Shiv Nadar Institute of Eminence
Mid Term Examination
Monsoon 2024

COURSE CODE: CSD101 **MAX. DURATION: 1.5 hr**
COURSE NAME: Introduction to Computing and Programming
COURSE CREDIT: 4 **MAX. MARKS: 40**
Date: 01-10-2024

Roll No: _____ Name of Student: _____

Department/ School: _____

INSTRUCTIONS: -

1. Do not write anything on the question paper except **name, roll number** and **department/school.**
2. All the sections are compulsory.

SECTION A (Max Marks = 24 Marks)

1. What is identifier. Illustrate two rules of creating an Identifier in C with example. (3 marks)

Solution:

In C programming, an **identifier** is a name given to entities such as variables, functions, arrays, and other user-defined items.

Rules for Creating Identifiers in C

Here are two fundamental rules for creating identifiers in C:

Identifiers must begin with a letter or an underscore (_):

Examples:

- Valid identifiers: myVariable, _count, total1, sum_of_numbers
- Invalid identifiers: 1stValue (starts with a digit), total# (contains a special character)

Identifiers are case-sensitive:

- In C, uppercase and lowercase letters are treated as distinct. This means that Variable, variable, and VARIABLE would be considered three different identifiers.

Examples:

- Valid identifiers: myVar, MyVar, MYVAR
- Invalid identifiers (in context): If you define int myVar; and later try to access it with myvar;, it will lead to an error since myVar and myvar are treated as different identifiers.

2. Evaluate the following expression: (2 marks)

7%7+7/7-7*7>>1

Solution:

Evaluate 7 % 7: 7%7=0

Evaluate 7 / 7: 7/7=1

Evaluate 7 * 7: $7*7=49$
Substituting the results back into the expression:
The expression now looks like this: $0+1-49>>1$
Evaluate the addition and subtraction: $1-49=-48$
Evaluate -48 >> 1: $=-24$.

Answer: -24

3. Write the output of the following code snippet

3.1 (2 marks)

```
int main() {
   int a = 6;
   int b = a++;
   printf("%d %d\n",_________ );
   return 0;
}
```

Explain the way one can fill in the blanks to get the following output for the above code:
6 7

A) a, b
B) b, a
C) a, a
D) b, b

Ans: B -- b, a

3.2 (2 marks)

```
int main() {
    int arr[5] = {15, 21, 25, 10, 8};
    int min = 1000,  i;
    for ( ___________ ) {
    if (arr[i] < min) {
            min = arr[i];}
    }
    printf("%d\n", min);
    return 0;}
```

Illustrate the way one can fill in the blanks to get the following output for the above code:
10
A) i = 1; i < 5; i++
B) i = 0; i < 5; i++
C) i = 0; i < 4; i++
D) None of the above

Solution:

C -- i = 0; i < 4; i++

4. Provide the output of the following program and **explain the reasoning behind** the chosen output.

4.1 (3 marks)

```
int incr (int i)
{
static int count = 0;
count = count + i;
return (count);
}
main ()
{
int i,j;
for (i = 0; i <=4; i++)
     j = incr(i);
}
```

A) 10
B) 4
C) 6
D) 7

Solution: A

After each iteration value of count is updated as:

0
1
3
6
10

4.2 (3 marks)

```
#include <stdio.h>
int foo(int *x,int *y,int *z)
{
*y = *y+1;
*z = *x+*x;
}
int main(void)
{
int a = 3;
int b = 3;
int c = a+b;
foo(&c,&a,&a);
printf("%d",a);
return 0;
}
```

A) 15
B) 12
C) 20
D) 13

Answer: B

Initialization:

```
int a = 3;
```

int b = 3;
int c = a + b;

c becomes 3 + 3 = 6.

Function Call:

foo(&c, &a, &a); passes the addresses of c, a, and a to the function foo.

Within foo:

*y = *y + 1; (where *y is the value of a):

Since a is 3, this operation increments a by 1.

Now, a = 4.

*z = *x + *x; (where *x is the value of c):

Since c is 6, this operation sets *z (which is also a since &a was passed) to double the value of c.

Therefore, *z becomes 6 + 6 = 12.

Thus, a now becomes 12.

Final Output:

The printf statement prints the value of a, which is now 12.

4.3 (2 marks)

```
#include <stdio.h>
int main()
{
int arr[5];
// Assume base address of arr is 2000 and size of integer is 32 bit
printf("%u %u %u", arr, arr + 1, arr + 3);
return 0;
}
```

Solution:

2000 2004 2012

Address Calculations

Base address of arr:

This is given as **2000**. This address is equivalent to arr.

Address of arr + 1:

arr + 1 points to the next integer in the array.

Address calculation: Address of arr+1=2000+4=2004\text{Address of } arr + 1 = 2000 + 4 = 2004Address of arr+1=2000+4=2004

So, arr + 1 is **2004**.

Address of arr + 3:

arr + 3 points to the fourth integer in the array.

Address calculation: Address of arr+3=2000+(3×4)=2000+12=2012\text{Address of } arr + 3 = 2000 + (3 \times 4) = 2000 + 12 = 2012Address of arr+3=2000+(3×4)=2000+12=2012

So, arr + 3 is **2012**.

5. Consider the function (3.5 marks)

```
find (int x, int y)
    {return (( x < y) ? 0 : ( x – y)); }
```

Let **a, b** be two non-negative integers. The call **find (a, find (a, b))** can be used to find which of the following operation. Justify your answer by providing step wise solution.

A) maximum of a, b
B) positive difference of a, b
C) sum of a, b
D) minimum of a, b

Solution: D

Step-by-Step Solution
Let's break down find(a, find(a, b)) step by step.
Evaluate find(a, b) first:
- If a < b, find(a, b) will return 0.
- If a >= b, find(a, b) will return a - b.

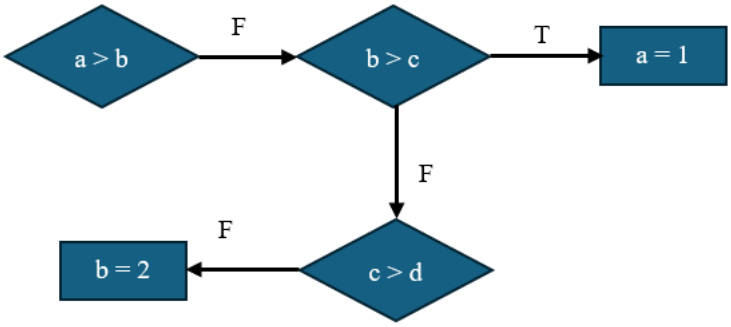
Substitute find(a, b) in the outer call find(a, find(a, b)):
- We now have two cases based on the result of find(a, b).

Case 1: a < b
If a < b, then find(a, b) = 0.
Now we evaluate find(a, 0).
Since a >= 0 (as a is non-negative), find(a, 0) = a - 0 = a.
So, if a < b, the expression find(a, find(a, b)) results in a.
Case 2: a >= b
If a >= b, then find(a, b) = a - b.
Now we evaluate find(a, a - b).
- If a < (a - b), which is not possible (since a - b will always be less than or equal to a when a >= b), this condition will not hold.
- Otherwise, find(a, a - b) will return a - (a - b) = b.

So, if a >= b, the expression find(a, find(a, b)) results in b.

Conclusion
The expression find(a, find(a, b)) essentially returns:
a if a < b
b if a >= b
This operation is equivalent to finding the **minimum** of a and b.
Justification
The function find(a, find(a, b)) is effectively performing the min(a, b) operation.

6. Consider the following flow chart (3.5 marks)



Which of the following correctly implement the above flow chart **(Select two options)**

A)
```
if (a > b)
if (b > c)
a = 1;
else if (c > d)
b = 2;
```

B)
```
if (a <=b)
if (b > c)
a = 1;
else if (c > d)
b = 2;
```

C)
```
if (a > b)
;
else if (b > c)
a = 1;
else if (c < = d)
b = 2;
```

D)
```
if (a > b)
;
else if (b > c)
a = 1;
else if (c > d)
;
else b = 2;
```

Note: Semicolon (;) is used to skip the line and do nothing. Do not select more than two options otherwise negative marks will be awarded.

Solution: C and D

C) if a is greater than b then do nothing. Else if b > c then assign 1 to a else if c <= d then assign 2 to b.
D) if a is greater than b then do nothing. Else if b > c then assign 1 to a else if c > d then do nothing else assign 2 to b.

SECTION B (Max Marks = 16 Marks)

1. Outline the steps involved in the execution of function in C, along with an example? (8 marks)

Or

Function Declaration (Prototype)

- The function is declared with its return type, name, and parameters (if any) at the beginning of the program or in a header file.
- This informs the compiler about the function's existence and its return type.

```
int add(int a, int b); // Declaration
```

Function Definition

- The function's code, also called the function body, is defined. This contains the actual instructions the function will execute when called.
- It specifies the data that the function accepts (parameters), what it returns (return type), and the logic or computations.

```
int add(int a, int b) { // Definition
   int result = a + b;
   return result;
}
```

Function Call

- A function is called (invoked) from another function, typically main() or another user-defined function.
- The control is transferred to the function being called, and the arguments (if any) are passed.

```
int sum = add(3, 5); // Calling the function
```

Example:

```
#include <stdio.h>

// Step 1: Function Declaration (Prototype)
int add(int a, int b);

int main() {
   int num1 = 3, num2 = 5;

   // Step 3: Function Call
   int result = add(num1, num2);

   // Step 7: Continue in the Calling Function
   printf("The sum is: %d\n", result);

   return 0;
}

// Step 2: Function Definition
int add(int a, int b) {
   // Step 5: Execution of Function Body
   int sum = a + b;

   // Step 6: Return Statement
   return sum;
}
```

1. (a) Illustrate the types of parameters or arguments in the function, along with an example. (4 marks)

Solution:
A function argument (or parameter) is a **value passed to a function** when it is called.

The function can use these values to perform its task.

Two types:

Formal Argument: declared in the function definition): Formal parameters behave like local variables inside the function and are **created upon entry** into the function and **destroyed upon exit.**

Actual argument (provided during the function call)

1. Formal Arguments:
Example: int add(**int a, int b**) {
   return a + b;
}
2. Actual Arguments: Arguments are passed to the function when it is called
Example: int result = add(**5, 10**); // 5 and 10 are actual arguments

(b) What are the types of the function call (with arguments) in C, along with an example. (4 marks)

Two ways to call a function:

Call by Value
Call by Reference

arguments can be passed to a function using any of the above way

Call by Value:

A copy of the actual argument is passed to the function.

Modifying the parameter inside the function does not affect the original argument.

Example:

```
void changeValue(int x) {
x = 20;
}
int main() {
int num = 10;
changeValue(num);
printf("%d", num); // Output: 10
        }
```

Call by Reference

A reference (address) to the actual argument is passed to the function.

Modifying the parameter inside the function does affect the original argument.

Example:

```
void changeValue(int *x) {
 *x = 20;
}
int main() {
 int num = 10;
 changeValue(&num);
 printf("%d", num); // Output: 20
}
```

2. How about you help the shopkeepers within the campus with your programming skills. To do this, write a point-of-sale program in C. The program should store the “item-codes” and “item-price” in a shop as a two-dimensional array (named -- "inventory-list") for five different items.

   You can initialize this list as follows,

inv_list[5][2] = {{3,10},{5,30},{9,12},{11,15},{15,80}}

Write a program which takes the shopping list as an input and outputs the amount to be paid by the customer?

(8 marks)

Input:

Enter the no. of bought items: 3

Enter the item id: 3
Enter the item quantity: 2
Enter the item id: 5
Enter the item quantity: 2
Enter the item id: 9
Enter the item quantity: 1

Output:
Total shopping cost is: 92.000

Solution:

```
#include <stdio.h>
int main() {
   int n_itms, item_id, item_quan;
   float inv_list[5][2] = {{3,10},{5,30},{9,12},{11,15},{15,80}};

   float tot_cost = 0;
   printf("Enter the no. of bought items: \n");
   scanf("%d", &n_itms);

   printf("\n\n");
   for (int i = 0; i<n_itms; i++)
   {
      printf("Enter the item id: \n");
      scanf("%d", &item_id);

      printf("Enter the item quantity: \n");
      scanf("%d", &item_quan);

      for (int j = 0; j < 5; j++)
      {
         if(inv_list[j][0] == item_id)
         {
            tot_cost = tot_cost + inv_list[j][1]*item_quan;
         }
      }
   }
   printf("Total shopping cost is: %f", tot_cost);
   return 0;
}
```

Or

2. Write a C program to print the following butterfly pattern. (8 marks)

Solution:

```
#include <stdio.h>

int main() {
   int n;

   printf("Enter the number of rows for the butterfly wings: ");
   scanf("%d", &n);

   // Upper half of the butterfly
   for (int i = 1; i <= n; i++) {
      // Left wing
      for (int j = 1; j <= i; j++) {
         printf("*");
      }
      // Spaces in between
      for (int j = 1; j <= 2 * (n - i); j++) {
         printf(" ");
      }
      // Right wing
      for (int j = 1; j <= i; j++) {
         printf("*");
      }
      printf("\n");
   }

   // Lower half of the butterfly
   for (int i = n; i >= 1; i--) {
      // Left wing
      for (int j = 1; j <= i; j++) {
         printf("*");
      }
      // Spaces in between
      for (int j = 1; j <= 2 * (n - i); j++) {
         printf(" ");
```

```
        }
        // Right wing
        for (int j = 1; j <= i; j++) {
            printf("*");
        }
        printf("\n");
    }

    return 0;
}
```