भारतीय सूचना प्रौद्योगिकी संस्थान गुवाहाटी

Indian Institute of Information Technology Guwahati

COMPUTER PROGRAMMING LAB (CS110)
ASSIGNMENTS–05

[Note: The `calloc` function dynamically allocates memory from the heap. We will learn later how to make use of it.]

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:

```c
#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);
}
```