**C LAB ASSIGNMENT-I**

**(BCA 2021-24)**

Submitted to :- Submitted by :-

Mr. Ashish Kumar Nayyar Divyansh Jha

Q1**. Write in brief about various data types available in C. Write their format specifiers, range and memory requirements.**

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

Following are the examples of some very common data types used in C:

* **char**: The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
* **int:** As the name suggests, an int variable is used to store an integer.
* **float**: It is used to store decimal numbers (numbers with floating point value) with single precision.
* **double**: It is used to store decimal numbers (numbers with floating point value) with double precision.

Different data types also have different ranges up to which they can store numbers. These ranges may vary from compiler to compiler. Below is list of ranges along with the memory requirement and format specifiers on 32-bit gcc compiler.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Types | Memory Size | Range | Format  Specifier |
| char | 1 byte | −128 to 127 | %c |
| signed char | 1 byte | −128 to 127 | %c |
| unsigned char | 1 byte | 0 to 255 | %c |
| int | 2 byte | −32,768 to 32,767 | %d |
| signed int | 2 byte | −32,768 to 32,767 | %d |
| unsigned int | 2 bytes | 0 to 65,535 | %u |
| short int | 2 bytes | −32,768 to 32,767 | %hd |
| signed short int | 2 bytes | −32,768 to 32,767 | %hd |
| unsigned short int | 2 bytes | 0 to 65,535 | %Hu |
| long int | 4 bytes | -2,147,483,648 to 2,147,483,647 | %ld |
| signed long int | 4 bytes | -2,147,483,648 to 2,147,483,647 | %ld |
| unsigned long int | 4 bytes | 0 to 4,294,967,295 | %lu |
| float | 4 bytes |  | %f |
| double | 8 bytes |  | %lf |
| long double | 10 bytes |  | %Lf |

**Q2. Describe in brief various operators and their usage in C program. Also write in brief about various types of errors that may occur in a C Program**.

**ANSWER**

i)

An operator is simply a symbol that is used to perform operations.

There are following types of operators to perform different types of operations in C language.

**Arithmetic Operators**

These operators are used to perform arithmetic/mathematical operations on operands. Examples: (+, -, \*, /, %, ++, –)

 Arithmetic operators are of two types

**a) Unary Operators**: Operators that operate or work with a single operand are unary operators. For example: Increment (++) and Decrement (–) Operators

**b) Binary Operators:** Operators that operate or work with two operands are binary operators. For example: Addition (+), Subtraction (-), multiplication (\*), Division (/) operators

**Relational Operators**

These are used for the comparison of the values of two operands

. Examples: (==! =,<,>, <=,>=)

**Logical Operators**

Logical Operators are used to combining two or more conditions/constraints. The result of the operation of a logical operator is a Boolean value either true or false.

. Examples: &&(AND), ||(OR),!(NOT)

**Bitwise Operators**

The Bitwise operators are used to perform bit-level operations on the operands.

EXAMPLE:

* **& (bitwise AND): The** result of AND is 1 only if both bits are 1.
* **| (bitwise OR): The** result of OR is 1 if any of the two bits is 1.
* **^ (bitwise XOR): The** result of XOR is 1 if the two bits are different.
* **<< (left shift):** takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.
* **>> (right shift): takes** two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.
* **~ (bitwise NOT):** takes one number and inverts all bits of it

**Ternary or Conditional Operators**

The conditional operator is of the form Expression1? Expression2: Expression3.

If the condition (Expression1) is True then we will execute and return the result of Expression2 otherwise if the condition (Expression1) is false then we will execute and return the result of Expression3.

**Assignment Operator**

Assignment operators are used to assigning value to a variable.

 The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.

Example: =, +=, -=, \*=, /=

**ii.)**

Errors are the problems or the faults that occur in the program, which makes the behavior of the program abnormal.

There are mainly five types of errors exist in C programming:

**Syntax error**

Syntax errors are also known as the compilation errors. These errors are mainly occurred due to the mistakes while typing or do not follow the syntax of the specified programming language.

**Run-time error**

Sometimes the errors exist during the execution-time even after the successful compilation known as run-time errors. When the program is running, and it is not able to perform the operation is the main cause of the run-time error. The division by zero is the common example of the run-time error.

**Linker error**

Linker errors are mainly generated when the executable file of the program is not created. This can be happened either due to the wrong function prototyping or usage of the wrong header file.

**Logical error**

The logical error is an error that leads to an undesired output. These errors produce the incorrect output, but they are error-free, known as logical errors. These types of mistakes are mainly done by beginners. The occurrence of these errors mainly depends upon the logical thinking of the developer.

**Semantic error**

Semantic errors are the errors that occurred when the statements are not understandable by the compiler.

**C ASSIGNMENT-II**

Submitted to :- Submitted by :-

Mr. Ashish Kumar Nayyar Divyansh Jha

**Q1 What are the advantages and disadvantages of using function? Compare call by value and call by reference technique. Explain with help of an example.**

**ANSWER**

In c, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}.

**Advantage of functions in C**

* By using functions, we can avoid rewriting same logic/code again and again in a program.
* We can call C functions any number of times in a program and from any place in a program.
* We can track a large C program easily when it is divided into multiple functions.
* Reusability is the main achievement of C functions.

**Disadvantage of functions in C**

* Complexity of the program increases.
* execution speed decreases
* It requires a programmer must be expert in programming

|  |  |
| --- | --- |
| **Call by value** | **Call by reference** |
| A copy of the value is passed into the function | An address of value is passed into the function |
| Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters. | Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters. |
| Actual and formal arguments are created at the different memory location | Actual and formal arguments are created at the same memory location |
| 1. #include<stdio.h> 2. **void** change (**int** num) { 3. printf("Before adding value inside function num=%d \n",num); 4. num=num+100; 5. printf("After adding value inside function num=%d \n", num); 6. } 7. **int** main () { 8. **int** x=100; 9. printf("Before function call x=%d \n", x);    change(x);//passing value in function     printf("After function call x=%d \n", x);  **return** 0; | 1. #include<stdio.h> 2. **void** change (**int** \*num) { 3. printf("Before adding value inside function num=%d \n", \*num); 4. (\*num) += 100; 5. printf("After adding value inside function num=%d \n", \*num); 6. } 7. **int** main () { 8. **int** x=100; 9. printf("Before function call x=%d \n", x); 10. change(&x) ;//passing reference in function 11. printf("After function call x=%d \n", x); 12. **return** 0; |

**Q2 What are the advantages and disadvantages of using arrays?**

**Write a program in c that multiplies two 2d arrays.**

**Advantage of C Array**

**1) Code Optimization:** Less code to the access the data.

**2) Ease of traversing**: By using the for loop, we can retrieve the elements of an array easily.

**3) Ease of sorting:** To sort the elements of the array, we need a few lines of code only.

**4) Random Access:** We can access any element randomly using the array.

**Disadvantage of C Array**

1. **Fixed Size**: Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically like LinkedList which we will learn later.

//multiplication of two 2D array

#include<stdio.h>

void main ()

{

    intr1, c1, r2, c2, a [10][10], b [10][10], c [10][10], i, j, k;

    printf("Enter the row and columns of a 1 matrix\n");

    scanf("%d%d", &r1, &c1);

    printf("Enter the row and columns of a 2 matrix\n");

    scanf("%d%d", &r2, &c2);

    if (c1 == r2)

    {

        printf("Enter the elements of a first matrix\n");

        for (i = 0; i<r1; i++)

        {

            for (j = 0; j<c1; j++)

            {

                scanf("%d", &a[i][j]);

            }

        }

        printf("Enter the elements of a second matrix\n");

        for (i = 0; i<r2; i++)

        {

            for (j = 0; j<c2; j++)

            {

                scanf("%d", &b[i][j]);

            }

        }

        printf("\nMultiplication of two matrices\n");

        for (i = 0; i<r1; i++)

        {

            for (j = 0; j<c2; j++)

            {

                c[i][j] = 0;

                for (k = 0; k<c1; k++)

                {

                    c[i][j] = c[i][j] + a[i][k] \* b[k][j];

                }

            }

        }

        for (i = 0; i<r1; i++)

        {

            for (j = 0; j<c2; j++)

            {

                printf("% d", c[i][j]);

            }

            printf("\n");

        }

    }

    else

    {

        printf("\n matrices multiplication is not possible");

    }

}

**C ASSIGNMENT-III**

Submitted to :- Submitted by :-

Mr. Ashish Kumar Nayyar Divyansh Jha

**Q1 What are enumerated datatypes and how are they created in C.**

**ANSWER**

The enum in C is also known as the enumerated type. It is a user-defined data type that consists of integer values, and it provides meaningful names to these values. The use of enum in C makes the program easy to understand and maintain. The enum is defined by using the enum keyword.

The following is the way to define the enum in C:

enumflag{integer\_const1, integer\_const2,..integter\_constN};

**FOR EXAMPLE**:

#include<stdio.h>

 enummonths{jan=1, feb, march, april, may, june, july, august, september, october, november, december};

intmain()

{

// printing the values of months

 for(inti=jan;i<=december;i++)

 {

 printf("%d, ",i);

 }

    return0;

}

**Q2 write short notes on:**

1. **Pointer to a pointer**
2. **Array of pointer.**
3. **Pointer to a function**

1.  a pointer is used to store the address of a variable in C. Pointer reduces the access time of a variable. However, In C, we can also define a pointer to store the address of another pointer. Such pointer is known as a double pointer (pointer to pointer). The first pointer is used to store the address of a variable whereas the second pointer is used to store the address of the first pointer.

SYNTAX:

int \*\*p;

FOR EXAMPLE:

#include<stdio.h>

void main ()

{

    Int a = 10;

    int \*p;

    int \*\*pp;

    p = &a; // pointer p is pointing to the address of a

    pp = &p; // pointer pp is a double pointer pointing to the address of pointer p

    printf("address of a: %x\n",p); // Address of a will be printed

    printf("address of p: %x\n",pp); // Address of p will be printed

    printf("value stored at p: %d\n", \*p); // value stored at the address contained by p i.e. 10

                                                                       will be printed

    printf("value stored at pp: %d\n",\*\*pp); // value stored at the address contained by the

                                                                          pointer stored at pp

}

2

[Array of pointers](https://www.geeksforgeeks.org/pointers-in-c-and-c-set-1-introduction-arithmetic-and-array/)” is an array of the [pointer variables](https://www.geeksforgeeks.org/pointers-c-examples/). It is also known as pointer arrays.  
**Syntax**:

int \*var\_name[array\_size];

Declaration of an array of pointers:

int \*ptr[3];

We can make separate pointer variables which can point to the different values or we can make one integer array of pointers that can point to all the values.[Array of pointers](https://www.geeksforgeeks.org/pointers-in-c-and-c-set-1-introduction-arithmetic-and-array/):

**Example:**

|  |
| --- |
| // C program to demonstrate  // example of array of pointers.  #include<stdio.h>  constintSIZE = 3;  void main ()  {      // creating an array      intarr[] = {1, 2, 3};        // we can make an integer pointer array to      // storing the address of array elements      inti, \*ptr[SIZE];        for (i = 0; i<SIZE; i++) {     // assigning the address of integer.          ptr[i] = &arr[i];      }      // printing values using pointer      for (i = 0; i<SIZE; i++) {            printf("Value of arr[%d] = %d\n", i, \*ptr[i]);      } |

3.  we can create a pointer of any data type such as int, char, float, we can also create a pointer pointing to a function. The code of a function always resides in memory, which means that the function has some address. We can get the address of memory by using the function pointer.

**Syntax of function pointer**

returntype (\*ptr\_name)(type1, type2…);

FOR EXAMPLE

#include<stdio.h>

voidfunc1(void (\*ptr)());

voidfunc2();

intmain()

{

    func1(func2);

     return0;

}

voidfunc1(void (\*ptr)())

{

    printf("Function1 is called");

    (\*ptr)();

}

voidfunc2()

{

    printf("\nFunction2 is called");

}

**C ASSIGNMENT-IV**

Submitted to :- Submitted by :-

Mr. Ashish Kumar Nayyar Divyansh Jha

**Q1 Write short notes on:**

1. **Bitfields**
2. **Unions.**

**i.) Bitfields**

In C, we can specify size (in bits) of structure and union members. The idea is to use memory efficiently when we know that the value of a field or group of fields will never exceed a limit or is within a small range.

SYNTAX:

struct {

   type [member\_name] : width;

};

For example, consider the following declaration of date without the use of bit fields.

#include<stdio.h>

#include<string.h>

struct {

   unsignedintage :3;

} Age;

intmain( ) {

   Age.age = 4;

   printf( "Sizeof( Age ): %d\n", sizeof(Age));

   printf( "Age.age : %d\n", Age.age );

   Age.age = 7;

   printf( "Age.age : %d\n", Age.age );

   Age.age = 8;

   printf( "Age.age : %d\n", Age.age );

   return0;

}

**ii.) Union**

**Union** can be defined as a user-defined data type which is a collection of different variables of different data types in the same memory location. The union can also be defined as many members, but only one member can contain a value at a particular point in time.

Union is a user-defined data type, but unlike structures, they share the same memory location.

FOR EXAMPLE

unionabc

{

   inta;

charb;

} var;

intmain()

{

  var.a = 66;

  printf("\n a = %d", var.a);

  printf("\n b = %d", var.b);

}

**Q2 Compare Union with Structure in C.**

**ANSWER**

**Structure** and **union** both are user-defined data types in the [C](https://www.javatpoint.com/c-programming-language-tutorial)

|  |  |
| --- | --- |
| Struct | Union |
| The struct keyword is used to define a structure. | The union keyword is used to define union. |
| When the variables are declared in a structure, the compiler allocates memory to each variable's member. The size of a structure is equal or greater to the sum of the sizes of each data member. | When the variable is declared in the union, the compiler allocates memory to the largest size variable member. The size of a union is equal to the size of its largest data member size. |
| Each variable member occupied a unique memory space. | Variables members share the memory space of the largest size variable. |
| Changing the value of a member will not affect other variables members. | Changing the value of one member will also affect other variables members. |
| Each variable member will be assessed at a time. | Only one variable member will be assessed at a time. |
| We can initialize multiple variables of a structure at a time. | In union, only the first data member can be initialized. |
| All variable members store some value at any point in the program. | Exactly only one data member stores a value at any particular instance in the program. |
| The structure allows initializing multiple variable members at once. | Union allows initializing only one variable member at once. |
| It is used to store different data type values. | It is used for storing one at a time from different data type values. |
| It allows accessing and retrieving any data member at a time. | It allows accessing and retrieving any one data member at a time. |