Tribhuvan University
Faculty of Humanities and Social Science
Office of the Dean
2018

Bachelor of Computer Applications (BCA)
Year / Semester: I / II
Subject: C Programming (CACS151) **FM: 60**
Time: 3 Hrs. **PM: 24**

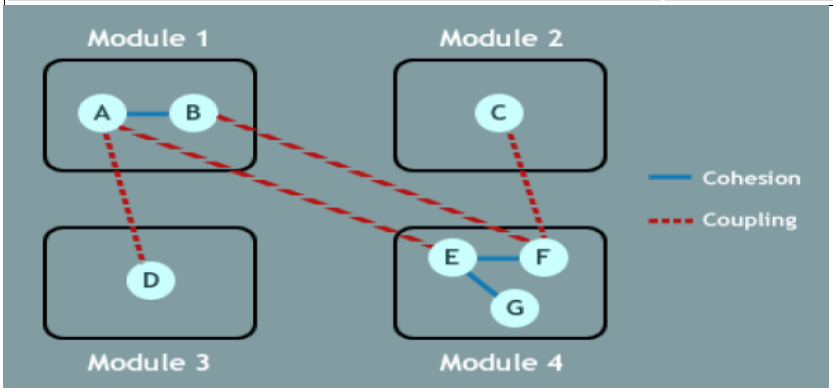
Group B

11. What is software process model? Differentiate between cohesion and coupling in Programming?

A software process model is a systematic approach to developing software that defines the various phases, activities, and tasks that need to be carried out during the software development life cycle. The process model provides a framework for software engineers to follow, ensuring that the software is developed efficiently, on time, and within budget.

There are several different software process models, including the waterfall model, the iterative model, the spiral model, and the agile model. Each model has its own unique characteristics, advantages, and disadvantages, and is suitable for different types of software projects.

Cohesion	Coupling
Cohesion is the indication of the relationship within module	Coupling is the indication of the relationships between modules
Cohesion shows the module's relative functional strength	Coupling shows the relative independence among the modules
Cohesion is a degree (quality) to which a component / module focuses on the single thing	Coupling is a degree to which a component / module is connected to the other modules
While designing we should strive for high cohesion. Ex: cohesive component/module focus on a single task with little interaction with other modules of the system	While designing we should strive for low coupling. Ex: dependency between modules should be less
Cohesion is Intra – Module Concept	Coupling is Inter -Module Concept



12. Define Keywords and Identifiers in C. Explain rules for defining valid identifiers in C. [2+3]

Keywords are reserved words that have a predefined meaning in the language. These words cannot be used as identifiers in a program because they have a special meaning in the language. Examples of keywords in C include if, else, while, for, int, float, char, struct, union, and typedef.

Identifiers are names given to various program elements, such as variables, functions, arrays, and structures. Identifiers are used to identify and refer to these program elements in the code. isdentifiers should be chosen so that they contribute to the readability and documentation of the program.

The rules for defining valid identifiers in C are:

- An identifier must begin with a letter (uppercase or lowercase) or an underscore (_).
- After the first character, an identifier may contain letters, digits, or underscores.
- Identifiers are case-sensitive, meaning x and X are considered different identifiers.
- Identifiers should not be a keyword in the C programming language.
- The length of an identifier should be kept within limits. C does not specify a maximum length for an identifier, but it is good practice to keep the identifier length reasonable.

Here are some examples of valid and invalid identifiers in C:

Valid identifiers:

- count
- _count
- my_variable
- PI
- temp_1

Invalid Identifiers:

- 1count (starts with a digit)
- hello-world (contains a hyphen)
- float (a reserved keyword)
- my long identifier (contains spaces)
- x*y (contains an invalid character)

13. List the operators used in C on the basis of utility. Explain the concept of bitwise operator [2+3]

Operators in C can be classified into several categories based on their usage:

- Arithmetic operators: Used for performing arithmetic operations such as addition, subtraction, multiplication, division, and modulus.
- Assignment operators: Used for assigning a value to a variable, such as the = operator.
- Comparison operators: Used for comparing two values and returning a Boolean value of true or false, such as the ==, !=, <, >, <=, and >= operators.
- Logical operators: Used for performing logical operations such as AND, OR, and NOT, such as the &&, ||, and ! operators.

- Bitwise operators: Used for performing bitwise operations on the binary representations of values, such as the &, |, ^, ~, <<, and >> operators.
- Conditional Operator: Used for making decisions based on a condition. The conditional operator in C is ?: (Ternary Operator).
- sizeof Operator: Used to determine the size of a data type or variable. The sizeof operator in C is used as sizeof().

Bitwise Operators

In arithmetic-logic unit (which is within the CPU), mathematical operations like: addition, subtraction, multiplication and division are done in bit-level.

- To perform bit-level operations in C programming, bitwise operators are used.
- Bitwise operators are efficient while doing low level programming or developing embedded systems.

Operator	Meaning of Operator
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise complement
<<	Shift left
>>	Shift right

Bitwise AND:
The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Bitwise OR:
The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1.
In C Programming, bitwise OR operator is denoted by |.

Bitwise exclusive OR (XOR):
The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite.
It is denoted by ^.

Bitwise complement operator
Bitwise compliment operator is an unary operator (works on only one operand).
It changes 1 to 0 and 0 to 1. It is denoted by ~.

Right Shift Operator:
Right shift operator shifts all bits towards right by certain number of specified bits.
The bit positions that have been vacated by the right shift operator are filled with 0.
It is denoted by >>.

Left Shift Operator:
Left shift operator shifts all bits towards left by a certain number of specified bits.
The bit positions that have been vacated by the left shift operator are filled with 0.
The symbol of the left shift operator is <<.

14. Differentiate between while loop and do while loop. Write a C program to find input number is prime or composite. [2+3]

In C programming, both while loop and do-while loop are used for executing a block of code repeatedly until a certain condition is met. However, there are some differences between the two:

Condition checking: In a while loop, the condition is checked at the beginning of each iteration. If the condition is false, the loop is never executed. On the other hand, in a do-while loop, the condition is checked at the end of each iteration. This means that the loop is executed at least once, even if the condition is false.

Loop execution: In a while loop, the loop is executed only if the condition is true. On the other hand, in a do-while loop, the loop is executed at least once, even if the condition is false.

Here's an example to illustrate the difference:

```
int x = 100;

// While loop
while (x > 100) {
    // This block will never be executed because x is initially 100
    x--;
}

// Do-while loop
do {
    // This block will be executed once, even though x is initially
100
    x--;
} while (x > 100);
```

```
//C program to find input number is prime or composite
#include <stdio.h>
int main() {
    int n, i, flag = 0;

    // Get user input
    printf("Enter a positive integer: ");
    scanf("%d", &n);

    // Check if the number is prime or composite
    for (i = 2; i <= n/2; ++i) {
        if (n % i == 0) {
            flag = 1;
            break;
        }
    }

    if (n == 1) {
        printf("1 is neither prime nor composite.");
    }
    else {
        if (flag == 0)
```

```
            printf("%d is a prime number.", n);
        else
            printf("%d is a composite number.", n);
    }

    return 0;
}
```

15. What is DMA? Write a program to find the largest and smallest number in a list of N number using DMA. [1+4]

DMA stands for "Dynamic Memory Allocation". It is a feature in C programming language that allows programs to allocate memory at runtime, rather than at compile time. In Array we have to declare the size before compiling the program. However, using DMA we can allocate memory dynamically at runtime. We can even change the size of allocated memory.

```
// C program to find the largest and smallest number in a list of N number using DMA

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

int main() {
    int n, i, *arr, max, min;

    // Get the number of elements in the list
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    // Allocate memory for the list
    arr = (int *) malloc(n * sizeof(int));
    // Get the elements of the list from the user
    printf("Enter %d integers:\n", n);
    for (i = 0; i < n; ++i) {
        scanf("%d", arr + i);
    }
    // Find the maximum and minimum elements in the list
    max = *arr;
    min = *arr;
    for (i = 1; i < n; ++i) {
        if (*(arr + i) > max) {
            max = *(arr + i);
        }
        if (*(arr + i) < min) {
            min = *(arr + i);
        }
    }
    printf("Maximum element = %d\n", max);
    printf("Minimum element = %d\n", min);

    // Free the memory allocated for the list
    free(arr);
    return 0;
}
```

Alternate Solution to find the largest number and smallest number in a list of N numbers using DMA.

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

int main() {
    int n, i, *arr, max, min;

    // Get the number of elements in the list
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Allocate memory for the list
    arr = (int *) malloc(n * sizeof(int));

    // Get the elements of the list from the user
    printf("Enter %d integers:\n", n);
    for (i = 0; i < n; ++i) {
        scanf("%d", &arr[i]);
    }

    // Find the maximum and minimum elements in the list
    max = arr[0];
    min = arr[0];
    for (i = 1; i < n; ++i) {
        if (arr[i] > max) {
            max = arr[i];
        }
        if (arr[i] < min) {
            min = arr[i];
        }
    }
    printf("Maximum element = %d\n", max);
    printf("Minimum element = %d\n", min);

    // Free the memory allocated for the list
    free(arr);

    return 0;
}
```

16. What is the difference between binary file and text file? Write a C program to write some text "Welcome to BCA program" in a file test.txt [2+3]

A text file is a file that contains human-readable characters, such as letters, digits, and symbols. The characters in a text file are typically encoded using ASCII or Unicode character sets. In C, text files are usually read and written using standard input/output functions such as fscanf() and fprintf().

A binary file, on the other hand, is a file that contains non-textual data, such as images, audio, video, or executable code. Binary files are usually read and written using low-level input/output functions

such as fread() and fwrite(). Binary files are not human-readable and may contain data that does not represent a character or a string.

In addition, text files and binary files have different file formats. Text files are usually stored in a plain text format, with each character represented by a unique code. Binary files are stored in a binary format, where each byte represents a binary value that can be interpreted as data.

```
// C program to write some text "Welcome to BCA program" in a file test.txt
#include <stdio.h>
int main() {
    FILE *fp;
    char text[] = "Welcome to BCA program";
    // Open the file for writing
    fp = fopen("test.txt", "w");
    if(fp==NULL) {
     printf("File not found”);
     exit();
    }
    // Write the text to the file
    fprintf(fp, "%s", text);
    // Close the file
    fclose(fp);
    return 0;
}
```

17. Explain any four graphics functions in C. Write a program to draw two concentric circles with center (50, 50) and radii 75 and 125. [2+3]

Initgraph()

For writing any graphics program in C we have to call the initgraph function that will initialize the graphics mode on the computer. Initgraph initializes the graphics system by loading the graphics driver from disk then putting the system into graphics mode. Initgraph also resets all graphics settings (color, palette, current position, viewport, etc.) to their defaults.

void initgraph(int *graphdriver, int *graphmode, char *pathtodriver);

closegraph()

closegraph function closes the graphics mode, deallocates all memory allocated by graphics system and restores the screen to the mode it was in before you called initgraph.

void closegraph();

circle():

circle() function is used to draw a circle with a specified center and radius.

e.g. circle(100, 100, 50);

The first two arguments specifies the center (x,y) of the circle and the third argument specifies the radius of the circle.

line():

line function is used to draw a line from a point(x1,y1) to point(x2,y2)

e.g. line(100,100,200,200);

```
// program to draw two concentric circles with center (50, 50) and radii 75 and 125.
#include<graphics.h>
int main()
{
 int gd = DETECT, gm;
 initgraph(&gd, &gm, "C:\\TC\\BGI");
 circle(50, 50, 75);
 circle(50, 50, 125);
closegraph();
return 0;
}
```


Group C

18. What is one dimensional array? How it is initialized? Write a C program to find the sum of two matrix of order m×n. [1+1+8]

A one-dimensional array is a collection of variables of the same data type, accessed by a single index or subscript. In other words, a one-dimensional array is a list of variables of the same data type, where each variable can be accessed using its index.

In C, a one-dimensional array can be initialized during its declaration or later using a loop. Here's an example of initializing a one-dimensional array during its declaration:

int arr[5] = {1, 2, 3, 4, 5};

// **C program to find the sum of two matrix of order m×n**

```
#include <stdio.h>
int main() {
   int m, n, i, j, mat1[m][n], mat2[m][n], sum[m][n];
   printf("Enter the order of the matrices: ");
   scanf("%d %d", &m, &n);

   printf("Enter the elements of the first matrix: ");
   for(i = 0; i < m; i++) {
      for(j = 0; j < n; j++) {
         scanf("%d", &mat1[i][j]);
      }
   }

   printf("Enter the elements of the second matrix: ");
   for(i = 0; i < m; i++) {
      for(j = 0; j < n; j++) {
         scanf("%d", &mat2[i][j]);
      }
   }

   for(i = 0; i < m; i++) {
      for(j = 0; j < n; j++) {
         sum[i][j] = mat1[i][j] + mat2[i][j];
      }
   }

   printf("The sum of the two matrices is:\n");
   for(i = 0; i < m; i++) {
      for(j = 0; j < n; j++) {
         printf("%d ", sum[i][j]);
      }
      printf("\n");
   }
   return 0;
}
```

19. Define structure and union? Write a C program using structure that reads the records of 35 students with members roll, name, address and makes and display the record of students who have obtained greater than 250 marks. [2+8]

In C, a structure is a collection of variables of different data types grouped together under a single name. A structure allows us to create a user-defined data type that can store data in a more organized and meaningful way. Here's an example of defining a structure in C:

```
struct student {
   int roll;
   char name[20];
   char address[50];
```

```
   float marks;
};
```

A union is a special data type in C that allows storing different data types in the same memory location. This means that a union variable can store only one value at a time, but that value can be of different types. Here's an example of defining a union in C:

```
union myUnion {
   int x;
   char y;
   float z;
};
```

```
#include <stdio.h>

struct student {
   int roll;
   char name[20];
   char address[50];
   float marks;
};
```

// **C program using structure that reads the records of 35 students with members roll, name, address and makes and display the record of students who have obtained greater than 250 marks.**

```
int main() {
   struct student s[35];
   int i;
   printf("Enter the records of 35 students:\n");
   for(i = 0; i < 35; i++) {
      printf("Student %d:\n", i + 1);
      printf("Roll: ");
      scanf("%d", &s[i].roll);
      printf("Name: ");
      scanf("%s", &s[i].name);
      printf("Address: ");
      scanf("%s", &s[i].address);
      printf("Marks: ");
      scanf("%f", &s[i].marks);
   }

   printf("Students who obtained greater than 250 marks:\n");
   for(i = 0; i < 35; i++) {
      if(s[i].marks > 250) {
         printf("Roll: %d\n", s[i].roll);
         printf("Name: %s\n", s[i].name);
         printf("Address: %s\n", s[i].address);
         printf("Marks: %.2f\n", s[i].marks);
      }
   }
   return 0;
}
```

20. What is function? List its advantages. Explain the concept of function call by value and function call by reference with example. [1+2+7]

In C, a function is a group of related statements that perform a specific task. It allows us to break a large program into smaller and more manageable modules, making the code easier to understand, test, and maintain. Functions also promote code reuse, as the same function can be called multiple times from different parts of the program.

Advantages of using functions in C:

- Code reuse: Functions can be reused in multiple parts of the program, reducing code redundancy and promoting modularity.
- Encapsulation: Functions allow us to encapsulate complex logic and algorithms, making the code more readable and easier to maintain.
- Testing: Functions can be tested independently, making it easier to detect and fix bugs.
- Efficiency: Functions allow us to write optimized and efficient code by breaking down complex tasks into smaller and simpler ones.

Function call by value is a mechanism in which a copy of the arguments is passed to the function, and any changes made to the arguments inside the function are not reflected outside the function. Here's an example:

```
#include <stdio.h>

void swap(int x, int y) {
   int temp;
   temp = x;
   x = y;
   y = temp;
   printf("Inside swap function: x = %d, y = %d\n", x, y);
}

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

Output:
Before calling swap function: a = 10, b = 20
Inside swap function: x = 20, y = 10
After calling swap function: a = 10, b = 20

Function call by reference is a mechanism in which the memory address of the arguments is passed to the function, allowing any changes made to the arguments inside the function to be reflected outside the function. Here's an example:

```c
#include <stdio.h>

void swap(int *x, int *y) {
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
   printf("Inside swap function: x = %d, y = %d\n", *x, *y);
}

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

Output:

Before calling swap function: a = 10, b = 20

Inside swap function: x = 20, y = 10

After calling swap function: a = 20, b = 10