Date. No.

X) Functions:

Number of statements grouped into a single logical unit to perform specific task.

a) Importance:

- → It reduces redundancy.
- → Makes debugging easier.
- → makes program efficient with better memory utilization.

b) Types:

Standard library functions.	User-defined functions.
They are in-built functions.	- Function that are created by the user.

c) Use of functions:

⇒ Function prototype: ~~return type function~~

return_type function_name (parameters datatype);

⇒ Function definition:

```
return_type functionname (parameters)
{
    function body
}
```

⇒ Function calling

function_name (arguments);

Date. No.

(d) Note: → <cmath> header contains pow(a,b) ⇒ a^b

sqrt(a) ⇒ $\sqrt{a}$

- → parameters passed during function call is called actual parameters.
- → parameters passed during function definition is called formal parameters.

c) Function types on values passed:

⇒ Pass by value: function called by directly passing the values of variables by creating copies.

⇒ Pass by reference: function called by passing the address of the variable by passing variable itself.

(d): Default value Parameters:

The value in the function declaration automatically assigned by the compiler if the calling function doesn't pass any value to the argument.

(e): Recursion:

The process of calling a function by itself is called recursion.

To stop recursion, base condition must be satisfied.

Date. No.

(f): Inline functions:

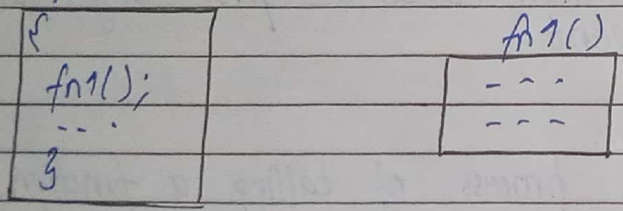
If a function is inline, the compiles places a copy of the code of that function at each point where the function is called is called inline functions.

If we change the body of the inline function, the program needs to be recompiled to replace all the code or else program continues with old code.

* Syntax: inline return-type function-name (parameter)
{
.....function body;
}

for a normal function, the control transfer changes from main function to the function.

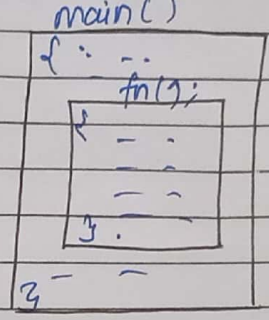
Since there is change in transfer control, the program execution is a bit slow.



Date. No.

for inline function, there is no control transfer and code of the function is copied to that point.

Since no control transfer, there is prevention in time loss.



(g): Function overloading:

Function overloading is a feature of c++ in which two or more functions have same function name but different return type and parameters.

It is an example of polymorphism.

Eg: int add (int a, int b)
float add (float a, float b, float c)
~~int~~ add (float a, int b)
float

Date. No.

Array:

Array is the collection of homogeneous data items stored in contiguous memory location.

(A): For 1-d Array:

Syntax: datatype array_name [array size];

Total size = size of datatype x size of array.

* Passing into function:

i) Prototype: return_type function_name (datatype []);
ii) Definition: return_type function_name (datatype arrayname[]);
iii) Calling: a function_name (variable_name);

(B): 2-d array:

Multi-dimensional array is the array of arrays.

Syntax: datatype array_name [row][column]

*) Passing through function:

i) Prototype: returntype functionname (datatype [][col-size);
ii) Definition: returntype functionname (datatype arrayname [][col-size]);
iii) Calling: functionname (arrayname);

Date. No.

(C): Vectors:

Vectors are dynamic arrays ie. array can be resized when needed.

Header: #include <vector>

Initization: vector <datatype> vectorname; OR,
vector <datatype> vectorname [size];

(*): Operations on vector:

i) v.size () ⇒ gives length of vector.
Eg: vector <int> v = {0,1,2,3,4}
v.size (v) ⇔ 5

ii) v.resize (newsize) ⇒ gives new size.

iii) v.capacity () ⇒ gives memory capacity allocated to vector.

iv) v.pushback (element) ⇒ adds the element at the end of the vector.

v): v.begin () ⇒ position of first element.

vi) v.end () ⇒ position of second element.

vii): v.insert (position, element) ⇒ insert element in given position.

viii) v.popback () ⇒ removes last element of vector.

ix) v.clear () → clears all elements.

x): ~~v.ee~~ v.erase (position) → remove positional element.

Here, v ⇒ vectorname.

~~Initializli~~ *) Initializing:

```
vector <int> v ;
for (int i=0; i<5; i++)
{ int ele;
  cin >> ele;
  v.pushback (element);
}
```

Accessing:

```
for (int ele: v)
{
  cout << ele << endl;
}
```

*) Passing into function : for vectors, easier to prevent errors defining above main.

a) Pass by value:

returntype

Def: ~~datatype~~ functionname (vector <datatype> vectorname)

Calling: function_name (vectorname);

(b): Pass by reference:

Def: returntype function_name (vector<datatype> &vectorname)

Calling: function_name (vectorname);

~~*~~ x) Pointer:

Pointers are special variables that stores the address of a variable.

Syntax: datatype * pointer_name;

Eg: int p=10; int *q = &p;

*) Accessing data through a pointer:

done by using dereference operation (*)

cout << q; ⇒ gives address

cout >> << *q ⇒ 10 → gives value.

(*) Call by reference:

i) Prototype: return_type function-name (datatype*);

ii) Definition: return-type function-name (datatype *variable);

iii) Calling : function_name (&variablename);

*) Note: height +i ≈ &height [i]

*(height+i) ≈ height [i].

(*): Pointers arithmetic:
++ => increment -- => decrement
shifts the pointer forward or backward by the size of the datatype used.

(*): Types:

(a): Wild pointer: The pointer that is not initialized till its first use in program.
Syntax: int *ptr;

(b) Null pointer: The pointer that doesn't pointer to any memory location.
Syntax: int * ptr = NULL;

(c): Dangling pointer: The pointer that points to invalid memory location.

(d): Void pointer: The pointer that doesn't have associated datatype and that can be typecasted. It is called genesic pointer.

Eg: int x = 10;
int *a;
void *ptr;
a = (int*) ptr; cout << *a; => 10

Structure: They are user-defined datatype having different variables of the different datatypes under the same name.

*) Declaring:

struct structuretag
{ member1;
....
};

Using typedef:
typedef struct structuretag
{ ....
....
} newname;

(*) Initializing:

structtag variablename; newname variablename;

(*) Accessing:

structuretag.membername newname.membername;

(*): Passing structure to pointers. (*): Giving pointer to structure:

(i): returntype fname (structname); newname * variablename;

(ii) returntype fname (structname &); Passing to function by reference:

(iii) fname (variable);

(i): returntype functionname(newname*);
(ii) returntype functionname(newname* vn)
(iii) functionname (variablename);

Used to pass structure arrays.