global_1 = %d, &global_1 = %p\n",
        global_1, &global_1
    );
    printf(
        "global_2 = %d, &global_2 = %p\n",
        global_2, &global_2
    );
    printf(
        "global_3 = %d, &global_3 = %p\n",
        global_3, &global_3
    );
    printf("\n*** File: %s, Function: %s, Line: %d ***\n",
        __FILE__, __func__, __LINE__
    );
}

/* Access restricted to this file only */
static int static_global_1 = 4;
static int static_global_2 = 5;

/* Can be accessed from anywhere */
void print_static_global() {
    printf("\n*** File: %s, Function: %s, Line: %d ***\n",
        __FILE__, __func__, __LINE__
    );
    printf(
        "static_global_1 = %d, &static_global_1 = %p\n",
        static_global_1, &static_global_1
    );
    printf(
        "static_global_2 = %d, &static_global_2 = %p\n",
        static_global_2, &static_global_2
    );
    printf("\n*** File: %s, Function: %s, Line: %d ***\n",
        __FILE__, __func__, __LINE__
    );
```

```
}

/* Access restricted to this file only */
static void sayHello() {
    printf("Hello\n");
};
```

The following code is stored in a file named `main.c`:

```
/* Use the following to compile and link:
gcc -c main.c my_library.c -o executable.exe
*/

#include "my_library.h"

//int global_1; //Need not to do it; already in my_library.h
int global_2; // multiple-definition; undefined behavior
// no declaration or definition of global_3

static int static_global_1 = 100;

int main() {
    /*First, we look for global variables*/
    printf(
        "global_1 = %d, &global_1 = %p\n",
        global_1, &global_1
    );
    /*The following will cause undefined behavior.
    This is because global_2 is defined twice. */
    printf(
        "global_2 = %d, &global_2 = %p\n",
        global_2, &global_2
    );
    /*If included, the following will cause an error.
    This is because global_3 is neither declared nor
    defined. */
    //printf(
    //    "global_3 = %d, &global_3 = %p\n",
    //    global_3, &global_3
    //);

    print_global();


    /* Now, we look for static*/
    printf(
        "static_global_1 = %d, &static_global_1 = %p\n",
        static_global_1, &static_global_1
    );
    /*If included, the following will cause an error */
    //printf(
```

```
    //    "static_global_2 = %d, &static_global_2 = %p\n",
    //    static_global_2, &static_global_2
    //);

    print_static_global();

    /* If included, this will cause an error as it is undefined. */
    //sayHello();

    /* If included, this will cause an error as it is undefined. */
    //sayHi();

    return 0;
}
```

10. Realize the following program:

```
#include <stdio.h>
typedef struct node_t {
    int data;
    struct node_t *next;
} Node_t, *Node;
void f(Node_t *h) {
    h ? printf("%d -> ", h->data), f(h->next) : printf("NULL");
}
void g(Node_t *h) {
    h ? g(h->next), printf(" <- %d", h->data) : printf("NULL");
}
int main() {
    Node_t n4 = {4, 0}, n3 = {3, &n4}, n2 = {2, &n3}, n1 = {1, &n2};
    Node h = &n1;
    f(h);
    printf("\n");
    g(h);
    return 0;
}
```

11. Define a `union` that contains (i) a `char` variable, (ii) an `int` variable, (iii) a `float` variable, and (iv) a `double` variable. Now, perform the following:

    i. Create an object of the `union`. Print the address of each variable.

    ii. Print the size of the `union` using the `sizeof()` operator.

    iii. Create an object of the `union` and initialize it to zero (use "`= {0}`" during initialization). Assign a value to the `char` variable and print the other member variables.

    iv. Create an object of the `union` and initialize it to zero (use "`= {0}`" during initialization). Assign a value to the `int` variable and print the other member variables.

v. Use a pointer to the created object. Access the elements using the "->" operator.

12. Define a structure `S` that has two member variables: (i) a member of type `int` and (ii) a member of a nested structure, `P`. `P` has two member variables: (i) a member variable of type `char`, and (ii) a member variable of a nested-union `U`. `U` has a member of type `char` and a member of type float. Create an object of this structure. Scan each of these member variables from the keyboard. Print each of these member variables.

13. Realize the following program:

```c
# include <stdio.h>

typedef float (*FloatFunctionFloatFloat) (float, float);

float add(float x, float y) {
    return x + y;
}

float sub(float x, float y) {
    return x - y;
}

float mul(float x, float y) {
    return x * y;
}

float div(float x, float y) {
    return x / y;
}

FloatFunctionFloatFloat inverse(FloatFunctionFloatFloat function) {
    if (function == add) {
        return sub;
    }
    if (function == sub) {
        return add;
    }
    if (function == div) {
        return mul;
    }
    if (function == mul) {
        return div;
    }
    return NULL;
}

int main() {
    float x = 6, y = 4;
    float z = (mul - add + sub)(x, y); // calls div(x, y)
    printf("(mul - add + sub)(%g, %g) = %g\n", x, y, z);
```

```
    printf("(mul - add + sub) = %p, div = %p\n", mul - add + sub, div);

    FloatFunctionFloatFloat f = add;
    float w = inverse(f)(f(x, y), y); // calls sub(add(x, y), y)
    printf("inverse(f)(f(%g, %g), %g) = %g\n", x, y, y, w);

    return 0;
}
```

14. Realize the following program:

```
#include <stdio.h>

#define SIZE 10

int main() {
    //anonymous structure
    struct {int x; int y;} a = {
        .y = 7,
        .x = 6,
    };
    printf("a.x = %d, a.y = %d\n", a.x, a.y);

    // Less popular way of sparse array initialization
    int array[SIZE] = {[5] 7, 19, [3] 17, 18};
    for (int i = 0; i < SIZE; i++) {
        printf("%d ", array[i]);
    }
}
```

15. Realize the following program:

```
# include <stdio.h>

typedef double  (*DoubleFunctionDouble)(double);

double add(double x, double y) {
    return x + y;
}

DoubleFunctionDouble curryAdd(double x) { // currying
    double f(double y) { // nested function; not allowed in C; GCC extension
        return x + y;
    }
    return f;
}

int main(int argc, char *argv[]) {
    printf("%g\n", add(3.2, 5.6));
    printf("%g\n", curryAdd(3.2)(5.6));
```

```
    return 0;
}
```

16. Realize the following program:

```
# include <stdio.h>
# include <stdarg.h>

double add(const char *format, ...) { // function taking variable number of arguments
    double total = 0.0;
    va_list list;
    va_start(list, format);
    for (int i = 0; format[i]!= '\0'; i++) {
        int s = format[i];
        switch (s) {
            case 'c' : // char is promoted to int va_list
            case 'i' : total += va_arg(list, int);
            break;
            case 'f' : // float is promoted to double in va_list
            case 'd' : total += va_arg(list, double);
            break;
            default  : break;
        }
    }
    return total;
}

int main(int argc, char *argv[]) { // Pseudo function overloading in C
    printf("add(\"c\", 'a') = %lg\n", add("c", 'a'));
    printf("add(\"cf\", 'a', 1.0) = %lg\n",  add("cf", 'a', 1.0));
    printf("add(\"cid\", 'a', 1, 2.0) = %lg\n", add("cid", 'a', 1, 2.0));
    return 0;
}
```