20. The period of time between an allocation and its subsequent disposal is called

(a) Scope
(b) life time
(c) Binding
(d) longevity

(b) 20⇒ period of time b/w an allocation and its subsequent disposal is called :—

Ans- (b) Life-Time

21. Which of the following is a dangling reference?

(a) Accessing a storage that is already disposed at the request of the user.

(b) Accessing a storage that is already disposed at the request of the processor.

(c) Accessing a variable that is declared but not initialized

(d) None of these

Q 21 ⇒ Dangling Reference —

Ans - (a) Accessing a storage that is already disposed at the request of the user.

⊙ Dangling is a refrence to an object that no longer exists.

⊙ Dangling refrence do not exist in garbage collected languages because objects are only reclaimed when they are no longer accessible. (Only garbage is collected).

Code —

```
(1) int x = 1000; // Creates memory location
(2) int *p = x;
(3) printf("%d", *p); // Valid - Print (20)
(4) delete p;
(5) printf("%d", *p); // Error, because p is
                      // inaccessible OR dangling.
```

dangling pointer

Teacher's Signature .....

22. Consider the following program segments:

```
char *a, *b, c[10], d[10];
a = b ; / * line 1 * /
b = c ;  / * line 2 * /
c = d ; / * line 3 * /
d = a ; / * line 4 * /
```

Which of the following have errors?

(a) line 1

(b) line 2

(c) line 3 and line 4

(d) none has errors

(C) 22⇒ Error — ?

```
char *a, *b, c[10], d[10];
(1) a = b;  ---> // Valid
(2) b = c;  ---> // Valid
(3) c = d;  ---> // Error
(4) d = a;  ---> // Error
```

Ans - (C) → We can not assign the address of one array to anather array and address of a variable to an array.

23. The use of macro in the place of functions:

(a) reduces execution time and code size

(b) increases execution time and reduces code size

(c) increases execution time and increases code size

(d) reduces execution time and increases code size

(a) 23⇒ Use of macro in place of functions—

Ans = (a) Reduce execution time and code size

- A macro is a fragment of code that is given a name.
- You can define a macro in C using #define preprocessor directive.

Example

```
#define pi 3.1415 // Value of pi
#define c 2.99792458 // Speed of light
```

Teacher's Signature .....

24. A function 'q' that accepts a pointer to a character as argument and returns a pointer to an array of integer can be declared as

(a) int *(*q(char*)) [ ]

(b) int *q(char*)[ ]

(c) int (*q) (char*) [ ]

(d) none of these

(a) 24⇒ Function ⟶ q( )

accept *pointer (char)

return pointer to array (int)

Ans ⇒ (a)

```
int *(*q(char*))[]
```

Common Data – 25 & 26 ⟶

```
Procedure Exchange (A : int ; B : int)
begin
    temp : int ;
    temp = A ;
    A = B ;
    B = temp ;
end ;
begin
    m = 2 ; x[m] = 9 ;
    Exchange (m, x[m]) ;
    Write (m, x[2]) ;
end ;
```

// Swaping

Common Data for Q25 & Q26

Consider the following program fragment :

```
Procedure Exchange(A : integer, B : integer)
begin
    temp : integer;
    temp = A;
    A = B;
    B = temp;
end;
begin
    m = 2 ; x[m] = 4 ;
    Exchange(m, x[m]) ;
    write(m, x[2]) ;
end ;
```

25. If the parameters are passed by value, the output will be

(a) 2, 4 (b) 4, 2 (c) 2, 2 (d) 4, 4

(a) 25⇒ Passed by value ⇒ Output (?)

Sol^n

```
m = 2;  x[m] = x[2] = 4;
Exchange(2, 4);   // swap
write(2, 4);
```

→ 2, 4 → output

Ans⇒ (a) (2, 4) because call by value

26. 4, 2 will be the output if the parameters are passed by

(a) reference
(b) name
(c) value
(d) none

(a) 26⇒ Output - 4,2 , parameter = ?

I see ~~ed~~ in Que 25 Call by value.

print same value. Exchange ( ) function swap the value. If we print swaped value then parameter must be -

call by reference.

Ans ⇒ (a) - | Call by reference. |

Teacher's Signature .....

27. What does the following declaration specify?

Char * (*P) (int * a, int (* b) [3])

(a) 'P' is a pointer to a character return pointer to an integer & accepting pointer to 'b' with 3 elements.

(b) 'P' is a function pointer return pointer to an integer and accepting pointer to an array with 3 elements.

(c) 'P' is a function pointer returning pointer to a character and accepts an integer pointer and integer pointer to an array with 3 elements.

(d) There is an error in the declaration.

(C) 27⇒ Declaration specify ?

char *(*p)(int *a, int (*b)[3])

Ans ⇒ (C) char *(*p)( )

↓

p is a function pointer returning pointer to a character.

And accepts an integer pointer and integer pointer to an array with 3 elemo

28. Consider the program given below:

```
main ( )
{
    int c [ ] = { 2, 8, 3, 4, 4, 6, 7, 5};
    int j, *p = c, *q = c;
    for (j = 0; j < 5; ++j)
    {
        printf ("%d", *c);
        ++q;
    }
}
```

What is the output of the program?

(a) 2 2 2 2 2 (b) 2 3 4 7 5

(c) 2 2 2 2 3 (d) 2 3 4 4 6

28⇒

```
main(){
  int c[] = {2, 8, 3, 4, 9, 6, 7, 5};
  int j; *p = c, *q = c;
  for(j=0; j<5; j++)
  {  printf("%d", *c);
     ++q;
  }
}
```

*p = c , *q = c

	0	1	2	3	4	5	6	7
c	2	8	3	4	4	6	7	5

100 102 ...

*p [100] (200) *q [100] (300)

for j = 0 – 4
print (*c)

j = 0 → 2
j = 1 → 2
j = 2 → 2
j = 3 → 2
j = 4 → 2

Ans ⇒ (a) → 2, 2, 2, 2, 2

Teacher's Signature

29. Consider the program below:

```
main ( )
{
    struct KRK
    {
        int x = 3;
        char name [ ] = "Hello";
    };
    struct KRK *p;
    printf ("%d", p → x);
    printf ("%s", p → name);
}
```

What is the output of the program?

(a) 3 Hello

(b) junk output

(c) 3 garbage value

(d) Compiler error

(a) 29⇒

```
main()
{
    struct KRK
    {
        int x=3;
        char name[] = "Hello";
    };
    struct KRK *p;
    printf("%d", p->x);        // print 3
    printf("%s", p->name);     // print Hello
}
```

Output -

(a) 3 Hello

Because — x & name are member variables.