**//Find GCD and LCM of two numbers given by the user using recursion**

```
#include<stdio.h>
int GCD(int, int);
int main()
{
   int  num1, num2, result_gcd;
   printf("Enter first number : ");
   scanf("%d", &num1);
   printf("Enter second number : ");
   scanf("%d", &num2);
   result_gcd = GCD(num1, num2);
   printf("GCD of %d and %d is %d\n", num1, num2,
result_gcd);
   return 0;
}

int GCD(int x, int y)
{
   int rem;
   rem = x % y;
   if(rem == 0)
   {
      return y;
   }
   else
   {
      return(GCD(y, rem));
   }
}
```

**Pointer Function :-**

A pointer to a function is a variable that holds the memory address of a function in computer programming. Instead of storing data, it stores the location in memory where a function is stored. This allows for dynamic function invocation and enables the flexibility to choose and execute different functions during runtime. Pointers to functions are commonly used for implementing callbacks, function tables, and achieving polymorphic behavior in programming languages that support them.

**One Dimentional array using pointer :-**

```
#include <stdio.h>
int main() {
   int arr[5] = {10, 20, 30, 40, 50};
   int *ptr = arr;
   printf("Accessing array elements using
pointers:\n");
   for (int i = 0; i < 5; i++) {
      printf("Value at arr[%d] = %d\n", i, *ptr);
      ptr++;
   }
   return 0;
}
```

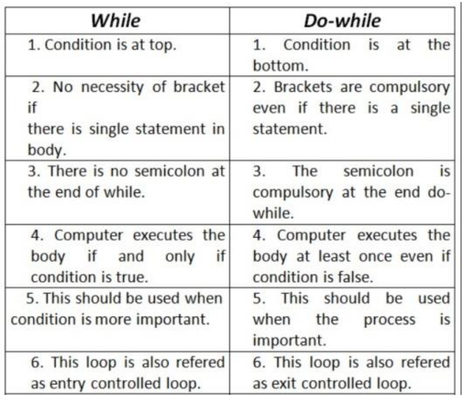
**Write a program in C to check whether a number is divisible by 5 and 11 or not:**

```
#include <stdio.h>
int main() {
   int number;
   printf("Enter a number: ");
   scanf("%d", &number);
   if (number % 5 == 0 && number % 11 == 0) {
      printf("%d is divisible by both 5 and 11.\n",
number);
   } else {
      printf("%d is not divisible by both 5 and 11.\n",
number);
   }
   return 0;
}
```

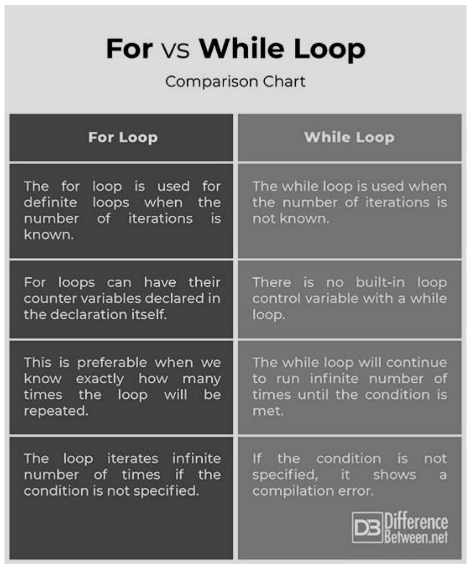
**Differentiate between call by value and call by address.:**

Call by Value:
- Passes actual values.
- Changes don't affect the original.
- Requires additional memory.

Call by Address (Reference):
- Passes memory addresses.
- Changes affect the original.
- No additional memory overhead.



| While | Do-while |
|---|---|
| 1. Condition is at top. | 1. Condition is at the bottom. |
| 2. No necessity of bracket if there is single statement in body. | 2. Brackets are compulsory even if there is a single statement. |
| 3. There is no semicolon at the end of while. | 3. The semicolon is compulsory at the end do-while. |
| 4. Computer executes the body if and only if condition is true. | 4. Computer executes the body at least once even if condition is false. |
| 5. This should be used when condition is more important. | 5. This should be used when the process is important. |
| 6. This loop is also refered as entry controlled loop. | 6. This loop is also refered as exit controlled loop. |



**For** vs **While Loop**
Comparison Chart

| For Loop | While Loop |
|---|---|
| The for loop is used for definite loops when the number of iterations is known. | The while loop is used when the number of iterations is not known. |
| For loops can have their counter variables declared in the declaration itself. | There is no built-in loop control variable with a while loop. |
| This is preferable when we know exactly how many times the loop will be repeated. | The while loop will continue to run infinite number of times until the condition is met. |
| The loop iterates infinite number of times if the condition is not specified. | If the condition is not specified, it shows a compilation error. |

DB Difference Between.net

**//Cheak the year leap year or not**

```
#include<stdio.h>
int main()
{
   int year;
   printf("Enter the year to cheak leap year or not :
");
   scanf("%d", &year);

   if(((year % 4) == 0 && (year % 100) != 0) || ((year
% 400) == 0))
   {
      printf("The year is leap year");
   }
   else
   {
      printf("The year is not leap year");
   }
   return 0;
}
```

**Difference between Compiler and Interpreter:**

### Interpreter:
- *Translation:* Translates code line-by-line during runtime.
- *Execution:* Executes code immediately without creating a separate executable file.
- *Speed:* Generally slower due to interpreting code during runtime.
- *Debugging:* Provides informative error messages during runtime.
- *Portability:* More portable as it doesn't produce machine-specific executables.
- *Edit-Run Cycle:* Shorter since changes can be executed without a separate compilation step.
### Compiler:
- *Translation:* Translates the entire code into machine code or an intermediate form beforehand.
- *Execution:* Produces a separate executable file that can be run independently.
- *Speed:* Faster execution as code is already translated into machine code.
- *Debugging:* May provide less informative error messages compared to interpreters.
- *Portability:* Might need recompilation for different platforms.
- Edit-Run Cycle: Longer as it requires a separate compilation step after code changes.

**Read & Write int value Dinamic Memory Allocation:**

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int n;
   int *arr;
   printf("Enter the number of integers: ");
   scanf("%d", &n);
   arr = (int *)malloc(n * sizeof(int));
   if (arr == NULL) {
      printf("Memory allocation failed. Exiting...\n");
      return 1;
   }
   printf("Enter %d integers:\n", n);
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }
   printf("Entered integers are:\n");
   for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
   free(arr);
   return 0;
}
```

**What is an array?**

An array is a data structure that stores elements of the same type in contiguous memory locations. It has a fixed size, and each element is accessed using an index or subscript. Arrays are commonly used for organizing and managing collections of data in programming.

**Find Reverse of a String :-**

```
#include <stdio.h>
#include <string.h>
int main() {
   char inputString[100];
   printf("Enter a string: ");
   fgets(inputString, sizeof(inputString), stdin);
   int length = strlen(inputString) - 1;
   for (int i = 0; i < length / 2; i++) {
      char temp = inputString[i];
      inputString[i] = inputString[length - 1 - i];
      inputString[length - 1 - i] = temp;
   }
   printf("Reversed string: %s\n", inputString);
   return 0;
}
```

***State advantages of an array***

1. Sequential and Random Access: Arrays allow both sequential and direct access to elements.
2. Memory Efficiency:Fixed-size allocation leads to predictable memory usage and avoids fragmentation.
3. Simplicity: Arrays are easy to declare, use, and understand.
4. Ease of Manipulation: Facilitates various operations like sorting and searching.
5. Compact Code: Enables concise and readable code through loop iterations.
6. Compatibility with Algorithms: Suited for implementing algorithms, especially in numerical processing.
7. Ease of Parameter Passing: Arrays can be efficiently passed as parameters to functions.

**Storage Class :-**

**1. auto** : Default storage class for local variables inside functions. Limited to the block in which it's declared.
**2. register :** Suggests storing variables in CPU registers for faster access. Its usage is a hint to the compiler.
**3. static :** Persists throughout the program execution, retains its value between function calls, and retains storage even after the block ends.
**4. extern** : Declares variables or functions defined in other files, enabling access to them from the current file.

**String Handaling Functions :-**

**1. String Copying Functions:**
- strcpy: Copies one string to another.

**2. String Concatenation Functions:**
- strcat: Concatenates two strings.

**3. String Comparison Functions:**
- strcmp: Compares two strings.
- strncmp: Compares a specified number of characters of two strings.

**4. String Length Functions:**
- strlen: Returns the length of a string.

**5. String Searching Functions:**
- strstr: Finds the first occurrence of a substring in a string.
- strchr: Finds the first occurrence of a character in a string.

**6. Other String Functions:**
- strncpy: Copies a specified number of characters from one string to another.
- strncat: Concatenates a specified number of characters from one string to another.
- strtok: Splits a string into tokens based on specified delimiters.
- strspn: Returns the length of the initial segment of a string consisting entirely of characters from another string.

**Write a C program to convert lowercase string to uppercase and vice versa. Do Not use string.h**

```
#include <stdio.h>
void convertToLowercase(char str[]) {
   int i = 0;
   while (str[i] != '\0') {
      if (str[i] >= 'A' && str[i] <= 'Z') {
         str[i] += 32;  // Convert uppercase to
lowercase
      }
      i++;
   }
}
void convertToUppercase(char str[]) {
   int i = 0;
   while (str[i] != '\0') {
      if (str[i] >= 'a' && str[i] <= 'z') {
         str[i] -= 32;  // Convert lowercase to
uppercase
      }
      i++;
   }
}
int main() {
   char inputString[100];
   printf("Enter a string: ");
   gets(inputString);
   convertToUppercase(inputString);
   printf("Uppercase: %s\n", inputString);
   convertToLowercase(inputString);
   printf("Lowercase: %s\n", inputString);
   return 0;
}
```

**Source vs Object :-**

1. Nature:
   - Source code: Human-readable, written by programmers.
   - Object code: Machine-readable, generated from source code by compilers/assemblers.
2. Format:
   - Source code: Written in a high-level programming language.
   - Object code: Consists of binary/hexadecimal instructions.
3. Readability:
   - Source code: Understandable by humans.
   - Object code: Not directly readable by humans.
4. Usage:
   - Source code: Used for programming and modifications.
   - Object code: An intermediate stage before generating the final executable.
5. Execution:
   - Source code: Needs compilation before execution.
   - Object code: Requires linking or further processing to become an executable.

**Write a program in C to add two numbers using pointers and function.**

```
#include <stdio.h>
int add(int *a, int *b);
int main()
{
   int x, y, sum;
   printf("Enter two numbers: ");
   scanf("%d%d", &x, &y);
   sum = add(&x, &y);
   printf("The sum is %d\n", sum);
   return 0;
}
int add(int *a, int *b)
{
   return *a + *b;
}
```

**Looping:**

In C programming, looping refers to the process of repeatedly executing a block of code as long as a certain condition is true. It allows the program to execute a set of statements multiple times, making the code more efficient and reducing redundancy.

# Key Points about Loops in C :-

-Loops help in executing a block of code repetitively based on a specified condition.

-They aid in controlling the flow of the program and avoiding redundant code.

-It's crucial to ensure that the loop's condition eventually becomes false to prevent infinite loops.

**State advantages of using pointer in C :**

**Efficient Memory Management:**
Pointers enable direct access to memory, optimizing usage.

**Dynamic Memory Allocation:**
Allows dynamic memory allocation for flexibility.

**Passing Addresses as Function Arguments:**
Enables functions to modify caller's variables.

**Efficient Array Manipulation:**
Supports direct array manipulation using pointers.

**Enhanced String Handling:**
Facilitates efficient string manipulation.

**Types of Arrays :-**

**#Based on Dimensions:**

**1. One-dimensional Array**: Linear collection of elements accessed using a single index.

**2. Multi-dimensional Array**: Arrays with multiple dimensions accessed using multiple indices.

**#Based on Declaration:**

**1. Static Array**: Fixed size determined at compile-time.

**2. Dynamic Array:** Size determined at runtime using memory allocation functions.

**#Based on Element Type:**

**1. Homogeneous Array**: Elements of the same data type.

**2. Heterogeneous Array:** Elements of different data types or structures.

**Write a program in C to find ASCII value of a character:**

```
#include <stdio.h>
int main() {
   char character;
   printf("Enter a character: ");
   scanf("%c", &character);
   printf("ASCII value of '%c' is %d\n", character,
character);
   return 0;
}
```

**Sizeof() Operator :-**

- sizeof() returns the size in bytes.
- The result of sizeof() may vary depending on the compiler and system architecture.
- It's particularly useful for dynamic memory allocation and determining the size of data structures

**Example :-** sizeof(Expression);

**Write a program in C to find all prime numbers between given interval using functions:**

```
#include <stdio.h>
#include <stdbool.h>
bool isPrime(int num) {
   if (num <= 1) return false;
   for (int i = 2; i * i <= num; i++)
      if (num % i == 0) return false;
   return true;
}
void displayPrimesInInterval(int start, int end) {
   printf("Prime numbers between %d and %d:\n",
start, end);
   for (int i = start; i <= end; i++)
      if (isPrime(i)) printf("%d\n", i);
}
int main() {
   int start, end;
   printf("Enter start and end of the interval: ");
   scanf("%d %d", &start, &end);
   displayPrimesInInterval(start, end);
   return 0;
}
```

**Types of User-Defined-Function :-**

**1. No Return Value, No Arguments:**
   - Function doesn't return any value and doesn't take any arguments.

**2. Return Value, No Arguments:**
   - Function returns a value but doesn't take any arguments.

**3. No Return Value, With Arguments:**
   - Function doesn't return a value but takes arguments.

**4. Return Value, With Arguments:**
   - Function both returns a value and takes arguments.

**Explain malloc with example:**

malloc() is a C function for dynamic memory allocation, allowing you to request a specific number of bytes in the heap at runtime. It returns a void* pointer, which can be cast to the desired type.

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int dynamicArray = (int)malloc(5 * sizeof(int));
   if (dynamicArray != NULL) {
      free(dynamicArray);    }
   return 0;
}
```

**Explain calloc with example:**

calloc() is a C function for dynamic memory allocation that initializes allocated memory to zero. It takes two arguments: the number of elements to allocate and the size of each element in bytes.

Example:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int dynamicArray = (int)calloc(5, sizeof(int));
   if (dynamicArray != NULL) {
      free(dynamicArray);    }
   return 0;
}
```

**Advantages of Structured Program :-**

1. Clarity and Readability: Clear, understandable code structure.
2. Modularity and Reusability: Breaks problems into manageable modules, promoting code reuse.
3. Ease of Maintenance: Easier to maintain and update.
4. Debugging and Testing: Facilitates systematic debugging and testing.
5. Structured Control Flow: Enhances program control and logic.
6. Reduction of Complexity: Organized code reduces overall complexity.
7. Portability and Maintainability: More portable and easier to maintain.
8. Structured Design: Promotes a systematic approach to problem-solving.

**Header File :-**
A C header file contains declarations of functions, constants, and data types used in multiple program files. It allows sharing these declarations across different parts of the program for code reusability and consistency. They are included using #include and have a .h extension.

**What is C?**
C is a powerful, versatile, and efficient programming language known for its portability, direct hardware access, and wide applicability in system software, embedded systems, and applications due to its rich standard library and structured nature.

**Explain realloc with example:**
realloc() resizes dynamically allocated memory in C. It takes a pointer to the original block and the new size in bytes. Returns a pointer to the resized block. If unsuccessful, returns NULL.
**Example**:
```
 #include <stdio.h>
#include <stdlib.h>
int main() {
   int arr = (int)malloc(3 * sizeof(int));
   arr = (int*)realloc(arr, 5 * sizeof(int));
   free(arr);
   return 0;
}
```

**Macros :-**
Macros in C are preprocessor directives created using #define. They define symbolic constants or code snippets that get replaced throughout the code before compilation. They improve code readability, aid in defining constants, and create reusable code fragments.

**Explain free with example:**
free() is a C function used to deallocate memory allocated by malloc(), calloc(), or realloc(). It takes a pointer to the allocated memory as an argument.
Example:
```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int arr = (int)malloc(5 * sizeof(int));
   free(arr);
   return 0;
}
```

**Assembler :-**
Assembler in C translates assembly code to machine code for the computer's hardware, creating object files for linking into executable programs.

**Typecasting :-**
Typecasting, or type conversion, is the process of converting one data type into another. In C, it can be implicit (automatic) or explicit (done by the programmer using (type)value). It's crucial for ensuring compatibility between different data types but may result in loss of precision or information.

**What is local and global veriable in C :**
**Local Variables:**
Declared within a block (e.g., a function).
Limited scope to the block.
Memory is allocated upon entering the block and released upon exit.
Accessible only within the block.
**Global Variables:**
Declared outside of functions.
Program-wide scope.
Memory is allocated at program start and released at program end.
Accessible from any part of the program.
Local variables are specific to a block, while global variables have a broader scope across the entire program.

**= vs == :-**
= (Assignment Operator) is used to assign a value to a variable, whereas == (Equality Operator) is used to compare whether two values are equal in a condition.

**Token :-**
Tokens in C are the smallest units of code, including keywords, identifiers, constants, operators, special symbols, and comments. They form the fundamental elements parsed by the compiler to understand the program's structure.

**Syntax Error :-**
Syntax errors violate programming language rules, found by the compiler during compilation, stopping code execution. Examples: missing punctuation, incorrect syntax, must fix for successful compilation.

**'\0' Character :-**
The '\0' character marks the end of a string in programming languages like C and C++, signaling where the string ends in memory.

**<> vs " " :-**
1. < > : Used for system header files. Compiler searches in system directories.
2. " " : Used for user-defined or local header files. Compiler searches in the current directory or specified directories.

**Debugging :-**
Debugging is finding and fixing errors in code. It involves identifying issues, locating their source, making corrections, and ensuring the code functions as expected.

**Library Function :-**
Library functions in C are pre-written functions available in libraries like standard ('stdio.h', 'stdlib.h') or third-party. They perform specific tasks, aiding in code reusability and saving time by providing ready-made solutions for common programming tasks.

**Prototype Declaration :-**
Function prototype in C declares the function's signature (return type, name, parameter types) without its body. It enables the compiler to check for errors during compilation and allows calling functions before their actual definitions.

**What is Keywords :-**
Keywords are reserved words in programming languages with predefined roles, controlling program flow, data types, and structures. Examples: if, int, return. They define syntax and are crucial for code functionality.

**Avoid & Disadvantages goto() :-**
Avoid goto in modern programming. Its disadvantages include complex and less readable code, hindered debugging, violation of structured programming principles, and potential unintended side effects. Using structured alternatives enhances code readability, organization, and reduces errors.

**Break vs Continue :-**
1. ***Exit Behavior* :-**
  - break: Terminates the loop entirely.
  - continue: Skips the current iteration, proceeding to the next.
2. ***Loop Control* :-**
  - break: Exits the innermost loop.
  - continue: Skips the current iteration's code.
3. ***Execution Flow* :-**
  - break: Jumps to the code following the loop after termination.
  - continue: Continues with the next iteration of the loop.

**Source File vs Object File vs Binary Executable File :-**
**# Source File :-**
- Contains human-readable code written in a programming language (e.g., C, C++).
- Has file extensions like .c (C), .cpp (C++), or others.
- Contains the program's logic and is editable by programmers.
**#Object File :-**
- Generated by compiling the source code using a compiler.
- Contains machine code and is not directly executable by the CPU.
- Includes translated code from the source file, symbol table information, and references to external functions.
- File extensions might be .o (Unix-based systems) or .obj (Windows).
**#Binary Executable File :-**
- Created by linking one or more object files together, along with necessary libraries.
- Contains machine code directly executable by the CPU.
- The final form of the program that users run.
- File formats differ based on the operating system, such as .exe on Windows, no extension on Unix-based systems, etc.

**Break vs Continue :-**
1. ***Exit Behavior* :-**
  - break: Terminates the loop entirely.
  - continue: Skips the current iteration, proceeding to the next.
2. ***Loop Control* :-**
  - break: Exits the innermost loop.
  - continue: Skips the current iteration's code.
3. ***Execution Flow* :-**
  - break: Jumps to the code following the loop after termination.
  - continue: Continues with the next iteration of the loop.

**String Functions :-**
1. **strlen :-** Calculates the length of a string.
2. **strcmp :-** Compares two strings lexically.
3. **strcat :-** Concatenates (appends) one string to another.
4. **strcpy :-** Copies one string to another.

**Array vs Pointer :-**
**#Arrays:**
Advantages :-
- Indexed access
- Fixed size
- Compile-time error checking
Disadvantages :-
- Fixed size limitation
- Lack of dynamic memory allocation
- Pass-by-value in function calls

**#Pointers:**
Advantages :-
- Dynamic memory allocation
- Flexibility in memory addressing
- Pass-by-reference in function calls
Disadvantages :-
- Manual memory management
- Potential for memory errors
- Indirection complexity

**#Differences :-**
- Arrays store elements of the same type; pointers store memory addresses.
- Arrays have a fixed size; pointers can change dynamically.
- Arrays allow direct element access; pointers need to be dereferenced to access data.