

ऐरे

पाठ्यपुस्तक के प्रश्न

वस्तुनिष्ठ प्रश्न

प्रश्न 1. 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

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

प्रश्न 2. लीनियर सर्च एल्गोरिथ्म की जटिलता है।

- (अ) $O(n^2)$
- (ब) $O(\log n)$
- (स) $O(n \log n)$
- (द) $O(n + 1)$

उत्तर: (द) $O(n + 1)$

प्रश्न 3. लीनियर सर्च एल्गोरिथ्म में औसत मामले कब होते हैं?

- (अ) जब आइटम ऐरे के बीच कहीं हो
- (ब) जब आइटम ऐरे में बिल्कुल भी नहीं हो
- (स) जब आइटम ऐरे के पिछले हिस्से में हो
- (द) जब आइटम ऐरे में पिछले हिस्से में हो या नहीं हो

उत्तर: (अ) जब आइटम ऐरे के बीच कहीं हो

प्रश्न 4. दिये गये मान से किसी तत्व के स्थान को ढूँढ़ना है

- (अ) टूर्वस
- (ब) सर्च
- (स) सॉर्ट
- (द) इनमें से कोई भी नहीं

उत्तर: (ब) सर्च

प्रश्न 5. निम्न में से कौन-सा मामला जटिलता सिद्धान्त में मौजूद नहीं है

- (अ) सबसे अच्छा मामला
- (ब) सबसे खराब मामला
- (स) औसत के मामले
- (द) अशक्त मामले

उत्तर: (द) अशक्त मामले

लघु उत्तरीय प्रश्न

प्रश्न 1. बाइनरी खोज की समय जटिलता क्या है?

उत्तर- बाइनरी खोज की समय जटिलता निम्न है

- Best case – $O(1)$ i.e., constant
- Average case – $O(\log n)$.
- Worst case – $O(\log n)$

लेकिन सामान्यतया: जब कभी बाइनरी खोज की समय जटिलता पूछी जाती है तो वह by default worst case मान कर $O(\log n)$ मानी जाती है।

प्रश्न 2. ऐरे से आपका क्या मतलब है?

उत्तर- ऐरे (Array) – ऐरे, समरूप डेटा तत्वों के परिमित क्रमों का एक संग्रह है जो कि क्रमिक मेमोरी स्थानों में संग्रहित होते हैं

यहाँ शब्द

परिमित का अर्थ डेटा रेंज निर्धारित होनी चाहिए।

क्रम का अर्थ डेटा निरन्तर मेमोरी स्थानों में संग्रहित किया जाना चाहिए।

समरूप का अर्थ डेटा एक ही प्रकार का होना चाहिए।

ऐरे दो प्रकार का होता है ।

- एकल या एक आयामी ऐरे
- बहु आयामी ऐरे

प्रश्न 3. स्ट्रिंग क्या है?

उत्तर- स्ट्रिंग वास्तव में एक आयामी करैक्टर ऐरे है, जिसके अन्त में Null करैक्टर '0' होता है।

उदाहरण: ऐरे के अन्त में Null करैक्टर जोड़ने के लिए करैक्टर स्ट्रिंग की लम्बाई "Hello" शब्द में कुल अक्षरों की संख्या से एक अधिक है।

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

प्रश्न 4. सूचक से आपको क्या मलतब है?

उत्तर- सूचक (Pointers)-पोइन्टर एक वैरिएबल है जिसका मान अन्य वैरिएबल का एड्रेस (मैमोरी लोकेशन का एड्रेस) होता है। किसी भी अन्य वैरिएबल और कॉन्स्टेन्ट की तरह पोइन्टर को भी उपयोग में लेने से पहले इसे डिक्लेयर करना आवश्यक है। पोइन्टर डिक्लेरेशन का साधारण रूप निम्नानुसार है।

```
type * var-name;
```

यहाँ type पोइन्टर प्रकार है। यह सी के मान्य डेटा प्रकारों में से होना चाहिये और var-name पोइन्टर वैरिएबल का नाम है। पोइन्टर को डिक्लेयर करने के लिए उपयोग में लिया जाने वाला तारांकन '***' गुणा में उपयोग लिये जाने वाले तारांकन चिह्न के समान ही है कुछ मान्य पोइन्टर 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 पोइन्टरों के मानों का डेटा प्रकार एक ही होता है, यह एक लॉन्ग (long) हेक्साडेसिमल नम्बर होता है और यह एक मैमोरी पते को प्रदर्शित करता है। विभिन्न डेटा प्रकारों के पोइन्टरों के बीच फर्क सिर्फ उनके पोइंट किये गये वैरिएबल और कॉन्स्टेन्ट के डेटा प्रकार में ही होता है।

प्रश्न 5. गतिशील स्मृति आबंटन क्या है?

उत्तर- गतिशील स्मृति आबंटन (Dynamic Memory Allocation)-गतिशील स्मृति आबंटन रन टाइम पर किया जाता है। गतिशील स्मृति आबंटन इसकी मैमोरी को हीप में सहेजता है।

कम्प्यूटर में हमें एक ऐसी व्यवस्था की जरूरत होती है जिससे यदि Data बढ़ते हैं तो नए वैरिएबल्स create हो सकें और यदि Data घटते हैं तो किसी पुराने वैरिएबल्स को Delete किया जा सके ताकि उस वैरिएबल के द्वारा Reserve की गई जगह का कोई अन्य Program उपयोग कर सके। इसी व्यवस्था को गतिशील स्मृति आबंटन (Dynamic Memory Allocation) कहते हैं।

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

```
#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 होगा :

Name = Zara Ali

Description : Zara ali a DPS student in class 10th

निबंधात्मक प्रश्न

प्रश्न 1. उदाहरण के साथ दो आयामी ऐरे समझाओ।

उत्तर- दो आयामी ऐरे-ऐरे का ऐरे एक बहु आयामी ऐरे कहलाता है। बहु आयामी ऐरे का सरलतम रूप दो आयामी ऐरे कहलाता है।

उदाहरण: $\text{int} \times [3] [4]$;

	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 प्रारम्भ-एक आयामी ऐरे की तरह, 2डी ऐरे को भी दोनों प्रकार (कम्पाइल टाइम व रन टाइम) से प्रारम्भ किया जा सकता है।

कंपाइल टाइम आरम्भीकरण – जब एक ऐरे के डिक्लेरेशन के साथ उसे प्रारम्भ किया जाता है तो दो आयामी ऐरे निम्न प्रकार से प्रारम्भ होगा

```
int table[2][3] = {
{0, 2, 5}
{ 1, 3, 0}
};
```

रन टाइम आरम्भीकरण-एक ऐरे को स्पष्ट रूप से चलाने के लिए रन टाइम आरम्भ किया जा सकता है। दो आयामी ऐरे को लूप स्ट्रक्चर की मदद से आरम्भ करते हैं। दो लूप स्ट्रक्चर उपयोग में ली जाती हैं। जिसमें आउटर लूप पंक्ति के लिए एवं इनर लूप कॉलम के उपयोग में आती है। उदाहरण के लिए निम्नलिखित सी प्रोग्राम के खण्ड पर विचार करते हैं

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

प्रश्न 2. विस्तार से Malloc फंक्शन को समझाओ।

उत्तर- Malloc() फंक्शने ।

इस फंक्शन का प्रयोग करके हम मैमोरी का एक ब्लॉक (Block) create कर सकते हैं और उसे किसी वैरिएबल को allocate कर सकते हैं।

जब हम डायनेमिक मैमोरी एलोकेशन के लिए इस फंक्शन का प्रयोग करते हैं, तब ये फंक्शन मैमोरी में किसी specified डाटा टाइप का एक मैमोरी ब्लॉक बनाता है और उस मैमोरी लोकेशन या इस malloc() फंक्शन द्वारा बनाए गए ब्लॉक स्पेस का एड्रेस return करता है।

इस Address को उसी डेटा टाइप प्रकार के पोइन्टर वैरिएबल में store किया जाता है और इस पोइन्टर का उपयोग करके आवश्यकतानुसार डेटा इनपुट किया जाता है। इस फंक्शन का सिन्टेक्स निम्नानुसार होता है

ptr = (Data Type *) malloc (size of (Data type);

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

```
#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 आउपुट निम्नानुसार होगा:

Name = Zara Ali

Description: Zara ali a DPS student in class 10th

प्रश्न 3. रिकर्शन के लिए कौन-सा डेटा स्ट्रक्चर का उपयोग किया जाता है।

उत्तर- रिकर्शन के लिए Stacks का उपयोग किया जाता है। ये LIFO (Last In First Out) के सिद्धान्त पर काम करता है, जिसमें सभी Data एक सिरे से जोड़े या घटाए जाते हैं। इस सिरे को TopOfThe Stack (TOS) कहा जाता है। Stack में किसी Data को एक सिरे से जोड़ा या घटाया जा सकता है। इसलिए जब किसी Stack में Data को इनपुट किया जाता है, तो सबसे पहले इनपुट किया गया Data सबसे बाद में प्राप्त होता है व सबसे बाद में इनपुट किया गया Data सबसे पहले प्राप्त होता है। इसलिए हम यह कह सकते हैं कि Stack, Data इकाइयों की वह List होती है, जिसमें एक ही सिरे से Data को जोड़ा या घटाया जा सकता है। Stack के दूसरे सिरे को BOS (Bottom Of The Stock) कहते हैं।

उदाहरण: माना 10 किताबें हैं। इन किताबों को क्रम से एक के ऊपर एक रखते हैं। अब यदि हम किसी किताब को पढ़ना चाहें तो सबसे पहले हमें वो किताब उठानी पड़ती है, जिसे सबसे बाद में रखा था और सबसे पहले रखी गई किताब का उपयोग हम बाद में कर सकते हैं। यही Stack की अवधारणा है।

प्रश्न 4. बाइनरी सर्च, लीनियर सर्च से क्यों बेहतर है?

उत्तर- लीनियर सर्च व बाइनरी सर्च: लीनियर सर्च बुनियादी और सरल सर्च एल्गोरिथ्म है। लीनियर सर्च में तत्व और मान को तब तक सर्च करते हैं जब तक मिल नहीं जाता। इसमें दिये गये तत्वों को ऐरे में उपलब्ध सभी तत्वों से तुलना करते हैं एवं मिलान होने पर ऐरे इन्डेक्स का मान प्राप्त होता है। अन्यथा -1। लीनियर सर्च सॉर्टेड और अनसॉर्टेड मानों पर लागू करते हैं जब तत्वों की संख्या कम हो।

बाइनरी सर्च सॉर्टेड ऐरे या लिस्ट पर लागू किया जाता है। सर्वप्रथम हम दिये गये तत्व की ऐरे के बीच के तत्व से तुलना करते हैं यदि तत्व के मान का मिलान हो जाता है तो ऐरे का इन्डेक्स मान रिटर्न करता है। यदि तत्व का मान कम है तो निचले आधे हिस्से में होगा अन्यथा ऊपरी आधे हिस्से में होगा। बाइनरी सर्च का प्रयोग तत्वों की संख्या अधिक होने पर किया जाता है।

बाइनरी सर्च ordering comparisons परफॉर्म करता है और लीनियर सर्च equality comparison परफॉर्म करता है।

प्रश्न 5. Character स्ट्रिंग को समझाओ।

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

निम्नलिखित डिक्लरेशन और इनिशलाइजेशन "Hello" शब्द से बनी स्ट्रिंग Create करता है। ऐरे के अन्त में Null करैक्टर जोड़ने के लिए करैक्टर स्ट्रिंग की लम्बाई "Hello" शब्द में कुल अक्षरों 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 हम स्ट्रिंग के अन्त में Null करैक्टर को इन्सर्ट नहीं करते, सी कम्पायलर स्वचालित रूप से स्ट्रिंग के अन्त में Null करैक्टर को इन्सर्ट 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 होगा :

Greeting message : Hello

सी भाषा में निम्नलिखित फंक्शन होते है।

फंक्शन	उद्देश्य
strcpy(s1, s2);	स्ट्रिंग s2 को s1 में कॉपी करता है। strcat(s1, s2);
strcat(s1, s2);	स्ट्रिंग s1 के अन्त में s2 को जोड़ता है।
strlen(s1);	s1 स्ट्रिंग की लम्बाई को बताता है।
strcmp(s1, s2);	यदि स्ट्रिंग s1 एवं s2 समान है तो 0 रिटर्न करता है अन्यथा यदि s1 < s2 तो 0 से कम, और यदि s1 > s2 तो 0 से ज्यादा रिटर्न करता है।
strchr (s1,ch);	स्ट्रिंग s1 में करैक्टर ch की पहली आवृत्ति का पोइन्टर देता है।
strstr (s1, s2);	स्ट्रिंग s1 में स्ट्रिंग s2 की पहली आवृत्ति का पोइन्टर देता है।

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

```
#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);
    /* concatenated str1 and str2 */
    strcat(str1, str2);
    printf("strcat(str1, str2): %s\n", str1);
    /* total length 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(str 3, str1): Hello
strcat (str 1, str 2): Hello World
strlen(str 1): 10
```

अन्य महत्वपूर्ण प्रश्न

अतिलघु उत्तरीय प्रश्न

प्रश्न 1. ऐरे कितने प्रकार के होते हैं?

उत्तर- ऐरे दो प्रकार के होते हैं

- एकल या एक आयामी ऐरे,
- बहु आयामी ऐरे।

प्रश्न 2. एकल या एक आयामी ऐरे क्या होता है?

उत्तर- 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।

प्रश्न 3. एक आयामी ऐरे को किस प्रकार डिक्लेयर किया जाता है? उदाहरण सहित बताइए।

उत्तर- एक आयामी ऐरे को निम्न प्रकार डिक्लेयर किया जाता है- `type variable-name[size];`

उदाहरण: `char name [10];`
`int marks[5];`
`float height[20];`

प्रश्न 4. ऐरे को उपयोग करने से पहले डिक्लेयर करना क्यों आवश्यक है?

उत्तर- ऐरे को उपयोग करने से पहले डिक्लेयर किया जाना चाहिए ताकि कम्पाइलर उनके लिए मेमोरी में स्पेस आबंटित कर सके।

प्रश्न 5. बहुआयामी ऐरे से आप क्या समझते हैं?

उत्तर- ऐरे का ऐरे एक बहु आयामी ऐरे कहलाता है।

प्रश्न 6. बहु आयामी ऐरे की घोषणा किस प्रकार की जाती है? उदाहरण भी दीजिए।

उत्तर- सामान्य रूप से बहुआयामी ऐरे की घोषणा निम्न प्रकार से होती है
`type variable-name [size 1] [size 2]..... [size N];`

उदाहरण: `int x [3][4];`

प्रश्न 7. बहुआयामी ऐरे को लीनियराइज करने के कितने तरीके होते हैं? नाम बताइए।

उत्तर- लीनियराइज करने के दो तरीके होते हैं-रो (पंक्ति) मेजर और कॉलम (स्तम्भ) मेजर।

प्रश्न 8. सर्च से आप क्या समझते हैं? एक लीनियर ऐरे में सर्च कितने प्रकार की होती है?

उत्तर- एक डाटा स्ट्रक्चर में निर्दिष्ट डेटा तत्व को खोजने को सर्च कहते हैं। एक लीनियर ऐरे में सर्च दो प्रकार की होती है।

- लीनियर सर्च,
- बाइनरी सर्च।

प्रश्न 9. पोइन्टर क्या होता है?

उत्तर- पोइन्टर एक वैरिएबल है, जिसका मान अन्य वैरिएबल का एड्रेस (मैमोरी लोकेशन का एड्रेस) होता है।

प्रश्न 10. क्या पोइन्टर को उपयोग करने से पहले डिक्लेयर करना आवश्यक है?

उत्तर- हाँ, किसी भी अन्य वैरिएबल और कॉन्स्टेंट की तरह पोइन्टर को भी उपयोग करने से पहले डिक्लेयर करना आवश्यक है। पोइन्टर डिक्लेरेशन को साधारण रूप निम्नानुसार है
`type * var – name;`

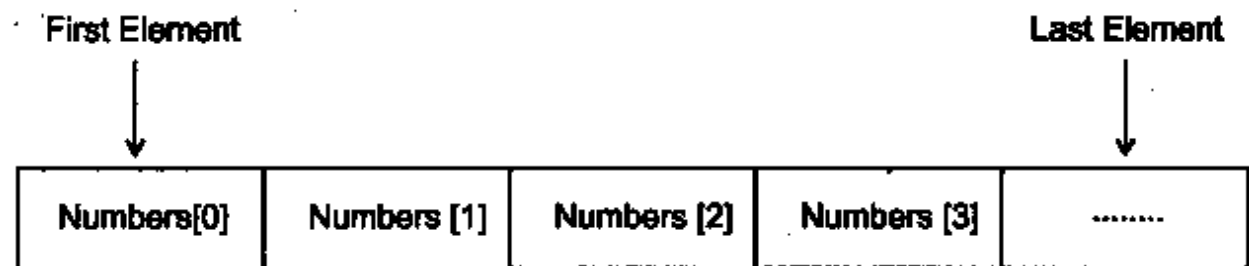
प्रश्न 11. NULL पोइन्टर किसे कहते हैं?

उत्तर- पोइन्टर वैरिएबल जिसे NULL वैल्यू असाईन की गई है उसे NULL पोइन्टर कहते हैं।

लघु उत्तरीय प्रश्न

प्रश्न 1. एकल या एक आयामी ऐरे पर संक्षिप्त टिप्पणी लिखिए।

उत्तर- 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 आवंटित कर सके। ऐरे को निम्न प्रकार से डिक्लेयर किया जाता है:

`type variable-name [size];`

उदाहरण

`int group [10];`

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

प्रश्न 2. सी प्रोग्रामिंग में एक ऐरे के प्रारम्भ होने के विभिन्न चरण लिखिए।

अथवा

एकल या एक आयामी ऐरे के प्रारम्भ के विषय में बताइए।

उत्तर- एकल या एक आयामी ऐरे का प्रारम्भ: एक ऐरे के डिक्लेरेशन के बाद तत्त्व प्रारम्भ किये जाते हैं सी प्रोग्रामिंग में एक ऐरे निम्न चरणों में प्रारम्भ किया जा सकता है:

- कंपाइल टाइम
- रन टाइम

कंपाइल टाइम प्रारम्भ: जब एक ऐरे के डिक्लेरेशन के साथ उसे प्रारम्भ किया जाता है तो ऐरे निम्न प्रकार से प्रारम्भ होगा:

```
type array-name [size] = {list of values};  
लिस्ट में मानों को कोमा से अलग किया जाता है उदाहरण के लिए  
int number [3] = {0, 5, 4}
```

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

ध्यान रखने वाली बात है, यदि घोषित आकार की तुलना में अधिक वैल्यू है, तो एक त्रुटि का उत्पादन होगा। रन टाइम प्रारम्भ: एक ऐरे को स्पष्ट रूप से चलाने के लिए रन टाइम आरम्भ किया जा सकता है उदाहरण के लिए निम्नलिखित C प्रोग्राम के खण्ड पर विचार करते हैं।

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

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

प्रश्न 3. ऐरे में तत्त्वों को स्टोर करने और प्रिन्ट करने के लिए सरल C प्रोग्राम लिखिए।

अथवा

एक आयामी ऐरे का प्रोग्राम लिखिए।

उत्तर- एक आयामी ऐरे का प्रोग्राम:

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

```
#include
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]);
}
print("Element in the array are-\n\n");
for (i=0;<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

प्रश्न 4. सी प्रोग्राम में एक बहुआयामी ऐरे के प्रारम्भ होने के विभिन्न चरण लिखिए।

अथवा

बहु आयामी ऐरे के प्रारम्भ के विषय में लिखिए।

उत्तर- बहु आयामी (2डी) ऐरे का प्रारम्भ : एक आयामी ऐरे की तरह, 2डी ऐरे को भी दोनों प्रकार (कमाइल टाइम व रन टाइम) से प्रारम्भ किया जा सकता है।

कंपाइल टाइम आरम्भीकरण-

जब एक ऐरे के डिक्लेरेशन के साथ उसे प्रारम्भ किया जाता है तो दो आयामी ऐरे निम्न प्रकार से प्रारम्भ होगा:

```
int table- [2] [3] = {  
{0, 2, 5}  
{1, 3, 0}  
};
```

रन टाइम आरम्भीकरण – एक ऐरे को स्पष्ट रूप से चलाने के लिए टाइम आरम्भ किया जा सकता है दो आयामी ऐरे को लूप स्ट्रक्चर की मदद से आरम्भ करते हैं। दो लूप स्ट्रक्चर उपयोग में ली जाती है। जिसमें लूप पक्ति के लिए एवं इनर लूप कॉलम के उपयोग में आती है। उदाहरण के लिए निम्नलिखित सी प्रोग्राम के खण्ड पर विचार करते हैं

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

प्रश्न 5.

एक 2 डी ऐरे का प्रोग्राम लिखिए।

उत्तर-

2डी ऐरे का प्रोग्राम:

/* 2-डी ऐरे का सी प्रोग्राम */

```
#include  
#include  
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

```

प्रश्न 6. एकल आयामी ऐरे में पता की गणना का सूत्र बताइए।

उत्तर- एकल (एक) आयामी ऐरे में पता गणना:

Actual Address of the 1st element of the array is known as **Base Address(B)**
Here It is 1100

↓

Memory space acquired by every element in the Array is called **Width (W)**
Here It is 4 bytes

┌──────────┴──────────┐

Actual Address in the Memory	1100	1104	1108	1112	1116	1120
Elements	15	7	11	44	93	20
Address with respect to the Array (Subscript)	0	1	2	3	4	5

↑

Lower Limit/Bound of Subscript (LB)

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

$$\text{Address of A [I]} = B + W * (I - LB)$$

Where, B= आधार पता

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

I= जिस तत्व को पता ज्ञात करना है उसका सबस्क्रिप्ट

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

प्रश्न 7. ऐरे पर कौन-कौन से ऑपरेशन किये जा सकते हैं?

अथवा

ऐरे पर होने वाले कुछ बुनियादी ऑपरेशन के विषय में बताइए।

उत्तर- ऐरे पर बुनियादी ऑपरेशन: निम्नलिखित ऑपरेशन ऐरे पर किये जा सकते हैं।

(क) ट्रवर्सिंग (Traversing): एक डाटा स्ट्रक्चर में मौजूद सभी डेटा तत्वों के प्रसंस्करण (प्रोसेसिंग) को ट्रवर्सिंग कहते हैं।

(ख) इनसर्शन (Insertion): इनसर्शन का अर्थ एक डाटा स्ट्रक्चर में एक नये डेटा तत्व को जोड़ना है।

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

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

(ङ) अपडेट (Update): दिए गए सूचकांक में एक तत्व अपडेट करता है।

प्रश्न 8. निम्नलिखित ऑपरेशन को एल्गोरिथ्म सहित समझाइये।

(i) ट्रवर्सिंग

(ii) इनसर्शन

(iii) डिलीशन ।

उत्तर- (i) ट्रवर्सिंग (traversing): एक डाटा स्ट्रक्चर में मौजूद सभी डेटा तत्वों के प्रसंस्करण (प्रोसेसिंग) को ट्रवर्सिंग कहते हैं। निम्नलिखित एल्गोरिथ्म से लीनियर ऐरे को ट्रवर्स कर सकते हैं

1. Repeat For $I = LB$ to UB
2. Apply PROCESS to $A [I]$
[End of For Loop]
3. Exit

(ii) इनसर्शन (Insertion): इनसर्शन का अर्थ एक डाटा स्ट्रक्चर में एक नये डेटा तत्व को जोड़ना है। निम्नलिखित एल्गोरिथ्म से लीनियर ऐरे में इनसर्ट कर सकते हैं

Algorithm: Let LA be a Linear Array (unordered) with N elements and K is a positive integer such that $N/LK \leq N$. 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 $J \geq K$
5. Set $LA[J+1] = LA[J]$
6. Set $J = J - 1$
7. Set $LA[K] = \text{ITEM}$
8. Stop

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

Algorithm: Consider LA is a linear array with N elements and K is a positive integer such that $K \leq N$. 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

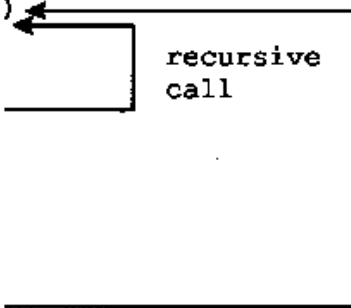
प्रश्न 9. रिकर्शन से आप क्या समझते हैं? यह किस प्रकार कार्य करता है?

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

How does recursion work?

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

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



The diagram illustrates the recursive process. A box labeled 'recursive call' has an arrow pointing from the `recurse();` line in the `main()` function to the `recurse()` function definition. Another arrow points from the `recurse()` function definition back to the `recurse();` line in the `main()` function, indicating the return path.

रिकर्शन निम्नानुसार काम करता है।

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

प्रश्न 10. रिकर्शन फंक्शन के लिए शर्तें बताइए व रिकर्शन के द्वारा फिबोनैकी सीरीज का प्रोग्राम भी लिखिए।

उत्तर- रिकर्शन फंक्शन के लिए शर्तें:

- सभी रिकर्शन फंक्शन में एक बेस मानदण्ड (Termination condition) होनी आवश्यक है और इसके लिए वह खुद को कॉल नहीं करना चाहिए।
- जब भी एक फंक्शन खुद को कॉल करें तब यह बेस मानदण्ड के करीब आये।

प्रोग्राम

निम्नलिखित प्रोग्राम फिबोनैकी संख्याओं की 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 प्रोग्राम: N/L निम्नलिखित प्रोग्राम, उपयोगकर्ता से स्केनफ फंक्शन का उपयोग करके इनपुट के रूप में फिबोनैकी सीरीज के पदों की संख्या को लेता है। इसमें ऊपर बताये अनुसार रिकर्शन का उपयोग करते हुए 'fibonacci' नामक यूजर परिभाषित फंक्शन है 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 हैं।

```
#include
#include
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 term + (N-2) th term) ;
*/
print ("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);
}

```

प्रोग्राम का आउटपुट होगा।

Enter number of terms in Fibonacci series: 9

Fibonacci series till 9 terms

0 1 1 2 3 5 8 13 21

प्रश्न 11. रिकर्शन द्वारा दो नम्बरों का GCD (HCF) निकालने के लिए 'C' भाषा में प्रोग्राम लिखिए।

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

```

#include
/* 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\n", num1, num2, hcf);
return ();
}

```

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);
}

```

आउटपुट होगा:

Enter any two numbers to find GCD: 12 30

GCD of 12 and 30 = 6

निबंधात्मक प्रश्न

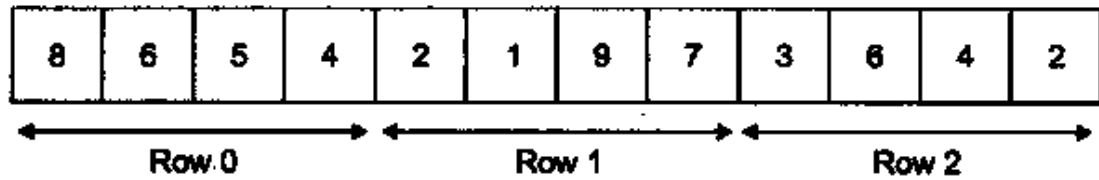
प्रश्न 1. मल्टी (बर्छ आयामी ऐरे में पता गणना किस प्रकार की जाती है?

उत्तर- 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 हैं। इसलिए उसके भण्डारण को सक्षम करने के लिए 2-डी ऐरे को लीनियराइज करते हैं। लीनियराइज करने के दो तरीके होते हैं- से (पंक्ति) मेजर और कॉलम (स्तंभ) मेजर।

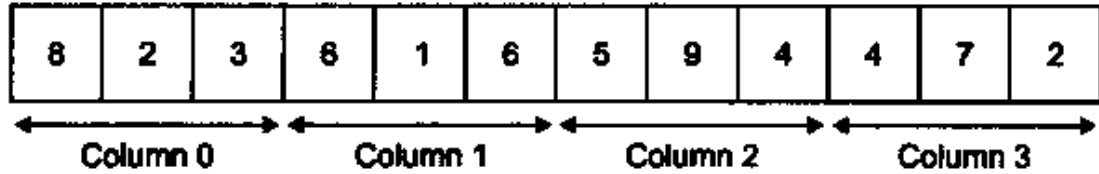
		Column Index			
		0	1	2	3
Row Index	0	8	6	5	4
	1	2	1	9	7
	2	3	6	4	2

Two-Dimensional Array

Row-Major (Row Wise Arrangement)



Column-Major (Column Wise Arrangement)



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)

(क) पंक्ति प्रमुख प्रणाली

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

$$A[I] [J] \text{ तत्व का पता} = B + W * [N * (I-L_r) + (J-L_c)]$$

स्तंभ (कॉलम) प्रमुख प्रणाली:

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

$$A [I] [J] \text{ तत्व को पता} = B + W * [(I-L_r) + M * (J-L_c)]$$

यहाँ पर

B= आधार पला

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

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

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

L_r= पंक्ति की निचली सीमा/उपलब्ध नहीं होने पर शून्य माने 0 (शून्य)

L_c= स्तंभ की निचली सीमा/उपलब्ध नहीं होने पर शून्य माने 0 (शून्य)

M= मैट्रिक्स में पंक्तियों की संख्या

N= मैट्रिक्स में स्तंभों की संख्या

प्रश्न 2. किसी ऐरे में एलीनेन्ट को इन्सर्ट करने के लिए एक प्रोग्राम लिखिए।

उत्तर: C Program for Insertion:

```
#include
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

```

प्रश्न 3. किसी ऐरे में एक एलीमेंट (Element) को डिलीट करने के लिए प्रोग्राम लिखिए।

उत्तर: C Program for Deletion:

```

#include
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 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 deletion:

```

LA [0]=1
LA [1]=3
LA [2]=7
LA [3]=8

```

प्रश्न 4. लीनियर सर्च के लिए एक प्रोग्राम लिखिए।

अथवा

ऐरे में किसी एलीमेन्ट (element) को सर्च करके उसकी पोजिशन ज्ञात करने के लिए प्रोग्राम लिखिए

उत्तर- 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 होगा:

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.

प्रश्न 5. बाइनरी सर्च के लिए प्रोग्राम लिखिए।

अथवा

ऐरे में किसी एलीमेन्ट (element) को सर्च करके उसकी पोजिशन ज्ञात करने के लिए प्रोग्राम लिखिए।

उत्तर: C Program for Binary Search:

```
#include
#define MAX 20

// array of items on which linear search will be conducted.
int Array [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 lower Bound = 0;
    int upper Bound = MAX-1;
    int mid Point = -1;
    int comparisons = 0;
    int index = -1;
    while (lower Bound < = upper Bound)
    {
        printf ("Comparison %d\n", (comparisons + 1));
        printf ("lower Bound : %d, int Array [%d] =%d\n", lower Bound, lower Bound,
        int Array (lower Bound) );
        printf ("upper Bound : %d, int Array [%d] = %d\n", upper Bound, upper Bound,
        int Array (upper Bound));
        comparisons++;
        // compute the mid point
        // mid point = (lower Bound + upper Bound) / 2;
        mid Point = lower Bound + (upper Bound - lower Bound) / 2;
        // data found
        if (int Array [mid Point]= data)
        {
            index = mid Point;
            break;
        }
    }
}
```

```

}
else
{
// if data is larger if
(intArray (mid Point) < data)
{
// data is in upper half
lower Bound = mid Point + 1;
}
// data is smaller
else
{
// data is in lower half
upper Bound = mid Point -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", int Array [i]);
}
printf ("]\n");
}
main ()
{
printf ("Input Array:");
display ();
printfline (50);
// find location of 1
int location = find (55);
// if element was found
if (location != -1)

```

```
printf ("\n Element found at location: %d", (location + 1));  
else  
printf ("\n Element not found.");  
}
```

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

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

Comparison 1

lower Bound: 0, int Array [0] = 1

upper Bound : 19, int Array [19] = 66

Comparison 2

lower Bound : 10, int Array [10] = 15

upper Bound: 19, int Array [19] = 66

Comparison 3

lower Bound : 15, int Array [15] = 34

upper Bound: 19, int Array [19] = 66

Comparison 4

lower Bound : 18, int Array [18] = 55

upper Bound : 19, int Array [19] = 66

Total comparisons made: 4

Element found at location: 19