Assignment

Introduction to 'C' Programming

Batch 2023-27

BCA hons. with AI/DS

Submitted by:

Name: Nikhil Khantwal

ID: 23151431

Submitted to:

Mr. Rishi Kumar

Asst. Prof. CSIT, GEU

ASSIGNMENT

Question 1. What are Constants and variables, Types of Constants, Keywords, Rules for identifiers, int , float, char, double, long void.

Answer 1.

Constants and variables -: A constant is an entity that doesn't change ,whereas ,a variable is an entity that may change .

Types of Constants-:

Constants in C can be divided into two major categories-:

(a) Primary Constants- Integer, Real, character

(b) Secondary Constants-Pointer ,Array ,String ,Structure ,Union ,Enum

At this stage we would restrict our discussion to only primary constants , namely ,Integer ,Real and Character constants.

Keywords-: A keyword is a word that carries special meaning .

Rules for identifiers-:

(a) A variable name is any combination of 1 to 31 alphabets,digits or underscores.Some compilers allow variables names whose length could be up to 247 characters.However ,you should not create any long variable names as it adds to your typing effort

(b) The first character in the variable name must be an alphabet or underscore(_).

Rules for Integer Constants-:

(a) An integer constant must contain at least one digit

(b) It must not contain a decimal point

(c) Its value can be zero,positive or negative.If no sign precedes an integer constant,it is assumed to be positive

(d) Commas or blanks are not allowed within an integer constant

(e)The allowable range for integer constant is -2147483648 to +2147483647

Example-: si_int pop_e_89 , avg , basic_salary

Rules for Floating point Constants-:

(a) A real constant must contain at least one digit

(b) It must contain a decimal point

(c) It can be zero,positive or negative.Default sign is positive.

(d)Commas or blanks are not allowed within a real constant

Example-: +325.34,-32.76,426.0

Rules for character-:

(a) A character constant is a single alphabet,digit or special symbol enclosed within single inverted commas.

(b) Both the single inverted commas should point to the left .

Example-: ‘A’ is a valid character constant,whereas ‘A’ is not.

Question 2. Explain with examples Arithmetic Operators, Increment and Decrement Operators, Relational Operators, Logical Operators, Bitwise Operators, Conditional Operators, Type Conversions, and Precedence, and associativity of operators.

Answer 2.

Arithmetic Operators: These operators are used to perform arithmetic operations like addition, subtraction, multiplication, and division.

Increment and Decrement Operators: These operators are used to increment or decrement the value of a variable by 1.

Relational Operators: These operators are used to compare two values and return a boolean value (1 for true and 0 for false). They are ==, !=, >, <, >=, and <=.

Logical Operators: These operators are used to perform logical operations like AND, OR, and NOT. They are &&, ||, and !.

Bitwise Operators: These operators are used to perform bitwise operations like AND, OR, XOR, NOT, and shift operators. They are &, |, ^, ~, <<, and >>.

Conditional Operators: These operators are used to perform conditional operations based on the result of a logical operation. It is the ternary operator, ? :.

Type Conversions: These operators are used to convert one data type to another.

Precedence: Precedence is the order in which operations are performed when more than one operation needs to be performed on an expression.

Associativity: Associativity determines the order in which operators of the same precedence level are evaluated in an expression.

Expressions: An expression is a sequence of operators and operands that performs some calculation and returns a value.

In C, operators have precedence and associativity, which affect how the expression is evaluated. The associativity of an operator is left-to-right, right-to-left, or not associative, depending on the operator.

The arithmetic operators are addition (+), subtraction (-), multiplication (*), and division (/), and the increment and decrement operators are ++ and --. The relational operators are ==, !=, >, <, >=, and <=. The logical operators are &&, ||, and !. The bitwise operators are &, |, ^, ~, <<, and >>. The conditional operator is the ternary operator, ? :. The type conversion operators are used to convert one data type to another, like (int), (float), (char), etc.

Q3.Explain with example conditional statements if,if-else,elseif,nested if else.

The if statement in C is used to conditionally execute a block of code if a specified condition is true. It consists of the keyword "if", followed by a condition enclosed in parentheses, and a block of code.

Example:

```
int x = 10;

if (x > 5) {

   printf("x is greater than 5\n");

}
```

The if-else statement is used to conditionally execute a block of code if a specified condition is true, and another block of code if the condition is false. It consists of the keyword "if", followed by a condition enclosed in parentheses, and a block of code. The else keyword is then used to specify an alternative block of code to be executed.

Example:

```
int x = 10;

if (x > 5) {

   printf("x is greater than 5\n");

} else {

   printf("x is not greater than 5\n");

}
```

The elseif statement (also known as elif or elsif) is used to provide multiple conditions in an if-else statement. It consists of the keyword "else", followed by the keyword "if", and a condition enclosed in parentheses.

```
int x = 10;

if (x > 15) {

   printf("x is greater than 15\n");

} else if (x > 5) {

   printf("x is greater than 5\n");

} else {

   printf("x is not greater than 5\n");

}
```

Nested if-else statements are used when the if-else statements are placed within another if-else statement.

Example:

```
int x = 10;

int y = 20;

if (x > 5) {

   printf("x is greater than 5\n");

   if (y > 25) {

      printf("y is greater than 25\n");

   } else {

      printf("y is not greater than 25\n");

   }

} else {

   printf("x is not greater than 5\n");

}
```

In the examples above, the condition is a simple comparison operation, but it can be any expression that returns a boolean value. The blocks of code inside the if, else if, and else statements can be a single statement or multiple statements enclosed in curly braces.

Q4.Explain Switch Case statement with example.

The switch-case statement is a selection control mechanism used in C programming to allow a program to select from several alternatives, each of which has a value. It works by sequentially examining each case until it finds one that matches the value of the expression, then executing the corresponding code.

Syntax:

```
switch (expression) {
  case value1:
    // code to be executed if the expression equals value1;
    break;
  case value2:
    // code to be executed if the expression equals value2;
    break;
  ...
  default:
    // code to be executed if the expression does not match any of the cases;
}
```

Example:

```
int x = 10;
char grade;

switch (x) {
  case 90:
  case 100:
    grade = 'A';
    break;
  case 80:
    grade = 'B';
    break;
  case 70:
    grade = 'C';
    break;
```

```
    case 60:
        grade = 'D';
        break;
    default:
        grade = 'F';
}
```

```
printf("The grade for x is %c\n", grade);
```

In this example, the expression is the integer variable x, and the switch-case statement compares this variable with each case value (90, 100, 80, 70, 60) in order. If a match is found, the corresponding code block is executed. In this case, the grade variable is assigned the corresponding letter grade for the score of x. If no match is found, the code block following the default keyword is executed.

The break statement is used to terminate the switch-case statement and transfer the control to the next line following the switch-case statement. Without a break statement, the program will continue executing the next case block, even if the condition is not met.

Q5.Explain Loops,for loop ,while loop,do while loop with examples.

Loops are control flow statements that allow code to be executed repeatedly based on a given Boolean condition.

For loop:

The for loop is a control flow statement that allows a developer to efficiently write a loop that needs to execute a specific number of times.

Syntax:

```
for (initialization; condition; increment/decrement) {
    // code to be executed
}
```

Example:

```
int i;
for (i = 0; i < 5; i++) {
    printf("Hello, world! %d\n", i);
}
```

In this example, the loop begins by initializing the variable i to 0. Then, before each iteration of the loop, the value of i is compared with 5. If i is less than 5, the loop continues and the code within the loop is executed. Finally, after each iteration of the loop, the value of i is incremented by 1.

While loop:

The while loop is a control flow statement that allows a developer to repeatedly execute a block of code as long as a given condition is true.

Syntax:

```
while (condition) {
   // code to be executed
}
```

Example:

```
int i = 0;
while (i < 5) {
   printf("Hello, world! %d\n", i);
   i++;
}
```

In this example, the loop begins by initializing the variable i to 0. Then, before each iteration of the loop, the value of i is compared with 5. If i is less than 5, the loop continues and the code within the loop is executed. Finally, after each iteration of the loop, the value of i is incremented by 1.

Do while loop:

The do while loop is a control flow statement that allows a developer to repeatedly execute a block of code at least once, before checking if the given condition is true.

Syntax:

```
do {
   // code to be executed
} while (condition);
```

Example:

```
int i = 0;
```

```
do {
   printf("Hello, world! %d\n", i);
   i++;
} while (i < 5);
```

In this example, the code within the loop is executed first, before the condition is checked. Then, as long as the condition is true, the loop continues to execute. Finally, after each iteration of the loop, the value of i is incremented by 1.

In all types of loops, if the Boolean condition is not met, the program control will exit the loop and continue with the next statement after the loop.

Q6. Explain with examples debugging importance,tools common errors : syntax,logic,and runtime errors,debugging , and Testing C Programs.

Debugging is the process of correcting bugs and defects within the program to ensure its proper functioning. Debugging can be done using a debugger, which is a software tool that helps in understanding and resolving bugs.

The importance of debugging cannot be overstated. A single bug can cause significant issues and problems within a program, including incorrect output, system crashes, and even data loss.

Common debugging tools include debuggers such as GDB for C programs, Visual Studio Debugger for C++, and breakpoints, which are markers used in debugging to pause the execution of the program at a particular line or statement.

There are several types of errors that can occur in a program:

Syntax Errors: These are errors caused by incorrect usage of the programming language's syntax. They are often caught by the compiler.

Logic Errors: These are errors that occur when the program's logic is incorrect, such as when a program calculates the wrong value. They are typically not caught by the compiler and require manual debugging and testing.

Runtime Errors: These are errors that occur when the program is running and are often due to external factors such as memory constraints, system configuration issues, or user input errors. They are not caught by the compiler and can only be detected and resolved through debugging.

Debugging, along with Testing, plays a crucial role in the software development lifecycle. A program should be thoroughly tested before being deployed to ensure its stability and reliability. This includes testing for correctness, functionality, and performance.

The C Programming Language, being a low-level language, is known for its steep learning curve. Mastering it requires continuous practice and attention to detail. To minimize the occurrence of bugs and ensure high-quality software, developers should use effective debugging and testing strategies.

Example of Debugging in C:

```
#include <stdio.h>

int add(int a, int b) {
   return a + b;
}


int main() {
   int x = 5;
   int y = 10;
   int result = add(x, y);
   printf("The result of adding %d and %d is %d\n", x, y, result);
   return 0;
}
```

In this example, if the add function does not produce the correct result, it could indicate a bug in the code. A developer would then use a debugger to step through the code and identify the problem.

Q7. What is the user defined and pre-defined functions.Explain with example call by value and call by reference.

User-defined functions and pre-defined functions are different types of functions in C programming.

User-defined functions: These are functions created by the programmer for specific tasks or calculations. They are usually called by their names, followed by parentheses enclosing the required arguments. User-defined functions can be divided into two categories: library functions and user-defined functions.

Example of user-defined function:

```
#include <stdio.h>

// Function declaration
int add(int a, int b);

int main() {
   int x = 5;
   int y = 10;
   int result = add(x, y); // Function call
   printf("The result of adding %d and %d is %d\n", x, y, result);
```

```
    return 0;
}

// Function definition
int add(int a, int b) {
    return a + b;
}
```

Pre-defined functions: These are functions provided by the C standard library or other libraries. They can be called directly in the code without needing to be defined first. Examples of pre-defined functions include printf, scanf, and getchar.

Explanation of Call by Value and Call by Reference:

Call by Value: In this method, the value of the argument is passed to the function. The function can access the value, but cannot modify it. Any changes made to the argument within the function are not reflected in the original variable.

Example of Call by Value:

```
#include <stdio.h>
void modify(int num) {
    num = 50;
}

int main() {
    int a = 10;
    modify(a);
    printf("Value of a is %d\n", a); // Output: Value of a is 10
    return 0;
}
```

Call by Reference: In this method, the reference (or address) of the argument is passed to the function. The function can access and modify the original variable through this reference.

Example of Call by Reference:

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

```
void modify(int *num) {
   *num = 50;
}

int main() {
   int a = 10;
   modify(&a);
   printf("Value of a is %d\n", a); // Output: Value of a is 50
   return 0;
}
```

In conclusion, understanding and mastering the concepts of debugging, testing, user-defined and pre-defined functions, and call by value and call by reference are crucial for becoming a proficient C programmer

Q8. 1) Explain with Passing and returning arguments to and from Function. 2) Explain Storage classes, automatic, static, register, external. 3) Write a program for two strings S1 and S2. Develop a C Program for the following operations. a) Display a concatenated output of S1 and S2 b) Count the number of characters and empty spaces in S1and S2.

Passing and returning arguments to and from Function

When we pass an argument to a function, the value of the argument is copied into the function's parameter. Therefore, any changes made to the parameter within the function do not affect the original argument.

For example, in the function int add(int a, int b), the values of a and b are passed to the function as parameters.

Returning an argument from a function means passing the result of a function back to the caller. The return statement in a function contains the value to be returned.

For example, in the function int add(int a, int b), the return statement return a + b; returns the sum of a and b to the caller.

Storage classes, automatic, static, register, external:

Automatic: By default, variables declared inside a function are of the automatic storage class. These variables are automatically allocated memory when the function is called and are deallocated when the function is exited.

Static: Variables declared as static inside a function are of the static storage class. These variables are only allocated memory once and retain their values across multiple function calls.

Register: Variables declared as register are of the register storage class. These variables are stored in a CPU register instead of memory. They are accessed and modified faster than regular variables, but they have limited scope.

External: Variables declared as external outside any function are of the external storage class. These variables are declared to the compiler but defined and initialized in another file.

Program for two strings S1 and S2:

```
#include <stdio.h>

#include <string.h>

// Function to display a concatenated output of S1 and S2

void displayConcatenated(char S1[], char S2[]) {

   printf("Concatenated output of S1 and S2: %s%s\n", S1, S2);

}

// Function to count the number of characters and empty spaces in S1 and S2

void countCharacters(char S1[], char S2[]) {

   int count1 = 0, count2 = 0;

   int i = 0;

   while (S1[i] != '\0') {

      count1++;

      i++;

   }

   i = 0;

   while (S2[i] != '\0') {

      count2++;

      i++;

   }

   printf("Number of characters in S1: %d\n", count1);

   printf("Number of characters in S2: %d\n", count2);

}

int main() {

   char S1[] = "Hello";
```

```
    char S2[] = " World";

    displayConcatenated(S1, S2);

    countCharacters(S1, S2);

    return 0;

}
```

This program includes two functions, displayConcatenated and countCharacters, that perform the desired operations on the strings S1 and S2

Q9. Explain with example 1D array and multidimensional array. Consider two matrices of the size m and n. Implement matrix multiplication operation and display results using functions. Write three functions 1) Read matrix elements 2) Matrix Multiplication 3) Print matrix elements.

1D Array:

In a 1D array, the elements are stored in a linear block of memory. For example, if we have an integer array int arr[5];, the elements of the array are stored in a contiguous block of memory.

To initialize and access a 1D array, we can use the following code:

```
#include <stdio.h>

void printArray(int arr[], int size) {

    for (int i = 0; i < size; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

int main() {

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

    int size = sizeof(arr) / sizeof(arr[0]);

    printArray(arr, size);

    return 0;

}
```

2D Array (Matrix):

In a 2D array, also known as a matrix, the elements are stored in a 2D block of memory. For example, if we have an integer matrix int mat[2][3];, the elements of the matrix are stored in a 2D block of memory.

To initialize and access a 2D array, we can use the following code:

```
#include <stdio.h>

void printMatrix(int mat[][3], int m, int n) {
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         printf("%d ", mat[i][j]);
      }
      printf("\n");
   }
}

int main() {
   int mat[][3] = {{1, 2, 3}, {4, 5, 6}};
   int m = sizeof(mat) / sizeof(mat[0]);
   int n = sizeof(mat[0]) / sizeof(mat[0][0]);
   printMatrix(mat, m, n);
   return 0;
}
```

Matrix Multiplication:

```
#include <stdio.h>

void readMatrix(int mat[][3], int m, int n) {
   printf("Enter matrix elements:\n");
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         scanf("%d", &mat[i][j]);
      }
```

```
    }
}
void matrixMultiplication(int mat1[][3], int mat2[][3], int res[][3], int m, int n, int p) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < p; j++) {
            res[i][j] = 0;
            for (int k = 0; k < n; k++) {
                res[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
}

void printMatrix(int mat[][3], int m, int n) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", mat[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int mat1[][3] = {{1, 2, 3}, {4, 5, 6}};
    int mat2[][3] = {{7, 8, 9}, {10, 11, 12}};
    int res[2][3];
```

```
    int m = sizeof(mat1) / sizeof(mat1[0]);

    int n = sizeof(mat1[0]) / sizeof(mat1[0][0]);

    int p = sizeof(mat2[0]) / sizeof(mat2[0][0]);

    matrixMultiplication(mat1, mat2, res, m, n, p);

    printMatrix(res, m, p);

    return 0;

}
```

In this code, the readMatrix function is used to read the elements of the input matrices. The matrixMultiplication function is used to perform the matrix multiplication operation. Finally, the `print

Q10. Explain with example with Structure, Declaration, and Initialization, Structure Variables, Array of Structures, and Use of typedef, Passing Structures to Functions. Define union declaration, and Initialization Passing structures to functions. Explain difference between Structure and Union. Write a program on details of a bank account with the fields account number, account holder's name, and balance. Write a program to read 10 people's details and display the record with the highest bank balance.

Structure: A structure is a user-defined data type in C that can be used to group variables under one name.

Declaration and Initialization:

```
struct bank_account {

    int account_number;

    char account_holder_name[50];

    float balance;

};

struct bank_account account1 = {101, "John Doe", 5000.0};
```

Structure Variables: A structure variable can be declared and initialized in the same way as other variables in C.

Array of Structures:

```
struct bank_account accounts[10];
```

Use of typedef:

```
typedef struct {

    int account_number;

    char account_holder_name[50];
```

```
    float balance;
} BankAccount;
BankAccount account1 = {101, "John Doe", 5000.0};
```

Passing Structures to Functions:

```
void print_bank_account(struct bank_account account) {
    printf("Account Number: %d\n", account.account_number);
    printf("Account Holder: %s\n", account.account_holder_name);
    printf("Balance: %.2f\n", account.balance);
}
print_bank_account(account1);
```

Union: A union is a special data type that allows a variable to store different data types at different times. The size of a union is the size of the largest member.

Difference between Structure and Union:

Structure:

Allows data of different data types to be stored.

Memory allocated depends on the individual members.

Memory not shared among members.

Union:

Allows only one data type to be stored at a time.

Memory allocated for the largest member.

Memory shared among members.

Here's a program that reads the details of 10 people and displays the record with the highest bank balance:

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

struct bank_account {
    int account_number;
```

```
    char account_holder_name[50];
    float balance;
};
void print_bank_account(struct bank_account account) {
    printf("Account Number: %d\n", account.account_number);
    printf("Account Holder: %s\n", account.account_holder_name);
    printf("Balance: %.2f\n", account.balance);
}
float get_max_balance(struct bank_account accounts[], int n) {
    float max_balance = accounts[0].balance;
    int max_index = 0;

    for (int i = 1; i < n; i++) {
        if (accounts[i].balance > max_balance) {
            max_balance = accounts[i].balance;
            max_index = i;
        }
    }
    return max_balance;
}
int main() {
    struct bank_account accounts[10];
    for (int i = 0; i < 10; i++) {
        printf("Enter details for account %d:\n", i + 1);
        printf("Account Number: ");
        scanf("%d", &accounts[i].account_number);
        printf("Account Holder Name: ");
```

```
        scanf("%s", accounts[i].account_holder_name);
        printf("Balance: ");
        scanf("%f", &accounts[i].balance);
    }

    float max_balance = get_max_balance(accounts, 10);
    for (int i = 0; i < 10; i++) {
        if (accounts[i].balance == max_balance) {
            print_bank_account(accounts[i]);
            break;
        }
    }
    return 0;
}
```

This program declares a structure bank_account and initializes it with the account details. The program then uses the get_max_balance function to find the maximum bank balance among the accounts and the print_bank_account function to display the account details with the highest bank balance.