Viva Questions:

Q1. What is the difference between iterative and recursive function call?

A1. An iterative function call is a function call that is made using a loop. A recursive function call is a function call that is made to the same function.

Iterative function calls are typically faster than recursive function calls, but they can be more difficult to write. Recursive function calls can be more concise and easier to read, but they can be slower and can lead to stack overflow errors if the function is called too many times.

Q2. What are formal parameters in functions?

A2. Formal parameters are the variables that are declared in the function definition. They are used to receive the arguments that are passed to the function.

Formal parameters are typically declared in the same way as local variables. They can be of any type, and they can be initialized with default values.

Q3. How is the structure node declared?

A3. A structure node is declared using the struct keyword. The struct keyword is followed by the name of the structure, and then a list of members.

Each member of the structure is declared in the same way as a variable. The member name is followed by the type of the member.

For example, the following code declares a structure called node:

```c
struct node {
  int data;
  struct node* next;
};
```

This structure has two members: data and next. The data member is an integer, and the next member is a pointer to another node structure**.**

Q4. Define a link list.

A4. A linked list is a data structure that consists of a series of nodes. Each node contains data and a pointer to the next node in the list.

Linked lists are a dynamic data structure, which means that they can be resized as needed. This makes them a good choice for storing data that is constantly changing.

Linked lists are also a good choice for storing data that is not stored in contiguous memory. This is because each node in a linked list can be stored anywhere in memory, as long as it has a pointer to the next node in the list.

Q5. Why do we need to store the address of the starting node of a link list for reversing a list.

A5. To reverse a linked list, we need to start at the beginning of the list and follow the pointers to the next node until we reach the end of the list. Once we reach the end of the list, we can start reversing the pointers.

To reverse the pointers, we need to store the address of the starting node. This is because we need to know where to start when we start reversing the pointers.

Without storing the address of the starting node, we would not be able to reverse the pointers in the correct order.

Viva Questions:

Q1. What is the use of the string.h header file?

A1. The string.h header file contains declarations for a number of functions that are used to manipulate strings. These functions include strlen(), strcpy(), strcat(), and strcmp().

Q2. How can we create a header file?

A2. To create a header file, you need to create a text file with the .h extension. The file should contain the declarations for any functions or variables that you want to make available to other source files.

```c
#ifndef MY_H
#define MY_H

void my_function();

#endif
```

Q3. Write the function to find the length of a string.

A3. The function to find the length of a string is strlen(). The strlen() function takes a pointer to a string as its argument and returns the length of the string.

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

int main() {
  char *str = "Hello, world!";
  int len = strlen(str);
  printf("The length of the string is %d\n", len);
  return 0;
}
```

Q4. Write the function to concatenate two strings.

A4. The function to concatenate two strings is strcat(). The strcat() function takes two pointers to strings as its arguments and concatenates the second string to the end of the first string.

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

int main() {
  char *str1 = "Hello, ";
  char *str2 = "world!";
  strcat(str1, str2);
  printf("%s\n", str1);
  return 0;
}
```

Q5. Explain the use of header file.

A5. A header file is a file that contains declarations for functions, variables, or other objects that are used by other source files. Header files are typically used to avoid duplicating code between different source files. For example, if you have a function called my_function() that you want to use in multiple source files, you can define the function in a header file and then include the header file in each of the source files that need to use the function. This will prevent you from having to copy and paste the definition of the function into each of the source files.

Viva Questions:

Q1. What is the Tower of Hanoi problem?

The Tower of Hanoi is a mathematical puzzle where you have three rods and a number of disks of different sizes. The disks are initially stacked in ascending order of size on one rod, the smallest at the top, thus resembling a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:

1. Only one disk can be moved at a time.
2. A larger disk cannot be placed on top of a smaller disk.

The Tower of Hanoi puzzle can be solved with a minimum of 2^n-1 moves, where n is the number of disks.

Q2. What are the various ways is stack created?

A stack can be created in a number of ways, including:

- Using an array
- Using a linked list
- Using a heap

An array-based stack is the simplest type of stack to implement. It consists of an array of elements, and the top of the stack is always the first element in the array.

A linked list-based stack is more complex than an array-based stack, but it is also more efficient. A linked list-based stack consists of a linked list of nodes, and the top of the stack is always the last node in the linked list.

A heap-based stack is the most efficient type of stack to implement, but it is also the most complex. A heap-based stack is a data structure that is used to store elements in a priority queue.

Q3. List the models of computation of language.

There are three main models of computation of language:

- The Turing machine model
- The finite state machine model
- The pushdown automaton model

The Turing machine model is the most general model of computation. It can be used to simulate any other model of computation.

The finite state machine model is a simpler model of computation than the Turing machine model. It can be used to simulate many real-world systems, such as vending machines and traffic lights.

The pushdown automaton model is a model of computation that can be used to simulate languages that have recursion.

Q4. What are objectives of principle of programming language?

The objectives of the principles of programming languages are to:

- Provide a foundation for the study of programming languages
- Develop a common vocabulary for discussing programming languages
- Identify the key concepts of programming languages
- Provide a framework for comparing programming languages

Q5. What are the Paradigms of Programming?

There are many different programming paradigms, but some of the most common include:

- Imperative programming
- Declarative programming
- Functional programming
- Object-oriented programming

Imperative programming is a programming paradigm that uses statements that change the state of the program.

Declarative programming is a programming paradigm that uses statements that describe the desired outcome of the program.

Functional programming is a programming paradigm that uses functions to manipulate data.

Object-oriented programming is a programming paradigm that uses objects to represent data and behavior.

Each paradigm has its own strengths and weaknesses, and the best paradigm for a particular project will depend on the specific requirements of the project.

Viva Questions:

Q1. List various type of languages.

A1. There are many different types of languages, but some of the most common include:

- Natural languages: These are the languages that humans use to communicate with each other. Natural languages are typically very complex and have a rich vocabulary.
- Programming languages: These are the languages that humans use to write computer programs. Programming languages are typically much simpler than natural languages and have a very limited vocabulary.
- Mathematical languages: These are the languages that humans use to express mathematical concepts. Mathematical languages are typically very precise and have a very limited vocabulary.
- Sign languages: These are the languages that people who are deaf or hard of hearing use to communicate with each other. Sign languages are typically very expressive and can be used to communicate a wide range of emotions.

Q2. What are the issues for languages?

A2. Some of the issues for languages include:

- Language barriers: Language barriers can make it difficult for people to communicate with each other. This can be a problem in both personal and professional settings.
- Language diversity: There are many different languages spoken in the world, and this can make it difficult to translate content from one language to another.
- Language change: Languages are constantly changing, and this can make it difficult to keep up with the latest changes.
- Language politics: Language can be a sensitive topic, and this can make it difficult to translate content that is politically sensitive.

Q3. What is translation?

A3. Translation is the process of converting text from one language to another. Translation can be used for a variety of purposes, including:

- Communication: Translation can be used to help people who speak different languages communicate with each other.

- Education: Translation can be used to make educational materials available to people who speak different languages.
- Business: Translation can be used to make business documents and marketing materials available to people who speak different languages.
- Government: Translation can be used to make government documents and policies available to people who speak different languages.

Q4. What are different types of translation and their roles?

A4. There are many different types of translation, including:

- Literary translation: Literary translation is the translation of novels, poems, and other works of literature. Literary translators must have a deep understanding of the source language and the target language, as well as a strong understanding of literature and the art of translation.
- Technical translation: Technical translation is the translation of technical documents, such as manuals, specifications, and software. Technical translators must have a deep understanding of the subject matter of the document, as well as a strong understanding of both the source language and the target language.
- Legal translation: Legal translation is the translation of legal documents, such as contracts, wills, and court rulings. Legal translators must have a deep understanding of the law, as well as a strong understanding of both the source language and the target language.
- Medical translation: Medical translation is the translation of medical documents, such as patient records, prescriptions, and medical reports. Medical translators must have a deep understanding of medicine, as well as a strong understanding of both the source language and the target language.

Q5. What is trade's off of translation?

- Accuracy: Translations must be accurate in order to convey the meaning of the original text. However, accuracy can sometimes come at the expense of fluency.
- Fluency: Translations should be fluent in order to read and understand easily. However, fluency can sometimes come at the expense of accuracy.
- Cost: Translations can be expensive, especially if they are complex or require specialized knowledge.
- Time: Translations can take time, especially if they are complex or require specialized knowledge.

Viva Questions:

Q1. Write any four important uses of programming languages.

A1. The four most important uses of programming languages are:

- Software development: Programming languages are used to develop software applications.
- Systems programming: Programming languages are used to develop operating systems, compilers, and other system software.
- Data science: Programming languages are used to analyze data and build machine learning models.
- Web development: Programming languages are used to develop websites and web applications.

Q2. The levels of acceptance of any language depend on the language description. Commenton this.

A2. The level of acceptance of any language depends on a number of factors, including the language description. The language description is the document that defines the syntax and semantics of the language. A good language description will be clear, concise, and easy to understand. It will also be complete, covering all aspects of the language. A good language description will help to ensure that the language is well-understood and accepted by developers.

Q3. Write the differences between lexical syntax and concrete syntax of the language.

A3. Lexical syntax is the part of the language grammar that deals with the identification of tokens. Tokens are the basic building blocks of a program, such as identifiers, keywords, operators, and literals. Concrete syntax is the part of the language grammar that deals with the layout of the program. Concrete syntax includes things like indentation, line breaks, and comments.

The main difference between lexical syntax and concrete syntax is that lexical syntax is concerned with the identification of tokens, while concrete syntax is concerned with the layout of the program.

Q4. List the design principle of imperative languages.

A4. The design principles of imperative languages are:

- Sequential execution: Imperative languages execute instructions in sequence.
- Data abstraction: Imperative languages allow programmers to abstract away the details of data representation.
- Control flow: Imperative languages allow programmers to control the flow of execution of a program.
- Recursion: Imperative languages allow programmers to define functions that call themselves.
- Error handling: Imperative languages provide mechanisms for handling errors.

Q5. Write the differences between array and enumerated data types inimperative languages?

A5. The main difference between arrays and enumerated data types is that arrays are a data structure that can store a collection of elements of the same type, while enumerated data types are a data type that can store a collection of named constants.

Arrays are typically used to store collections of data that are related in some way. For example, an array could be used to store the names of all the students in a class. Enumerated data types are typically used to store collections of data that are not related in any way. For example, an enumerated data type could be used to store the days of the week.

Q6. Distinguish between dangling pointers and memory leakage.

A6. A dangling pointer is a pointer that points to an object that has been deallocated. A memory leak is a situation where memory that has been allocated is not deallocated.

Dangling pointers can cause errors because they can be used to access memory that has been deallocated. This can lead to data corruption or other problems. Memory leaks can cause programs to use more memory than they need. This can lead to performance problems or even crashes.

It is important to avoid both dangling pointers and memory leaks. There are a number of techniques that can be used to avoid these problems. For example, it is important to always deallocate memory that has been allocated. It is also important to use smart pointers, which are automatically deallocated when they are no longer needed.

Viva Questions:

Q1. List the benefits of modular development approach.

A1. The benefits of modular development approach are:

- Increased modularity: Modularization breaks down a large program into smaller, more manageable modules. This makes the program easier to understand, maintain, and test.
- Improved reusability: Modules can be reused in other programs, which can save time and effort.
- Enhanced documentation: Modularization makes it easier to document the program, which can help to improve its understandability and maintainability.
- Reduced complexity: Modularization can help to reduce the complexity of a program, which can make it easier to understand and maintain.

Q2. Give some reasons why computer scientists and professional software developers should study general concepts of language design and evaluation.

A2. Computer scientists and professional software developers should study general concepts of language design and evaluation because:

- It can help them to understand the principles of programming languages and how they work.
- It can help them to design better programming languages.
- It can help them to evaluate the strengths and weaknesses of existing programming languages.
- It can help them to choose the right programming language for a particular task.

Q3. What constitutes a programming environment?

A3. A programming environment is a collection of tools and resources that a programmer needs to write and debug programs. A programming environment typically includes a text editor, a compiler or interpreter, a debugger, and a variety of other tools.

Q4. Give an example of how aliasing deters reliability.

A4. Aliasing is a situation where two different names refer to the same object. Aliasing can deter reliability because it can make it difficult to track the state of an object. For example, if two

different functions both have a reference to the same object, and one of the functions changes the state of the object, the other function may not be aware of the change. This can lead to errors.

Q5. How do type declaration statements effect the readability of programming language.

A5. Type declaration statements can improve the readability of programming language by making it clear what type of data is being used. This can help to prevent errors and make programs easier to understand. For example, if a variable is declared to be of type int, the programmer knows that the variable can only store integers. This can help to prevent errors such as trying to store a string in an int variable.

In addition, type declaration statements can make programs more readable by providing documentation about the program's data types. This can be helpful for other programmers who need to read and understand the program.

Viva Questions:

Q1. Write the uses of constructor and destructors in OOP.

A1. Constructors and destructors are special functions in object-oriented programming (OOP) that are used to initialize and destroy objects.

- Constructors are called when an object is created. They are used to initialize the object's data members.
- Destructors are called when an object is destroyed. They are used to clean up the object's resources.

Constructors and destructors are important for ensuring that objects are properly initialized and destroyed. They can also be used to perform other tasks, such as logging or error handling.

Q2. Describe any one method for bridging the gap between high-level languageand machine language.

A2. There are a number of methods for bridging the gap between high-level languages and machine language. One common method is to use a compiler. A compiler is a program that translates high-level language code into machine language code.

When a program is compiled, the compiler reads the high-level language code and generates machine language code. The machine language code is then stored in a file or executed directly by the computer.

Another common method for bridging the gap between high-level languages and machine language is to use an interpreter. An interpreter is a program that reads high-level language code and executes it one line at a time.

When a program is interpreted, the interpreter reads the high-level language code and executes it line by line. The interpreter then moves on to the next line of code and repeats the process.

Q3. Explain language evaluation criteria and the characteristics that affect them.

A3. There are a number of criteria that are used to evaluate programming languages. Some of the most important criteria include:

- Expressiveness: The ability of the language to express the desired program.
- Power: The ability of the language to solve a wide range of problems.
- Readability: The ease with which the language can be read and understood by humans.
- Writeability: The ease with which the language can be used to write programs.

- Portability: The ability of the language to be used on a variety of platforms.
- Efficiency: The ability of the language to generate efficient code.

The characteristics that affect these criteria include:

- The syntax of the language: The syntax of the language is the set of rules that govern how the language is written. A good syntax will be easy to learn and remember.
- The semantics of the language: The semantics of the language is the set of rules that govern how the language is interpreted. A good semantics will be clear and unambiguous.
- The libraries and tools that are available for the language: A good language will have a wide range of libraries and tools that are available to help programmers write programs.
- The community of users and developers for the language: A good language will have a large and active community of users and developers who are willing to help each other.

Q4. What Is Backus-naur Form (bnf)?

A4. Backus-Naur Form (BNF) is a metasyntax used to describe the syntax of programming languages. BNF is a formal grammar that uses productions to describe the legal constructs of a language.

A production is a rule that describes how one construct can be replaced with another. For example, the following production describes how a variable declaration can be replaced with a type and a name:

Code

variable_declaration ::= type identifier

This production says that a variable declaration can be replaced with a type and a name. The type must be a valid type, and the name must be a valid identifier.

BNF is a powerful tool that can be used to describe the syntax of any programming language. It is used by compiler writers, language designers, and other people who need to understand the syntax of a programming language.

Viva Questions:

Q1. What is printed by the print statements in the program P1 assuming call by reference parameter passing?

```
C++

ProgramP1()
{
x=10;
y=3;
func1(y, x, x);
print x;
print y;
}
func1 (x, y, z)
{
y = y + 4;
z = x + y + z;
}
```

A1. The output of the program will be:

```
Code snippet

14
7
```

This is because in call by reference parameter passing, the value of the variables is passed by reference. This means that any changes made to the variables in the function will be reflected in the calling function.

In this case, the value of y is passed by reference to the function func1. When y is incremented by 4 in func1, the value of y in the calling function is also incremented by 4.

The value of x is also passed by reference to the function func1. When x is added to y and z in func1, the value of x in the calling function is also updated.

Therefore, after the function func1 is called, the value of x will be 14 and the value of y will be 7.

Q2. The most appropriate matching for the following pairs

```
Code snippet

X: Indirect addressing 1: Loops
Y: Immediate addressing 2: Pointers
Z: Auto decrement addressing 3. Constants
```

A2. The most appropriate matching for the following pairs is:

```
Code snippet

X: Indirect addressing 2: Pointers
Y: Immediate addressing 3. Constants
Z: Auto decrement addressing 1: Loops
```

Indirect addressing is a method of accessing memory by using a pointer. The pointer contains the address of the memory location to be accessed.

Immediate addressing is a method of accessing memory by using a constant. The constant is the value to be stored in the memory location.

Auto decrement addressing is a method of accessing memory by decrementing a register after each access. The register is used to keep track of the current memory location.

Q3. How is memory allocated dynamically?

A3. Memory can be allocated dynamically using the malloc() function. The malloc() function takes a size argument, which specifies the amount of memory to be allocated. The malloc() function returns a pointer to the allocated memory.

The following code shows how to allocate memory dynamically:

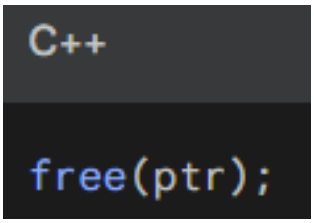
```
C++

int *ptr = (int *)malloc(sizeof(int));
```

In this code, the malloc() function is used to allocate 4 bytes of memory. The pointer ptr is then assigned to the address of the allocated memory.

The free() function can be used to deallocate memory that was allocated using the malloc() function. The free() function takes a pointer to the memory to be deallocated.

The following code shows how to deallocate memory that was allocated using the malloc() function:



Q4. What is the use of pointers?

A4. Pointers are used to access memory indirectly. This means that instead of referring to a memory location by its name, you can refer to it by its address.

Pointers are useful for a variety of tasks, including:

- Accessing arrays and strings
- Passing parameters to functions
- Returning values from functions
- Dynamically allocating memory

Pointers can be a powerful tool, but they can also be dangerous if not used correctly. It is important to understand how pointers work and how to use them safely.

Viva Questions:

Q1. List the various types of pointers.

A1. There are various types of pointers, including:

- Integer pointer: A pointer that points to an integer.
- Float pointer: A pointer that points to a floating-point number.
- String pointer: A pointer that points to a string.
- Object pointer: A pointer that points to an object.

Pointers can also be classified as:

- Null pointer: A pointer that does not point to any valid memory location.
- Wild pointer: A pointer that points to an invalid memory location.
- Dangling pointer: A pointer that points to a memory location that has been deallocated.

Q2. What is the use of functions? How are actual parameters different fromformal parameters?

A2. Functions are used to encapsulate code into reusable units. This makes the code easier to read, understand, and maintain. Functions can also be used to improve performance by avoiding code duplication.

Actual parameters are the values that are passed to a function when it is called. Formal parameters are the variables that are declared in the function definition. The actual parameters are used to initialize the formal parameters.

The main difference between actual parameters and formal parameters is that the actual parameters are evaluated when the function is called, while the formal parameters are not evaluated until the function is executed.

Q3. Differentiate between call by value and call by reference?

A3. Call by value and call by reference are two different ways of passing parameters to functions.

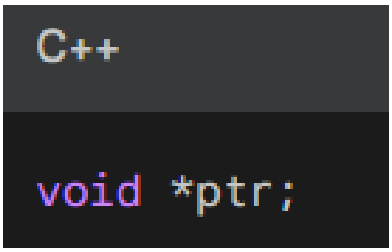
In call by value, the value of the actual parameter is copied to the formal parameter. Any changes made to the formal parameter do not affect the actual parameter.

In call by reference, the address of the actual parameter is passed to the function. Any changes made to the formal parameter are reflected in the actual parameter.

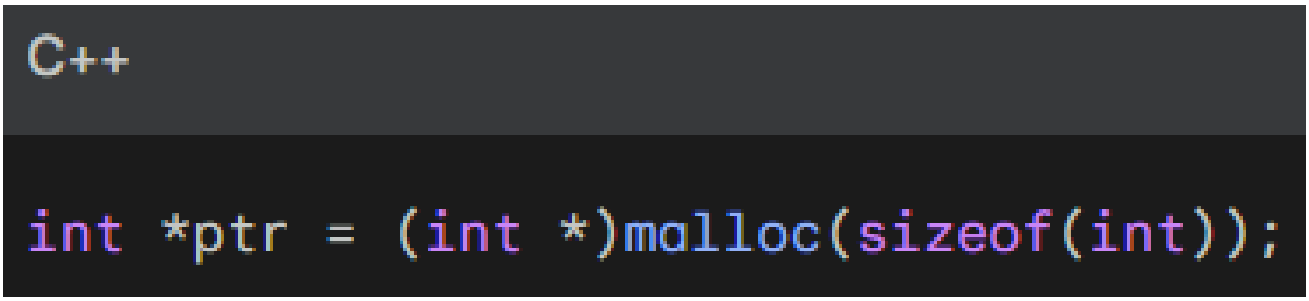
Q4. What is Void pointer?

A4. A void pointer is a pointer that can point to any type of data. Void pointers are typically used to pass pointers to functions that do not know the type of data that they will be working with.

Void pointers are declared using the void * type. For example:



The ptr variable can be used to point to any type of data. For example:



In this code, the ptr variable is a void pointer. It is then cast to an int * pointer. The malloc() function is used to allocate 4 bytes of memory. The ptr variable is then assigned to the address of the allocated memory.

Void pointers can be a dangerous tool if not used correctly. It is important to understand how void pointers work and how to use them safely.

Viva Questions:

Q1. What is the role of producer and consumer in the producer consumer problem?

In the producer-consumer problem, the producer is responsible for creating items and placing them in a buffer. The consumer is responsible for removing items from the buffer and consuming them. The two processes must coordinate their access to the buffer to avoid data corruption.

Q2. What is semaphore?

A semaphore is a synchronization primitive that can be used to control access to a shared resource. Semaphores are typically implemented using counting variables. A semaphore can be in one of two states: signaled or unsignaled. When a semaphore is signaled, it indicates that the resource is available. When a semaphore is unsignaled, it indicates that the resource is not available.

Q3. Explain Deadlock recovery?

Deadlock is a situation in which two or more processes are blocked, each waiting for a resource that is held by the other process. Deadlock recovery is the process of resolving a deadlock situation. There are a number of different deadlock recovery techniques, but the most common technique is to kill one of the deadlocked processes.

Q4. How threads are created?

Threads can be created in a number of different ways, depending on the programming language and operating system. In general, threads are created by calling a function that creates a new thread object. The new thread object is then started by calling its start() method.