अध्याय 2

ऐरे (Array)

ऐरे,समरूप डेटा तत्वों के परिमित क्रमो का एक संग्रह है जोकि क्रमिक मैमोरी स्थानों में संग्रहित होते है।

यहाँ शब्द

परिमित का अर्थ डेटा रेंज निर्धारित होनी चाहिए।

क्रम का अर्थ डेटा निरंतर मैमोरी स्थानों में संग्रहित किया जाना चाहिए।

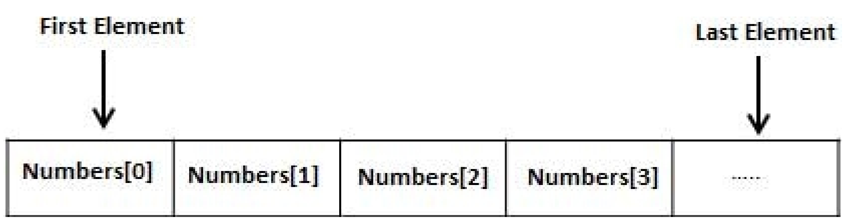
समरूप का अर्थ डेटा एक ही प्रकार का होना चाहिए।

ऐरे दो प्रकार का होता हैं:

1. एकल या एक आयामी ऐरे
2. बहु आयामी ऐरे

एकल या एक आयामी ऐरे:

आइटमों की एक सूची के लिए केवल एक सबस्क्रिप्ट का उपयोग करके एक वैरिएबल नाम दिया जा सकता है और इस तरह के वैरिएबल को एकल सबस्क्रिप्टेड वैरिएबल या एकल आयामी ऐरे कहा जाता है।



एक आयामी ऐरे की घोषणा (डिक्लेरेशन):किसी भी अन्य वैरिएबल की तरह, ऐरे को भी उपयोग से पहले डिक्लेयर किया जाना चाहिए ताकि कम्पाईलर उनके लिए मैमोरी में स्पेस आंवटित कर सके। ऐरे को निम्न प्रकार से डिक्लेयर किया जाता है:

type variable-name[size];

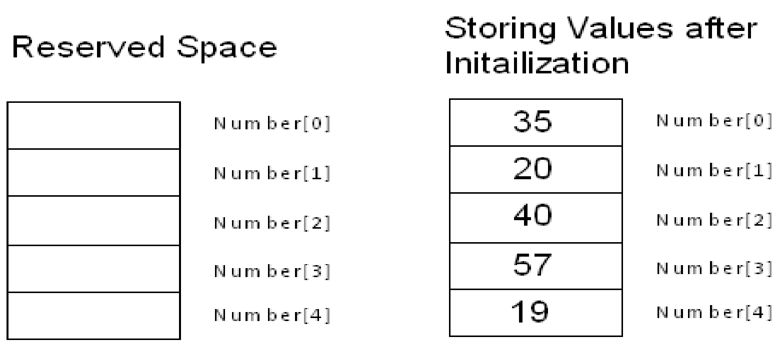
उदाहरण

```
int groupt[10];
float height[50];
char name[10];
```

टाईप ऐरे में संग्रहित होने वाले तत्वों के प्रकार को बताता है जैसे कि int, float, और char एवंम् वैरिएबल नेम ऐरे के नाम को बताता हैजैसे कि height, group और name है साईज ऐरे में सग्रंहित किये जा सकने वाले तत्वों कि अधिकतम संख्या को इंगित करता है। सी प्रोग्रामिंग भाषा, करेक्टर स्ट्रींग को करेक्टर के ऐरे के रूप में ही प्रबंध करता है।

पांच तत्वों के लिए एक ऐरे की घोषणा:int number[5];

नीचे दिखाये अनुसार कंप्यूटर मैमोरी में पांच भंडारण स्थानों को आरक्षित कर लेता है क्यों की ऐरे का साइज 5 है–



एकल या एक आयामी ऐरे का प्रारंभ: एक ऐरे के डिक्लेरेशन के बाद उसके तत्व प्रारंभ किये जाते है सी प्रोग्रामिंग में एक ऐरे निम्न चरणों में प्रारंभ किया जा सकता है:

– कंपाइल टाइम

– रन टाइम

कंपाइलटाइम प्रारंभ:जब एक ऐरे के डिक्लेरेशन के साथ उसे प्रारंभ किया जाता है तो ऐरे निम्न प्रकार से प्रारंभहोगा : type array-name[size] = { list of values };

लिस्ट में मानो को कोमा से अलग किया जाता है उदहारण के लिए

int number[3] = { 0,5,4 };

ऊपर दिए गए स्टेटमेंट में 3 आकार का एक नंबर नाम का ऐरे है और हर तत्व को वैल्यू आवंटित होगी। लिस्ट में वैल्यू की संख्या ऐरे साईज की तुलना में कम है, तो यह केवल कुछ ऐरे तत्वो को वैल्यू आवंटित करेगा। शेष तत्वों को स्वचालित रूप से शून्य आवंटित हो जायेगा।

याद रखें, यदि घोषित आकार की तुलना में अधिक वैल्यू है, तो एक त्रुटि का उत्पादन होगा।

रन टाइम प्रारंभ: एक ऐरे को स्पष्ट रूप से चलाने के लिए रन टाइम आरंभ किया जा सकता है उदाहरण के लिए निम्नलिखित सी प्रोग्राम के खंड पर विचार करें।

```
for(i=0;i<10;i++)
{
scanf(" %d ", &x[i] );
}
```

उदाहरण के लिए ऊपर कीबोर्ड से वैल्यू देते है रन टाइम में लूपिंग स्टेटमेंट जरुरी है असाइनमेंट ऑपरेटर की सहायता से एक एक वैल्यू ऐरे में स्टोर करते है

एक आयामी ऐरे का प्रोग्राम:

/ * ऐरे में तत्वो को स्टोर करने और प्रिंट करने के लिए सरल C प्रोग्राम * /

```
#include<stdio.h>
void main()
{
int array[5],i;

printf("Enter 5 numbers to store them in array \n");

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

{
scanf("%d",&array[i]);

}

printf("Element in the array are - \n \n");

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

{
printf("Element stored at a[%d] = %d \n",i,array[i]);

}
getch();

}
```

इनपुट(Input)– Enter 5 elements in the array – 23 45 32 25 45

आउटपुट (Output) – Elements in the array are –

Element stored at a[0]-23

Element stored at a[1]-45

Element stored at a[2]-32

Element stored at a[3]-25

Element stored at a[4]-45

बहु आयामी ऐरे: ऐरे का ऐरे एक बहुआयामी ऐरे कहलाता है। सामान्य रूप से बहुआयामी ऐरे की घोषणा निम्न प्रकार से होती है।

type variable-name [size1] [size2] --- [sizeN];

बहुआयामी ऐरे का सरलतम रूप दो आयामी ऐरे है उदाहरण:

int x [3] [4];

उपरोक्त x एक दो आयामी (2डी) ऐरे है। ऐरे में 12 तत्व है। यहाँ x 3 पंक्ति के साथ तालिका के रूप में ऐरे है। और प्रत्येक पंक्ति में 4 स्तंभ है।

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

दो आयामी (2डी) ऐरे का प्रारंभ:एक आयामी ऐरे की तरह, 2डी ऐरे को भी दोनों प्रकार (कंपाइल टाइम व रन टाइम) से प्रारंभ किया जा सकता है

कंपाइल टाइम आरंभीकरण –जब एक ऐरे के डिक्लेरेशन के साथ उसे प्रारंभ किया जाता है तो दो आयामी ऐरे निम्न प्रकार से प्रारंभहोगा :

int table-[2][3] = {
{ 0, 2, 5}

```
{ 1, 3, 0}
};
```

रन टाइम आरंभीकरण–एक ऐरे को स्पष्ट रूप से चलाने के लिए रन टाइम आरंभ किया जा सकता है दो आयामी ऐरे को लूप स्ट्रक्चर की मदद से आरम्भ करते है। दो लूप स्ट्रक्चर उपयोग में ली जाती है। जिसमें आउटर लुप पक्ंति के लिए एवंम इनर लूप कॉलम के उपयोग में आती है। उदाहरण के लिए निम्नलिखित सी प्रोग्राम के खंड पर विचार करें।

```
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&ar1[i][j]);
}
}
```

2डी ऐरे का प्रोग्राम:

/ * 2–डी ऐरे का सी प्रोग्राम * /

```
        #include<stdio.h>
        #include<conio.h>
         void main()
{
          int array[3][3],i,j,count=0;

           /* Run time Initialization */
for(i=1;i<=3;i++)

          {
for(j=1;j<=3;j++)
{
count++;
array[i][j]=count;
printf("%d \t",array[i][j]);
}
printf("\n");
}
```

getch();

}

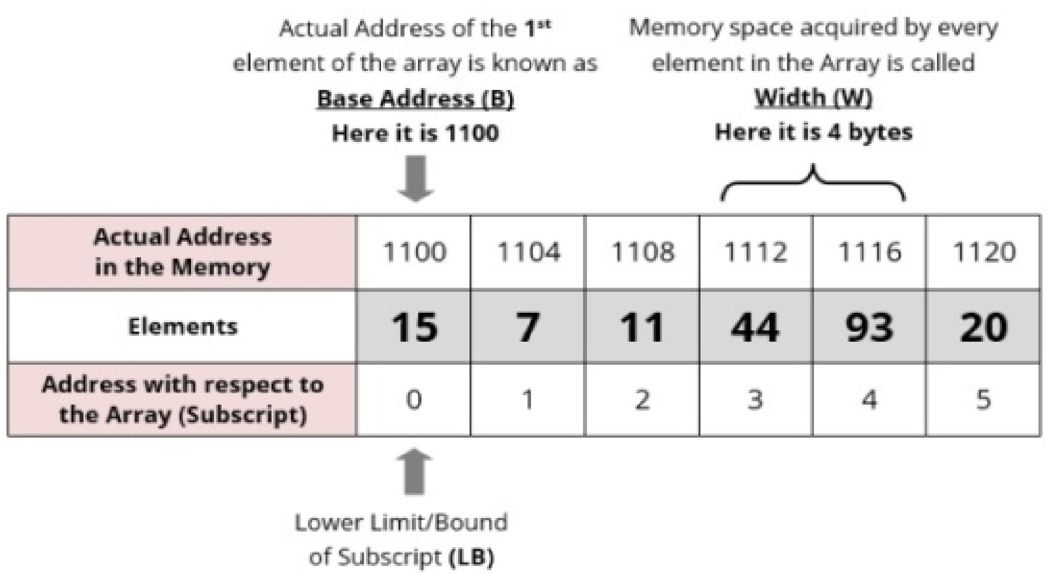
Output –

1 2 3

4 5 6

7 8 9

एकल (एक) आयामी ऐरे में पता गणना:



एक ऐरे "A [I]" के एक तत्व की गणना निम्न सूत्र के उपयोग से करते है–

Address of A [I] = B + W * (I – LB)

Where,

B = आधार पता

W = ऐरे में उपस्थित एक तत्व की स्टोरेज साईज (बाइट में)

I = जिस तत्व का पता ज्ञात करना है उसका सबस्क्रिप्ट

LB = नीचली सीमा/उपलब्ध नहीं होने पर शून्य माने 0 (शून्य)

उदाहरण**:**

एक ऐरे [1300 --- --1900] का आधार पता 1020 और प्रत्येक तत्व का आकार मैमोरी में 2 बाइट्स के रूप में है। B [1700], का पता गणना किजिए

उत्तर **:**

दिए गए मान निम्न है B = 1020, LB = 1300, W = 2, I= 1700

A [I] का पता = B + W * (I – LB)

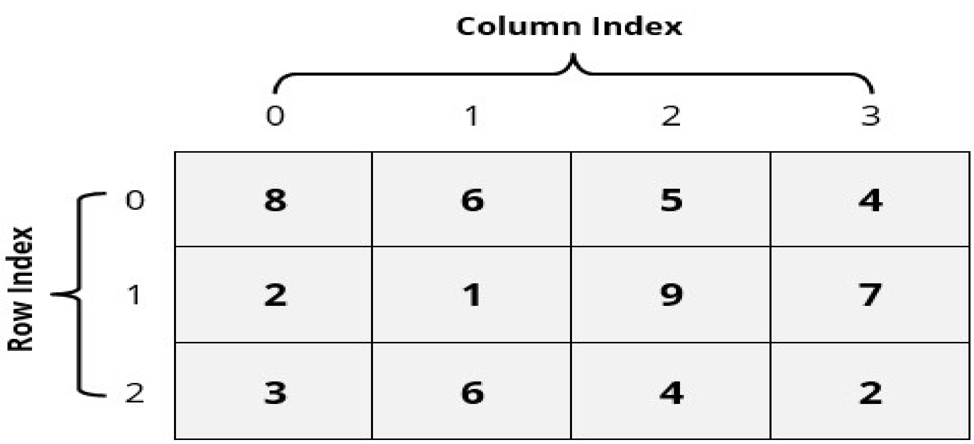
= 1020 + 2 * (1700 – 1300)

= 1020 + 2 * 400

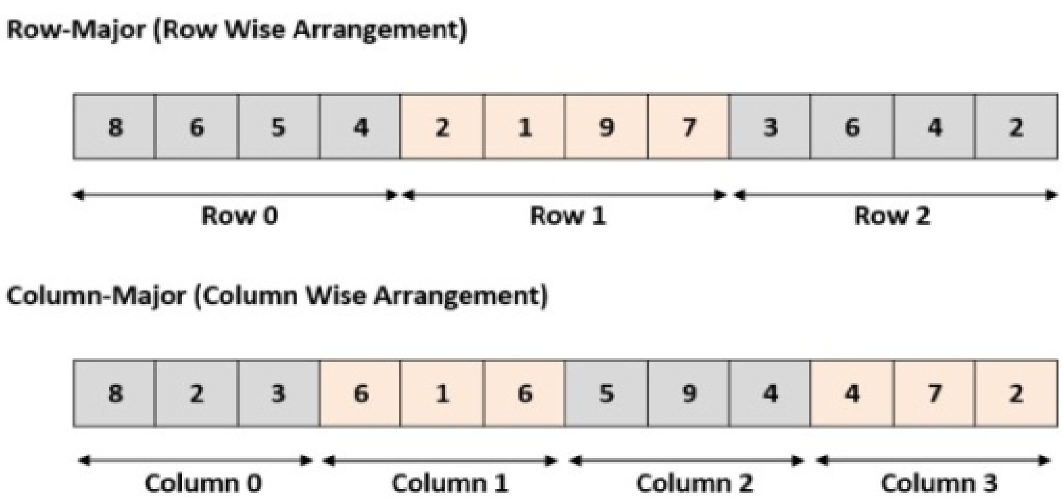
= 1020 + 800

= 1820 [Ans]

मल्टी (दो) आयामी ऐरे में पता गणना: मैमोरी में एक 2–डी ऐरे के तत्वों को संग्रह करते समय इन्हें क्रमिक मैमोरी लोकेशन आवंटित किये जाते है। इसलिए उसके भंडारण को सक्षम करने के लिए2–डी ऐरे को लीनियराइज करते है। लीनियराइज करने के दो तरीके होते है। रो (पंक्ति) मेजर और कॉलम (स्तंभ) मेजर।



Two-Dimensional Array



ऐरे के किसी तत्व "A [I] [J]" के पता की गणना नीचे दिए गए दो प्रकार से की जा सकती है

(क) पंक्ति प्रमुख प्रणाली (Row Major System)

(ख) कॉलम प्रमुख प्रणाली (Column Major System)

पंक्ति प्रमुख प्रणाली:

पंक्ति प्रमुख प्रणाली में एक लोकेशन का पता निम्न सूत्र का उपयोग करके किया जाता है:

A [I][J] तत्व का पता= B + W * [N * (I – Lr)+ (J – Lc)]

स्तंभ (कॉलम) प्रमुख प्रणाली:

कॉलम प्रमुख प्रणाली में एक लोकेशन का पता निम्न सूत्र का उपयोग करके किया जाता है:

A [I][J] तत्व का पता = B + W * [(I – Lr)+ M * (J – Lc)]

यहाँ पे,

B = आधार पता

I = जिस तत्व का पता ज्ञात करना है उसका पक्ंति सबस्क्रिप्ट

J = जिस तत्व का पता ज्ञात करना है उसका स्तंभ सबस्क्रिप्ट

W = ऐरे में उपस्थित एक तत्व की स्टोरजे साईज (बाइट में)

Lr = पंक्ति की नीचली सीमा/उपलब्ध नहीं होने पर शून्य माने 0 (शून्य)

Lc = स्तंभ की नीचली सीमा / उपलब्ध नहीं होने पर शून्य माने 0 (शून्य)

M = मैट्रिक्स में पंक्तियों की संख्यां

N = मैट्रिक्स में स्तंभों की संख्यां

नोट: एक मैट्रिक्स में आमतौर पर पंक्तियों और स्तंभों की संख्या (इस तरह A [20] [30] या A[40] [60]), दी जाती है

लेकिन A[Lr- – – – – Ur, Lc- – – – – Uc] के रूप में दिया जाता है, तो पंक्तियों और स्तंभों की संख्या निम्नलिखित तरीकों का उपयोग कर गणना कर करते हैं

पंक्तियाँ (M) की संख्या की गणना = (Ur – Lr) + 1

स्तंभों (N) की संख्या की गणना = (Uc – Lc) + 1

एवंम अन्य प्रक्रिया आवश्यकतानुसार पंक्ति मेजर और स्तंभ मेजर समान रहेगी

उदाहरण:

एक ऐरे ,X[-15 .... .10, 15 .... 40],जिसमें एक बाईट स्टोरेज की आवश्यकता है। अगर शुरुआत स्थान 1500 है तो लोकेशन x[15] [20], ज्ञात किजिए।

M = (Ur – Lr) + 1 = [10 – (- 15)] +1 = 26

N = (Uc – Lc) + 1 = [40 – 15)] +1 = 26

(I) उपरोक्त की कॉलम मेजर गणना:

दिए गए मान हैं B = 1500, W = 1 byte, I = 15, J = 20, Lr = -15, Lc = 15, M = 26

A [I][J] का पता = B + W * [(I – Lr) + M * (J – Lc)]

= 1500 + 1 * [(15 – (-15)) + 26 * (20 – 15)] = 1500 + 1 * [30 + 26 * 5] = 1500 + 1 * [160] = 1660 [Ans]

(II) उपरोक्त की पक्तिं मेजर गणना:

दिए गए मान हैं B = 1500, W = 1 byte, I = 15, J = 20, Lr = -15, Lc = 15, N = 26

A [I][J] का पता = B + W * [N * (I – Lr) + (J – Lc)]

= 1500 + 1* [26 * (15 – (-15))) + (20 – 15)] = 1500 + 1 * [26 * 30 + 5] = 1500 + 1 * [780 + 5] = 1500 + 785

= 2285 [Ans]

ऐरे पर बुनियादी ऑपरेशन: निम्नलिखित ऑपरेशन ऐरे पर किये जा सकते है

(क) ट्रर्वसिंग **(Traversing):** एक डाटा स्ट्रक्चर में मौजूद सभी डेटा तत्वों के प्रसंस्करण (प्रोसेसिंग) को ट्रर्वसिंग कहते है।

(ख) इनर्सशन **(Insertion):** इनर्सशन का अर्थ एक डाटा स्ट्रक्चर में एक नये डेटा तत्व को जोडना।

(ग) डिलिशन **(deletion):** डिलिशन का अर्थ एक डाटा स्ट्रक्चर में एक डेटा तत्व को हटाना यदि वह मौजूद है।

(घ) सर्च **(Search):** एक डाटा स्ट्रक्चर में निर्दिष्ट डेटा तत्व को खोजने को सर्च कहते है।

(ड़) अपडेट **(Update)**– दिए गए सूचकांक में एक तत्व अपडेट करता है।

ट्रर्वसिंग **(Traversing)**: एक डाटा स्ट्रक्चर में मौजूद सभी डेटा तत्वों के प्रसंस्करण (प्रोसेसिंग) को ट्रर्वसिंग कहते है। निम्नलिखित एल्गोरिथ्म से लीनियर ऐरे को ट्रर्वस कर सकते है।

1. Repeat For I = LB to UB
2. Apply PROCESS to A[I]

[End of For Loop]

3. Exit

इनर्सशन **(Insertion)**: इनर्सशन का अर्थ एक डाटा स्ट्रक्चर में एक नये डेटा तत्व को जोडना। निम्नलिखित एल्गोरिथ्म से लीनियर ऐरे में इनर्सट कर सकते है।

Algorithm: Let LA be a Linear Array (unordered) with N elements and K is a positive integer such that K<=N. Following is the algorithm where ITEM is inserted into the Kth position of LA −

1. Start
2. Set J = N
3. Set N = N+1
4. Repeat steps 5 and 6 while J >= K
5. Set LA[J+1] = LA[J]
6. Set J = J-1
7. Set LA[K] = ITEM
8. Stop

C Program for Insertion:

```
#include <stdio.h>

main() {
int LA[] = {1,3,5,7,8};
int item = 10, k = 3, n = 5;
int i = 0, j = n;

printf("The original array elements are :\n");

for(i = 0; i<n; i++) {
printf("LA[%d] = %d \n", i, LA[i]);
}

n = n + 1;

while( j >= k) {
LA[j+1] = LA[j];
j = j - 1;
}

LA[k] = item;

        printf("The array elements after insertion :\n");
```

```
for(i = 0; i<n; i++) {
printf("LA[%d] = %d \n", i, LA[i]);
}
}
```

जब उपरोक्त कोड कम्पायल और रन होगा तो आउटपुट निम्नानुसार होगा:

The original array elements are :

LA[0] = 1

LA[1] = 3

LA[2] = 5

LA[3] = 7

LA[4] = 8

The array elements after insertion :

LA[0] = 1

LA[1] = 3

LA[2] = 5

LA[3] = 10

LA[4] = 7

LA[5] = 8

डिलिशन **(deletion)**: डिलिशन का अर्थ एक डाटा स्ट्रक्चर में एक डेटा तत्व को हटाना यदि वह मौजूद है।निम्नलिखित एल्गोरिथ्म से लीनियर ऐरे के तत्वों को डिलीट कर सकते है।

Algorithm: consider LA is a linear array with N elements and K is a positive integer such that K<=N. Following is the algorithm to delete an element available at the Kth position of LA.

1. Start
2. Set J = K
3. Repeat steps 4 and 5 while J < N
4. Set LA[J-1] = LA[J]
5. Set J = J+1
6. Set N = N-1

7. Stop

C Program for Deletion:

```
#include <stdio.h>
main() {
int LA[] = {1,3,5,7,8};
int k = 3, n = 5;
int i, j;
printf("The original array elements are :\n");

for(i = 0; i<n; i++) {
printf("LA[%d] = %d \n", i, LA[i]);
}

j = k;

while( j < n) {
LA[j-1] = LA[j];
j = j + 1;
}

n = n -1;

printf("The array elements after deletion :\n");

for(i = 0; i<n; i++) {
printf("LA[%d] = %d \n", i, LA[i]);
}
}
```

जब उपरोक्त कोड कम्पायल और रन होगा तो आउटपुट निम्नानुसार होगाः

The original array elements are :

LA[0] = 1

LA[1] = 3

LA[2] = 5

LA[3] = 7

LA[4] = 8

The array elements after deletion :

LA[0] = 1

LA[1] = 3

LA[2] = 7

LA[3] = 8

सर्च **(Search)**: एक डाटा स्ट्रक्चर में निर्दिष्ट डेटा तत्व को खोजने को सर्च कहते है लीनियर ऐरे में सर्च दो प्रकार की होती है

(क) लीनियर सर्च

(ख) बाइनरी सर्च

लीनियर सर्च **:**लीनियर सर्च बुनियादी और सरल सर्च एल्गोरिथ्म है। लीनियर सर्च में तत्व और मान को तब तक सर्च करते है जब तक मिल नहीं जाता हैं इसमें दिये गये तत्वों को ऐरे में उपलब्ध सभी तत्वों से तुलना करते हैं एवंम मिलान होने पर ऐरे इनडेक्स का मान प्राप्त होता है अन्यथा –1। लीनियर सर्च सॉटेड और अनसॉटेड मानों पर लागु करते है। जब तत्वों की संख्या कम हो। लिनीयर सर्च की निम्नलिखित एल्गोरिथ्म है।

Consider LA is a linear array with N elements and K is a positive integer such that K<=N. Following is the algorithm to find an element with a value of ITEM using sequential search

1. Start
2. Set J = 0
3. Repeat steps 4 and 5 while J < N
4. IF LA[J] is equal ITEM THEN GOTO STEP 6
5. Set J = J +1
6. PRINT J, ITEM
7. Stop

C Program for Searching:

```
#include <stdio.h>
main() {
int LA[] = {1,3,5,7,8};
int item = 5, n = 5;
```

```
int i = 0, j = 0;

printf("The original array elements are :\n");

for(i = 0; i<n; i++) {
printf("LA[%d] = %d \n", i, LA[i]);
}

while( j < n){
if( LA[j] == item ) {
break;
}

j = j + 1;
}

printf("Found element %d at position %d\n", item, j+1);
}
```

जब उपरोक्त कोड कम्पायल और रन होगा तो आउटपुट निम्नानुसार होगाः

Output

The original array elements are:

LA[0] = 1

LA[1] = 3

LA[2] = 5

LA[3] = 7

LA[4] = 8

Found element 5 at position 3

बाइनरी सर्चः बाइनरी सर्च सोंटेड ऐरे या लिस्ट पर लागू किया जाता है। सर्वप्रथम हम दिये गये तत्व की ऐरे के बीच के तत्व से तुलना करते है यदि तत्व के मान का मिलान हो जाता है तो ऐरे का इनडेक्स मान रिर्टन करता है। यदि तत्व का मान कम है तो निचले आधे हिस्से में होगा अन्यथा ऊपरी आधे हिस्से में होगा। बाइनरी सर्च का उपयोग तत्वों की संख्या अधिक होने पर किया जाता हैं निम्नलिखित बाइनरी सर्च की एल्गोरिथ्म है।

Compare x with the middle element.

If x matches with middle element, we return the mid index.

Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.

Else (x is smaller) recur for the left half.

C Prgram for Binary Search:

```
#include <stdio.h>

#define MAX 20

// array of items on which linear search will be conducted.
int intArray[MAX] =
{1,2,3,4,6,7,9,11,12,14,15,16,17,19,33,34,43,45,55,66};

void printline(int count) {
int i;

for(i = 0;i <count-1;i++) {
printf("=");
}

printf("=\n");
}

int find(int data) {
int lowerBound = 0;
int upperBound = MAX -1;
int midPoint = -1;
int comparisons = 0;
int index = -1;

while(lowerBound <= upperBound) {
printf("Comparison %d\n" , (comparisons +1) );
printf("lowerBound : %d, intArray[%d] =
%d\n",lowerBound,lowerBound,
intArray[lowerBound]);
printf("upperBound : %d, intArray[%d] =
%d\n",upperBound,upperBound,
intArray[upperBound]);
```

```
comparisons++;

// compute the mid point
// midPoint = (lowerBound + upperBound) / 2;
midPoint = lowerBound + (upperBound - lowerBound) / 2;

// data found
if(intArray[midPoint] == data) {
index = midPoint;
break;
} else {
// if data is larger
if(intArray[midPoint] < data) {
// data is in upper half
lowerBound = midPoint + 1;
}
// data is smaller
else {
// data is in lower half
upperBound = midPoint -1;
}
}
}
printf("Total comparisons made: %d" , comparisons);
return index;
}

void display() {
int i;
printf("[");

// navigate through all items
for(i = 0;i<MAX;i++) {
printf("%d ",intArray[i]);
}

printf("]\n");
}
```

```
main() {
printf("Input Array: ");
display();
printline(50);

//find location of 1
int location = find(55);

// if element was found
if(location != -1)
printf("\nElement found at location: %d" ,(location+1));
else
printf("\nElement not found.");
}
```

जब उपरोक्त कोड कम्पायल और रन होगा तो आउटपुट निम्नानुसार होगाः

Input Array: [1 2 3 4 6 7 9 11 12 14 15 16 17 19 33 34 43 45 55 66]

Comparison 1

lowerBound : 0, intArray[0] = 1

upperBound : 19, intArray[19] = 66

Comparison 2

lowerBound : 10, intArray[10] = 15

upperBound : 19, intArray[19] = 66

Comparison 3

lowerBound : 15, intArray[15] = 34

upperBound : 19, intArray[19] = 66

Comparison 4

lowerBound : 18, intArray[18] = 55

upperBound : 19, intArray[19] = 66

Total comparisons made: 4

Element found at location: 19

अपडेट **(Update)–** दिए गए सूचकांक में एक तत्व अपडेट करता है। निम्नलिखित एल्गोरिथ्म एक तत्व को ऐरे में अपडेट करता है

Consider LA is a linear array with N elements and K is a positive integer such that K<=N.

Following is the algorithm to update an element available at the Kth position of LA.

1. Start
2. Set LA[K-1] = ITEM
3. Stop

C Program for Updation:

```
#include <stdio.h>
main() {
int LA[] = {1,3,5,7,8};
int k = 3, n = 5, item = 10;
int i, j;

printf("The original array elements are :\n");

for(i = 0; i<n; i++) {
printf("LA[%d] = %d \n", i, LA[i]);
}

LA[k-1] = item;

printf("The array elements after updation :\n");

for(i = 0; i<n; i++) {
printf("LA[%d] = %d \n", i, LA[i]);
}
}
```

जब उपरोक्त कोड कम्पायल और रन होगा तो आउटपुट निम्नानुसार होगाः

Output

The original array elements are :

LA[0] = 1

LA[1] = 3

LA[2] = 5

LA[3] = 7

LA[4] = 8

The array elements after updation :

LA[0] = 1

LA[1] = 3

LA[2] = 10

LA[3] = 7

LA[4] = 8

'C' में कैरेक्टर स्ट्रिंग:स्ट्रिंग्स वास्तव में एक आयामी कैरेक्टर ऐरे है। जिसके अन्त में Null कैरेक्टर '\0' होता है।

निम्नलिखित डिक्लेरेशन और इनिशलाईजेशन "Hello" शब्द से बनी स्ट्रिंग क्रियेट करता है। ऐरे के अन्त में Null कैरेक्टर जोडनें के लिए करैक्टर स्ट्रिंग की लम्बाई "Hello" शब्द में कुल अक्षरो की संख्या से एक अधिक है।

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
char greeting[] = "Hello";
```

इस स्ट्रिंग का मैमोरी में प्रदर्शन निम्नानुसार है।

Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

वास्तव में आप स्ट्रिंग के अंत में Null करैक्टर को इर्न्सट नहीं करते, सी कम्पायलर स्वचालित रूप से स्ट्रिंग के अंत में Null करैक्टर को इर्न्सट कर देता है जब यह ऐरे को इनिशिलाईज करता है। उपरोक्त स्ट्रिंग को प्रिन्ट करना –

```
#include <stdio.h>
int main () {
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
printf("Greeting message: %s\n", greeting );
```

```
return 0;
}
```

जब उपरोक्त कोड कम्पायल और रन होगा तो आउटपुट निम्नानुसार होगाः

```
Greeting message: Hello
```

सी भाषा में निम्नलिखित फंक्शन होते है।

क्र.स.	फंक्शन	उदेश्य
1	strcpy(s1, s2);	स्ट्रिंग s2को s1 में कोपी करता है।
2	strcat(s1, s2);	स्ट्रिंग s1के अन्त में s2 को जोड़ता है।
3	strlen(s1);	s1स्ट्रिंग की लम्बाई को बताता है।
4	strcmp(s1, s2);	यदि स्ट्रिंग s1एवंम s2समान है तो 0 रिर्टन करता है अन्यथा यदि s1<s2 तो 0 से कम, और यदि s1>s2 तो 0 से ज्यादा रिर्टन करता है।
5	strchr(s1, ch);	स्ट्रिंग s1में करेक्टर ch की पहली आवृत्ति का पोंइन्टर देता हैं।
6	strstr(s1, s2);	स्ट्रिंग s1में स्ट्रिंग s2 की पहली आवृत्ति का पोंइन्टर देता हैं।

निम्नलिखित उदाहरण उपरोक्त फंक्शनों में से कुछ का उपयोग दर्शाते है।

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

int main () {

char str1[12] = "Hello";
char str2[12] = "World";
char str3[12];
int  len ;

/* copy str1 into str3 */
strcpy(str3, str1);
printf("strcpy( str3, str1) :  %s\n", str3 );
```

```
/* concatenates str1 and str2 */
strcat( str1, str2);
printf("strcat( str1, str2):   %s\n", str1 );

/* total lenghth of str1 after concatenation */
len = strlen(str1);
printf("strlen(str1) :  %d\n", len );

return 0;
}
```

जब उपरोक्त कोड कम्पायल और रन होगा तो आउटपुट निम्नानुसार होगा:

```
strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10
```

स्टैटिक और डायनेमिक मैमोरी अलोकेशन (आबंटन): डायनेमिक मैमोरी अलोकेशन रन टाइम पर किया जाता हैं। जबकि स्टैटिक मैमोरी अलोकेशन रन टाइम से पहले किया जाता है, लेकिन वेरिएबल के मानों को रन टाइम पर बदला जा सकता है

स्टैटिक मैमोरी अलोकेशन रनिंग टाइम में बचत करता है, लेकिन सभी मामलों में यह संभव नहीं हो सकता है। डायनेमिक मैमोरी अलोकेशन इसकी मैमोरी को हीप में सहेजता है और स्टैटिक मैमोरी अलोकेशन इसके डेटा को मैमोरी के डेटा सगमेंन्ट में सहेजता है।

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
//static allocation example using integer array.
int arr[5]; // static memory allocation memory allocated before execution, the size of array should be initialized
for ( int j = 0; j<5; j++) //Waste of memory can be occured.
{
printf("Enter number for Static Array %d: " ,j);
scanf("%d", &arr[j]);
}
```

```
printf("\nThe Static Array is: \n");
for ( int j = 0; j<5; j++)
{
printf("The value of %d is %d\n", j, arr[j]);
}

//dynamic allocation example using integer array
int* array;
int n, i;
printf("\n---------------------------\n\nDynamic Allocation\n");
printf("Enter the number of elements: ");
scanf("%d", &n);
array = (int*) malloc(n*sizeof(int)); //memory is allocated during the execution of the program
//Less Memory space required.
for (i=0; i<n; i++) {
printf("Enter number %d: ", i);
scanf("%d", &array[i]);
}

printf("\nThe Dynamic Array is: \n");

for (i=0; i<n; i++) {
printf("The value of %d is %d\n", i, array[i]);
}
printf("Size= %d\n", i);

system("PAUSE");
return 0;
}
```

मैमोरी अलोकेशन के फंक्शनः सी प्रोग्रामिंग भाषा में मैमोरी अलोकेशन और प्रबन्धन के लिए कई फंक्शन होते है। यह फंक्शन <stdlib.h>हेडर फाईल में होते है।

क्र.स.	फंक्शन और विवरण
1	void *calloc(int num, int size); num तत्वों का एक ऐरे आवंटित करता है। जिसकी size बाइट में होगी।
2	void free(void *address); दिये गये पत्ते के मैमोरी ब्लोक को फ्री करता है।
3	void *malloc(int num); num तत्वों का एक ऐरे आवंटित करता है। जिसकी size बाइट में होगी एवंम् अनइनिशियलाईज होगी।
4	void *realloc(void *address, int newsize); दुबारा मैमोरी अलोकेशन के काम आता है।

निम्नलिखित प्रोग्राम डायनेमिक मैमोरी अलोकेशन का फंक्शन के साथ उदाहरण है।

1.

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

int main() {

char name[100];
char *description;

strcpy(name, "Zara Ali");

/* allocate memory dynamically */
description = malloc( 200 * sizeof(char) );

if( description == NULL ) {
fprintf(stderr, "Error - unable to allocate required memory\n");
}
else {
```

```
strcpy( description, "Zara ali a DPS student in class 10th");
}
printf("Name = %s\n", name );
printf("Description: %s\n", description );
}
```

जब उपरोक्त कोड कम्पायल और रन होगा तो आउटपुट निम्नानुसार होगाः

```
Name = Zara Ali
Description: Zara ali a DPS student in class 10th
```

उपरोक्त प्रोग्राम को निम्नानुसार calloc(); फंक्शन का उपयोग करके भी लिखा जा सकता है।

```
calloc(200, sizeof(char));
```

आप किसी भी साईज की मैमोरी अलोकेट कर सकते है। और आपका इस पर पूर्ण नियंत्रण होता है। जबकि इसके विपरीत ऐरे में एक बार इसकी साईज को परिभाषित करने के बाद बदल नहीं सकते है।

2.

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

int main() {

char name[100];
char *description;

strcpy(name, "Angad");

/* allocate memory dynamically */
description = malloc( 30 * sizeof(char) );

if( description == NULL ) {
fprintf(stderr, "Error - unable to allocate required memory\n");
}
```

```
else {
strcpy( description, "Angad is a cute Boy.");
}

/* suppose you want to store bigger description */
description = realloc( description, 100 * sizeof(char) );

if( description == NULL ) {
fprintf(stderr, "Error - unable to allocate required memory\n");
}
else {
strcat( description, "He is in 1st Class");
}

printf("Name = %s\n", name );
printf("Description: %s\n", description );

/* release memory using free() function */
free(description);
}
```

जब उपरोक्त कोड कम्पायल और रन होगा तो आउटपुट निम्नानुसार होगाः

Name = Angad

Description: Angad is a cute Boy.He is in 1st Class.

'C' में पोंइन्टर (Pointers):पोंइन्टरएक वेरिएबल है जिसका मान अन्य वेरिएबल का ऐड्रस (मैमोरी लोकेशन का ऐड्रस) होता है। किसी भी अन्य वेरिएबल और कॉन्सटेन्ट कि तरह पोंइन्टर को भी उपयोग में लेने से पहले इसे डिक्लेयर करना आवश्यक है। पोंइन्टर डिक्लेरेशन का साधारण रूप निम्नानुसार है।

type *var-name;

यहाँ type पोंइन्टर प्रकार है। यह सी के मान्य डेटा प्रकारों में से होना चाहिये और var-name पोंइन्टर वेरिएबल का नाम है। पोंइन्टर को डिक्लेयर करने के लिए उपयोग में लिया जाने वाला तारांकन "*" गुणा में उपयोग लिये जाने वाले तारांकन चिन्ह के समान ही है कुछ मान्य पोंइन्टर डिक्लेरेशन निम्नानुसार है

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

सभी पोंइन्टरों के मानों का डेटा प्रकार एक ही होता है, यह एक लोंग हेक्साडेसिमल नम्बर होता है और यह एक मैमोरी पत्ते को प्रदर्शित करता है। विभिन्न डेटा प्रकारों के पोंइन्टरों के बीच फर्क सिर्फ उनके द्वारा पोंइट किये गये वेरिएबल और कॉन्सटेन्ट के डेटा प्रकार में ही होता है।

निम्नलिखित उदाहरण पोंइन्टर वेरिएबल के उपयोग को प्रदर्शित करता है।

```
#include <stdio.h>
int main () {

int  var = 20;   /* actual variable declaration */
int  *ip;        /* pointer variable declaration */

ip = &var;  /* store address of var in pointer variable*/

printf("Address of var variable: %x\n", &var  );

/* address stored in pointer variable */
printf("Address stored in ip variable: %x\n", ip );

/* access the value using the pointer */
printf("Value of *ip variable: %d\n", *ip );

return 0;
}
```

जब उपरोक्त कोड कम्पायल और रन होगा तो आउटपुट निम्नानुसार होगाः

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

NULL पोंइन्टरः

यदि आपके पास पोंइन्टर वेरिएबल को असाइन करने के लिए कोई सही पत्ता नहीं हो तब इसे NULL वैल्यू असाइन करनी चाहिऐ। यह वेरिएबल डिक्लेरेशन के समय पर किया जाता है। पोंइन्टर वेरिएबल जिसे NULL वैल्य असाइन की गई है उसे NULL पोंइन्टर कहते हैं।

NULL पोंइन्टर एक शून्य मान वाला कॉन्सटेन्ट होता है और यह कई मानक लाईब्रेरी में परिभाषित हैं

निम्नलिखित प्रोग्राम पर विचार करे।

```
#include <stdio.h>
int main () {
int  *ptr = NULL;
printf("The value of ptr is : %x\n", ptr  );
return 0;
}
```

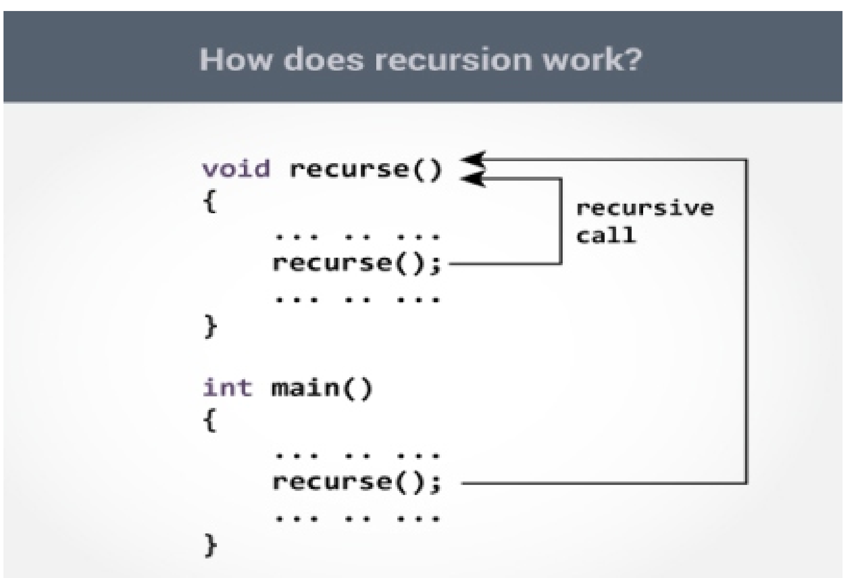
जब उपरोक्त कोड कम्पायल और रन होगा तो आउटपुट निम्नानुसार होगाः

The value of ptr is 0

पोंइन्टर को जाँचने के लिए if स्टेटमेंट का उपयोग निम्नानुसार किया जा सकता हैं:

```
if(ptr)     /* succeeds if p is not null */
if(!ptr)    /* succeeds if p is null */
```

'C' मे Recursion(रिकर्शन):रिकर्शन स्व–समान (self-similar) तरीके से आइटमों को दोहराने की एक प्रक्रिया है। प्रोग्रामिंग भाषाओं में यदि एक प्रोग्राम आपको एक ही फंक्शन के अंदर उसी फंक्शन को कॉल करने की अनुमति देता है तब इसे रिकर्शन फंक्शन के रूप में जाना जाता है। अन्य शब्दो में जब एक फंक्शन अपने आप को ही कॉल करता है तो इसे रिकर्सिव फंक्शन कहते है।



रिकर्शन निम्नानुसार काम करता है:

```
void recurse()
{
... .. ...
recurse();
... .. ...
}

int main()
{
... .. ...
recurse();
... .. ...
}
```

रिकर्शन फंक्शन के लिए शर्तें:

1. सभीरिकर्शन फंक्शन में एक बेस मानदंड (Termination condition) होनी आवश्यक है और इसके लिए वह खुद को कॉल नहीं करना चाहिए।
2. जब भी एक फंक्शन खुद को कॉल करे तब यह बेस मानदंड के करीब आये।

रिकर्शन के उदाहरण निम्नानुसार है:

(क) फिबोनैकी सीरीज (Fibonacci Series)

(ख) बिनोमिअल कोफिसिएंट (Binomial coefficient)

(ग) GCD

(क) फिबोनैकी सीरीज

फिबोनैकी सीरीज कि संख्याऐ निम्नानुसार पूर्णांक अनुक्रम में हैं:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 ....

फिबोनैकी सीरीज में पहले दो नंबर 0 और 1 हैं और प्रत्येक बाद वाला नंबर पिछले दो नंबरों का योग है। गणितीय संदर्भ में, फिबोनैकी संख्याओं कि Nth टर्म निम्नानुसार रेकर्रेंस संबंध द्वारा परिभाषित है:

fibonacci(N) = फिबोनैकी संख्याओं कि Nth टर्म

fibonacci(N) = fibonacci(N - 1) + fibonacci(N - 2);

जहाँ पर, fibonacci(0) = 0 और fibonacci(1) = 1

निम्नलिखित प्रोग्राम फिबोनैकी संख्याओं कि Nth टर्म निकालने के लिए रिकर्शन का उपयोग करता है। Nth फिबोनैकी संख्या निकालने के लिए यह सर्वप्रथम (N-1)th और (N-2)th ज्ञात करता है और फिर दोनों का योग करता है।

रिकर्शन का उपयोग करके फिबोनैकी सीरीज को Nth टर्म तक प्रिंट करने के लिए सी प्रोग्रामः

निम्नलिखित प्रोग्राम, उपयोगकर्ता से स्केनफ फंक्शन का उपयोग करके इनपुट के रूप में फिबोनैकी सीरीज के पदों की संख्या को लेता है। इसमें ऊपर बातये अनुसार रिकर्शन का उपयोग करते हुए 'fibonacci' नामक यूजर परिभाषित फंक्शनहै जो इनपुट के रूप में एक पूर्णांक N लेता है और Nth फिबोनैकी संख्या लौटता है। जब पदों की संख्या <2 होगी तब रिकर्शन समाप्त हो जाएगा क्योंकि फिबोनैकी सीरीज के पहले दो क्रम 0 और 1 होते है।

```
#include <stdio.h>

#include <conio.h>

int fibonacci(int term);
int main(){
int terms, counter;
printf("Enter number of terms in Fibonacci series: ");
scanf("%d", &terms);
/*
*  Nth term = (N-1)th therm + (N-2)th term;
*/
printf("Fibonacci series till %d terms\n", terms);
for(counter = 0; counter < terms; counter++){
printf("%d ", fibonacci(counter));
}
getch();
return 0;
}
/*
* Function to calculate Nth Fibonacci number
* fibonacci(N) = fibonacci(N - 1) + fibonacci(N - 2);
*/
int fibonacci(int term){
/* Exit condition of recursion*/
if(term < 2)
return term;
return fibonacci(term - 1) + fibonacci(term - 2);
}
```

Program Output

Enter number of terms in Fibonacci series: 9
Fibonacci series till 9 terms
0 1 1 2 3 5 8 13 21

(ख) बिनोमिअल कोफिसिएंट का प्रोग्रामः

```
#include<stdio.h>
int fact(int);
void main()
{
int n,r,f;
printf("enter value for n & r\n");
scanf("%d%d",&n,&r);
if(n<r)
printf("invalid input");
else f=fact(n)/(fact(n-r)*fact(r));
printf("binomial coefficient=%d",f);
}
int fact(int x)
{
if(x>1)
returnx*fact(x-1);
else return 1;
}
```

The sum of the 10 binomial coefficients of the form C 9 k?
45

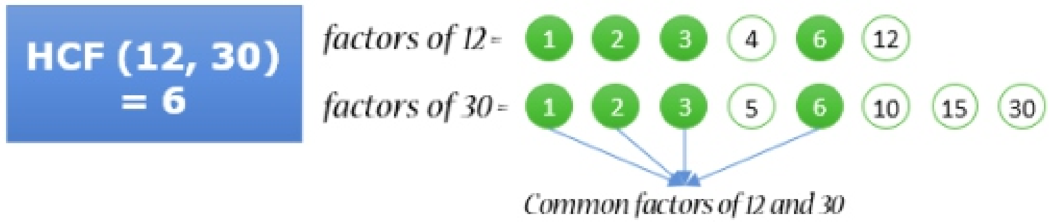
(ग) दो नंबरो का **GCD** निकालना**:**

Input first number: 10

Input second number: 15

Output GCD: 5

Logic to find GCD using recursion



GCD निकालना के लिए इयूक्लिडियन एल्गोरिथ्म:

```
Begin:
function gcd(a, b)
If (b = 0) then
return a
End if
Else
return gcd(b, a mod b);
End if
End function
End
```

Program to find GCD using recursion:

C program to find GCD (HCF) of two numbers using recursion:

```
#include <stdio.h>

/* Function declaration */
int gcd(int a, int b);

int main()
{
int num1, num2, hcf;

/* Reads two numbers from user */
printf("Enter any two numbers to find GCD: ");
scanf("%d%d", &num1, &num2);

hcf = gcd(num1, num2);
```

```
printf("GCD of %d and %d = %d\n", num1, num2, hcf);

return 0;
}
```

Recursive approach of euclidean algorithm to find GCD of two numbers:

```
int gcd(int a, int b)
{
if(b == 0)
return a;
else
return gcd(b, a%b);
}
```

Output:
Enter any two numbers to find GCD: 12
30
GCD of 12 and 30 = 6

महत्वपूर्ण बिंदु

- ऐरेसमरूप डेटा तत्वों के परिमित क्रमो का एक संग्रह है जोकि क्रमिक मैमोरी स्थानों में संग्रहित होते है।
- मैमोरी में एक 2–डी ऐरे के तत्वों को संग्रह करते समय इन्हें क्रमिक मैमोरी लोकेशन आवंटित किये जाते है।
- एक डाटा स्ट्रक्चर में मौजूद सभी डेटा तत्वों के प्रसंस्करण (प्रोसेसिंग) को ट्र्वसिंग कहते है।
- पोंइन्टरएक वेरिएबल है जिसका मान अन्य वेरिएबल का ऐड्रस (मैमोरी लोकेशन का ऐड्रस) होता है।

अभ्यासार्थ प्रश्न

वस्तुनिष्ठ प्रश्न

प्र1 लीनियर सर्च में वर्स्ट केस कब होता है?

(अ) आइटम ऐरे के बीच में हो

(ब) आइटम ऐरे में बिल्कुल भी नहीं हो

(स) आइटम ऐरे में पीछे हो

(द) आइटम ऐरे में पीछे हो या बिल्कुल नहीं हो

प्र2 लीनियर सर्च एल्गोरिथ्म की जटिलता है?

(अ) $O(n^2)$ (ब) O(logn)

(स) O(nlogn) (द) O(n+1)

प्र3 लीनियर सर्च एल्गोरिथ्म में औसत मामले कब होते हैं

(अ) जब आइटम ऐरे के बीच में कहीं हो

(ब) जब आइटम ऐरे में बिल्कुल भी नहीं हो

(स) जब आइटम ऐरे के पिछले हिस्से में हो

(द) जब आइटम ऐरे के पिछले हिस्से में हो या नहीं हो

प्र4 दिये गये मान से किसी तत्व के स्थान को ढूँढना है:

(अ) ट्रर्वस (ब) सर्च

(स) सॉट (द) इनमें से कोई भी नहीं

प्र5 निम्न में से कौन सा मामला जटिलता सिद्धांत में मौजूद नहीं है

(अ) सबसे अच्छा मामला (ब) सबसे खराब मामला

(स) औसत के मामले (द) अशक्त मामले

लघुत्तरात्मक प्रश्न

प्र1 बाइनरी खोज की समय जटिलता क्या है?

प्र2 ऐरे से आपका क्या मतलब है?

प्र3 स्ट्रिंग क्या है?

प्र4 सूचक से आपका क्या मतलब है?

प्र5 गतिशील स्मृति आबंटन क्या है?

निबंधात्मक प्रश्न

प्र1 उदाहरण के साथ दो आयामी ऐरे समझाओ।

प्र2 विस्तार से malloc फंक्शन को समझाओं।

प्र3 रिकर्शन के लिए कोनसा डेटा स्ट्रक्चर का उपयोग किया जाता हैं?

प्र4 बाइनरी सर्च,लीनियर सर्च से क्यो बेहतर है?

प्र5 Character स्ट्रिंग को समझाओ।

उत्तरमाला

उत्तर 1: द उत्तर 2: द उत्तर 3: अ

उत्तर 4: ब उत्तर 5: द