|  |  |
| --- | --- |
| Iteration | Recursion |
| Iterative method having a loop condition. | Recursive method is called with a base case. |
| Iteration uses a repetition structure. | Recursion uses a selection structure. |
| Iteration explicitly uses a repetition structure. | Recursion achieves repetition through repeated method calls. |
| Iteration terminates when the loop-continuation condition fails. | Recursion terminates when a base case is recognized. |
| An infinite loop occurs with iteration if the loop-continuation test never becomes false. | Infinite recursion occurs if the recursion step does not reduce the problem in a manner that converges on the base case. |
| Write an Example | Write an Example |

1. Compare iteration vs recursion

2. Differentiate/Compare Call by value and call by reference

|  |  |
| --- | --- |
| Call by Value | Call by reference |
| Passing the arguments to a function by its value. | Passing the arguments to a function by its address. |
| It may return only one value by a single variable. | It may return many values by direct memory address of the variable. |
| It uses the copy of original value. | It allows function to access the original value of the passed arguments. |
| It is a simple method. | It is more complex method than call-by-value. |
| The access time is more than call-by-reference. | The access time is less that it uses the direct memory address. |
| Give one EXAMPLE Function | Give one EXAMPLE Function |

3. Dynamic memory allocation and related functions.

C language requires the number of elements in an array to be specified at compile time. But we may not be able to do so always. Our initial judgment of size, if it is wrong, may cause failure of the program or wastage of memory space.

C languages permit a programmer to allocate memory at run time. C language has the ability to calculate and assign, during execution, the memory space required by the variables in a program. The process of allocating memory at the run time is known as dynamic memory allocation. There are four library functions known as “memory management functions” that can be used for allocating and freeing memory during program execution. To use these functions header file “stdlib.h” or “alloc.h” to be included in the C program.

|  |  |
| --- | --- |
| Function | Task |
| malloc() | Allocates request size of bytes and returns a pointer to the first byte of the allocated space. |
| calloc() | Allocates space for a number of elements, initialize them to 0 and then returns a pointer to the memory. |
| realloc() | Modifies the size of previously allocated space. |
| free() | Frees previously allocated space. |

malloc()🡪pointer\_variable=(cast\_type\*)malloc(byte\_size);

calloc()🡪 pointer\_variable=(cast\_type\*)calloc(n, element\_size);

realloc()🡪 realloc(pointer\_variable, new\_size);

free()🡪 free(pointer\_variable);

4. Short Note : String functions – strlen, strcpy, strcmp, strncmp, strrev, strcat

strlen() function:

This function counts and return the number of characters in a string.

n=strlen(String);

strcpy() function:

This function works almost like a string assignment operator. It copies one string to another.

strcpy(string2, string1); It copies one string1 to string2.

strcmp() function:

This function compares two strings identified by the arguments and has a value 0 if they are equal. It is case sensitive. So “abc” and “aBc” are not considered equal.

stricmp() function or strcmpi() function

This is a variation of the function strcmp() but it ignores case difference. So “ABC”, “abc”, “aBc” are all considered equal.

strrev() function:

This function reverse a string.

strrev(s1); It returns the reverse of string s1.

strcat() function:

This function concatenates two strings and store in the first string.

strcat(s1,s2); It returns the concatenated string as s1.

5. Compare IF Vs Switch

|  |  |
| --- | --- |
| if | switch |
| It can handle single condition. | It can handle multiple conditions. |
| ‘if’ has an ‘else’ part. | ‘switch’ has multiple ‘case’ part. |
| It is not necessary to give ‘break’ statement in if-else. | It is necessary to give ‘break’ statement in each ‘case’ part. |
| There is no ‘default’ statement. | There must be ‘default’ statement in switch-case. |
| Nesting of if-else is necessary for multiple condition. | It handles multiple conditions by its structure only. |
| If-else can work using any data type. | Switch-case can only use ‘int’ and ‘char’. |

6. Differentiate Array of pointers vs pointer to array

|  |  |
| --- | --- |
| Array of pointer | Pointer to array |
| An array consists of pointers. | An array can be pointed by a single pointer. |
| It can denote the 2-dimentional array. | It can denote the 1-dimentional array. |
| The address of multiple variables and arrays are stored together in array of pointer. | It can store only the address of an array. |
| Size represents the number of rows; the space for columns may be dynamically allotted. | Size represents the number of columns; the space for rows may be dynamically allotted. |

8. Comparing for, while and do-while loop.

For comparing only while and do-while loop OMIT POINT NO **1** & POINT NO **4**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | For | While | Do-while |
| 1 | * + Initialization of loop variables can be done as part of the loop at the beginning. It may also contain condition and usually counter variable increment or decrement. | Initializations of loop variables are to be done outside of loop, before the loop starts. | Initializations of loop variables are to be done outside of loop, before the loop starts. |
| 2 | Loop condition is checked at the beginning and if condition is satisfied loop body is executed.  Hence it is Entry checking loop. | Loop condition is checked at the beginning and if condition is satisfied loop body is executed.  Hence it is Entry checking loop. | Loop body is executed first and then condition is checked at the end of the loop.  Hence it is Exit checking loop. |
| 3 | As condition is checked first, it may not even enter into the loop, if the condition is false. | As condition is checked first, it may not even enter into the loop, if the condition is false. | As condition is checked at the end, at least one iteration takes places, even if the condition is false. |
| 4 | If continue statement is executed within the loop body program control goes at the beginning of the loop without executing further loop body but counter variable increment or decrement mentioned at the starting parenthesis is executed though that is the last statement of the loop. | If continue statement is executed within the loop body program control goes at the beginning of the loop without executing further loop body. | If continue statement is executed within the loop body program control goes at the beginning of the loop without executing further loop body |
| 5 | mention loop Syntax | mention loop Syntax | mention loop Syntax |

**9. Compilers vs. Interpreters**

|  |  |
| --- | --- |
| Interpreters | Compilers |
| An interpreter reads the source code of the program one line at a time, and converts the source code of that line into machine readable format to perform the specific instructions contained in that line. | A compiler reads the entire program and converts it into machine readable format or machine code, which is a translation of the program's source code into a form that the computer can execute directly. |
| Every time execution is done interpreter needs to convert the code | Once the program is compiled, further conversion is not required and so source code is  no longer required in the execution of the program |
| Programme execution is slow | Programme execution is fast |
| Debugging is easy | Debugging is not so easy |
| Example : Interpreter for BASIC Language | Example : Compiler for C language |

10. Pointer arithmetic

If ‘p’ is declared as a pointer variable of any type and it has been initialized properly, then, just like a simple variable , any operation can be performed with ‘\*p’. because \* implies value at address, working with ‘\*p’ means working with the variable whose address is currently held by ‘p’. any expression, whether relational, arithmetic, or logical, can be written, which is valid for a simple value variable. But with only ‘p’, operations are restricted as in each case address arithmetic has to be performed. The valid operations on pointers are:

(a) Assignment of pointers to the same type of pointers

(b) Adding or subtracting a pointer and an integer.

(c) Incrementing and decrementing the pointers (within array limits).

The following arithmetic operators will not works on pointers:

(a) Addition two pointers;

(b) Multiplying a pointer with a number;

(c) Dividing a pointer with a number.

11. In C array is not bounds checking.

In [computer programming](http://en.wikipedia.org/wiki/Computer_programming), bounds checking is any method of detecting whether a [variable](http://en.wikipedia.org/wiki/Variable_(programming)) is within some [bounds](http://en.wikipedia.org/wiki/Upper_and_lower_bounds) before it is used. It is usually used to ensure that a number fits into a given type (range checking), or that a variable being used as an [array](http://en.wikipedia.org/wiki/Array_data_structure) index is within the bounds of the array (index checking). A failed bounds check usually results in the generation of some sort of [exception](http://en.wikipedia.org/wiki/Exception_handling) signal.

Because performing bounds checking during every usage is time-consuming, it is not always done. [Bounds-checking elimination](http://en.wikipedia.org/wiki/Bounds-checking_elimination) is a [compiler optimization](http://en.wikipedia.org/wiki/Compiler_optimization) technique that eliminates unneeded bounds checking. In C language bounds checking for array is not done before it is used.

Example : array is declared as

int ar[5];

but the code is written as

for(i=0;i<10;i++)

scanf(“%d”,&ar[i])  
compiler will not generate any error.

Therefore an error may be occurred, such as, overflow error.

12. GOTO

A goto statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

But use of goto statement is discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

Syntax

The syntax for a goto statement in C is as follows –

goto label;

..

.

label: statement;

Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement. The label can be on a separate line or on the same line as the statement following it.

**WRITE EXAMPLE CODE**

13. WAP to print a square of size 5 using character ‘S’

* 1. S S S S S  
     S S S S S  
     S S S S S  
     S S S S S  
     S S S S S

Solve:

#include<stdio.h>

#include<conio.h>

void main()

{

int i,j;

for(i=0;i<5;i++)

{

for(j=0;j<5;j++)

{

printf(“S”);

}

printf(“\n”);

}

getch();

}

b. S S S S S  
S S  
S S  
S S  
S S S S S

Solve:

#include<stdio.h>

#include<conio.h>

void main()

{

int i,j;

for(i=0;i<5;i++)

{

for(j=0;j<5;j++)

{

if(i==0||j==0||i==4||j==4)

printf(“S”);

}

printf(“\n”);

}

getch();

}

c. S S S S S  
S S S S S  
S S A S S  
S S S S S  
S S S S S

Solve:

#include<stdio.h>

#include<conio.h>

void main()

{

int i,j;

for(i=0;i<5;i++)

{

for(j=0;j<5;j++)

{

if(i==2&&j==2)

printf(“A”);

else

printf(“S”);

}

printf(“\n”);

}

getch();

}