Problem Solving in C – Unit Wise Suggestion

Unit – 1 - Introduction to Computers and C

Q1) Explain the basic structure of a C program with an example

Q2) What is Token? What are the different types of token available in C language

Ans - Tokens in C is the most important element to be used in creating a program in C. We can define the token as the smallest individual element in C. For `example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C. Therefore, we can say that tokens in C is the building block or the basic component for creating a program in C language.

Keywords

Keywords are predefined, reserved words in C and each of which is associated with specific features. These words help us to use the functionality of C language. They have special meaning to the compilers.

There are total 32 keywords in C.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

Identifiers

Each program element in C programming is known as an identifier. They are used for naming of variables, functions, array etc. These are user-defined names which consist of alphabets, number, underscore '_'. Identifier's name should not be same or same as keywords. Keywords are not used as identifiers.

Rules for naming C identifiers −

- It must begin with alphabets or underscore.
- Only alphabets, numbers, underscore can be used, no other special characters, punctuations are allowed.
- It must not contain white-space.
- It should not be a keyword.
- It should be up to 31 characters long.

Strings

A string is an array of characters ended with a null character(\0). This null character indicates that string has ended. Strings are always enclosed with double quotes(“ “).

Let us see how to declare String in C language −

- char string[20] = {‘s’,’t’,’u’,’d’,’y’, ‘\0’};
- char string[20] = “demo”;
- char string [] = “demo”;

Here is an example of tokens in C language,

```
#include >stdio.h>
int main() {
  // using keyword char
  char a1 = 'H';
  int b = 8;
  float d = 5.6;
  // declaration of string
  char string[200] = "demodotcom";
  if(b<10)
  printf("Character Value : %c
",a1);
  else
  printf("Float value : %f
",d);
  printf("String Value : %s
", string);
  return 0;
}
```

Output

```
Character Value : H
String Value : demodotcom
```

Q3) What is an identifier (variable)?
Q4) What is variable? List the restrictions on the variable names
Q5) Define variable. Explain the rules for constricting variables in C language
Q6) What are basic data types available in „C‟? Write the significance of each data type

Q7)What is type conversion? Explain two types of conversion with examples

Typecasting is converting one data type into another one. It is also called as data conversion or type conversion in C language. It is one of the important concepts introduced in 'C' programming.

'C' programming provides two types of type casting operations:

1. Implicit type casting
2. Explicit type casting

Implicit type casting

Implicit type casting means conversion of data types without losing its original meaning. This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.

Implicit type conversion in C happens automatically when a value is copied to its compatible data type. During conversion, strict rules for type conversion are applied. If the operands are of two different data types, then an operand having lower data type is automatically converted into a higher data type.

This type of type conversion can be seen in the following example.

```
#include<stdio.h>
int main(){
        short a=10; //initializing variable of short data type
        int b; //declaring int variable
        b=a; //implicit type casting
        printf("%d\n",a);
        printf("%d\n",b);
}
```

Output:

```
10
10
```

Explicit type casting

In implicit type conversion, the data type is converted automatically. There are some scenarios in which we may have to force type conversion. Suppose we have a variable div that stores the division of two operands which are declared as an int data type.

```
int result, var1=10, var2=3;
result=var1/var2;
```

In this case, after the division performed on variables var1 and var2 the result stored in the variable "result" will be in an integer format. Whenever this happens, the value stored in the variable "result" loses its meaning because it does not consider the fraction part which is normally obtained in the division of two numbers.

To force the type conversion in such situations, we use explicit type casting.

It requires a type casting operator. The general syntax for type casting operations is as follows:

(type-name) expression

Here,

- The type name is the standard 'C' language data type.
- An expression can be a constant, a variable or an actual expression.

Let us write a program to demonstrate how to typecast in C with explicit type-casting.

```
#include<stdio.h>
int main()
{
	float a = 1.2;
	//int b  = a; //Compiler will throw an error for this
	int b = (int)a + 1;
	printf("Value of a is %f\n", a);
	printf("Value of b is %d\n",b);
	return 0;
}
```

Output:

```
Value of a is 1.200000
Value of b is 2
```

Q8) What are the formatted input and output functions. Explain with examples

Ans:- Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user. These types of I/O functions can help to display the output to the user in different formats using the format specifiers. These I/O supports all data types like int, float, char, and many more.

The following formatted I/O functions will be discussed in this section-

1. **printf()**
2. **scanf()**
3. **sprintf()**
4. **sscanf()**

1. printf()

The printf() function is the most used function in the C language. This function is defined in the **stdio.h** header file and is used to show output on the console (standard output).

This function is used to **print a simple text sentence** or **value of any variable** which can be of int, char, float, or any other datatype.

```
printf(“Hi”);
```

and the program will print the content of the string to the screen.

You can print the value of a variable, and it’s a bit tricky because you need to add a special character, a placeholder, which changes depending on the type of the variable. **For example we use %d for a signed decimal integer digit:**

```
Int age = 25;

Printf(“Myage is %d”, age);
```

We can print more than one variable by using commas:

```
Int age_yesterday = 36;

Int age_today = 37;

Printf(“ Yesterday my age was %d and today is %d”, age_yesterday, age_today);
```

There are other format specifiers like %d:

- **%c for a char**
- **%s for a string**
- **%f for floating point numbers**
- **%p for pointers**

and many more.

We can use escape characters in printf(), like \n which we can use to make the output create a new line.

2. scanf()

scanf() function is used to read/input values of variables using the standard input device such as keyboard. This function is used to get a value from the user running the program, from the command line.

We must first define a variable that will hold the value we get from the input:

```
Int age;
```

Then we call scanf() with 2 arguments: the format (type) of the variable, and the address of the variable:

```
scanf(“%d”, &age);
```

If we want to get a string as input, remember that a string name is a pointer to the first character, so you don’t need the & character before it:

```
char name[20];
```

```
scanf("%s", name);
```

Here's a little program that uses both printf() and scanf():

```
#include <stdio.h>

int main(void)  {

          char name[20];

          printf("Enter your name: ");

scanf("%s", name);

printf("you entered %s", name);

}
```

3. sprintf()

sprintf stands for **"string print"**. This function is similar to printf() function but this function prints the string into a character array instead of printing it on the console screen.

Syntax:

sprintf(array_name, "format specifier", variable_name);

4. sscanf():

sscanf stands for **"string scanf".** This function is similar to scanf() function but this function reads data from the string or character array instead of the console screen.

Syntax:

sscanf(array_name, "format specifier", &variable_name);

q9) Explain with example, the various constants available in „C" language

Constant is also known as variable where once defined, the value never changes during the program execution. Thus, we can declare a variable as constant that refers to fixed values. It is also called as literals. Const keyword has to be used to define a constant.

Syntax

The syntax for constant that is used in C programming language is given below −

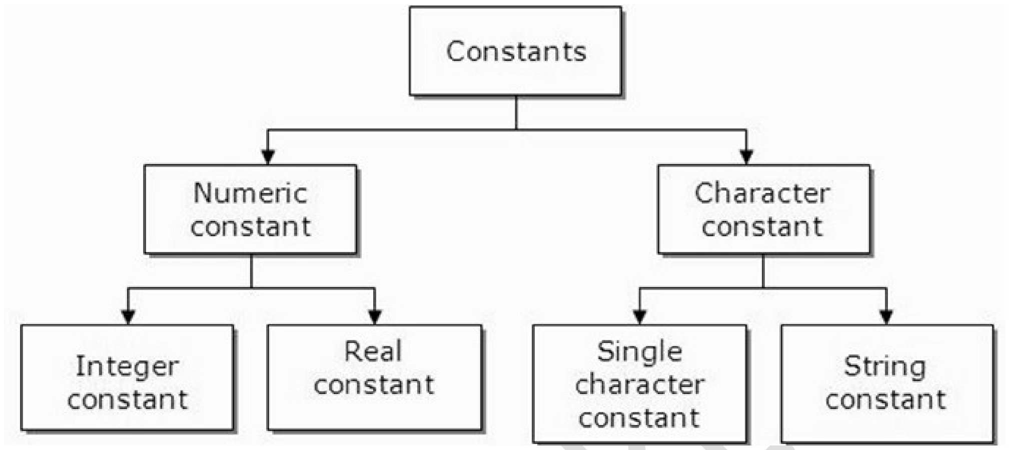
```
const type VariableName;
(or)
const type *VariableName;
```

Different types of constants

The different types of constants that are used in C programming language are as follows −

- **Integer constants** − For example: 1,0,34,4567
- **Floating-point constants** − For example: 0.0, 156.89, 23.456
- **Octal & Hexadecimal constants** − For example: Hexadecimal: 0x2a, 0xaa .. and Octal: 033, 024,..
- **Character constants** − For example: ‘a’, ‘B’, ‘x’
- **String constants** − For example: “TutorialsPoint”

The types of constants are also What ised in the diagram below −



Example 1

Following is the C program for **determining the value of a number** −

```
#include<stdio.h>
int main(){
  const int number=45;
  int value;
  int data;
  printf("enter the data:");
  scanf("%d",&data);
  value=number*data;
  printf("The value is: %d",value);
  return 0;
}
```

Output

When the above program is executed, it produces the following result −

```
enter the data:20
```

```
The value of number is: 900
```

In the above program, if we try to change the value of a number which is declared as constant, it displays an error

Example 2

Given below is the C program which **gives an error, if we try to change the const value**.

```
#include<stdio.h>
int main(){
   const int number=45;
   int data;
   printf("enter the data:");
   scanf("%d",&data);
   number=number*data;
   printf("The value of number is: %d",number);
   return 0;
}
```

Output

When the above program is executed, it produces the following result −

```
error
```

Q10) What is an operator? List and explain various types of operators.

<u>Unit – 2 Branching & Looping</u>

Q1) List all conditional control statements used in C

Q2) Explain switch statement with syntax and example

Q3) Define Loop? Explain different types of loops available in C programming language?

Q4) Difference between while loop and do-while loop and for-loop

Q5) Explain the Control and exit control loops with example

Q6) Show how break and continue statements are used in a C program, with example

Q7) Difference between Break and Continue Statement in C?

Q8) Explain Infinite Loop with example in C

Q9) Working of All types of loops

Q10) Describe the decision-making statement and looping statements in C with an example

Unit 3 –Arrays in C

1. What is array ? Explain the declaration and initialization of one dimensional and two dimensional array with an example
2. Define array.Explain with suitable example how to declare and initialize 1D array
3. Mention some advantages and disadvantages of Arrays.
4. What is the time complexity for performing basic operations in an array?
5. Difference between pointer and array in C?
6. How do you access elements in an array?
7. What is the difference between a one-dimensional and multi-dimensional array?
8. How do you pass an array to a function in C?
9. What is the difference between an array and a pointer in C?
10. How do you sort an array in C?
11. How do you search for a specific element in an array?
12. How do you find the length of an array in C?
13. How do you initialize an array in C?
14. What is a dynamic array and how is it implemented in C?
15. What are the different methods for searching an element in an array?
16. How does linear search algorithm work and what is its time complexity?
17. How does binary search algorithm work and what is its time complexity?
18. What are the advantages and disadvantages of using linear search vs binary search?
19. How can you optimize the binary search algorithm?
20. How does the bubble sort algorithm work and what is its time complexity?
21. How does the selection sort algorithm work and what is its time complexity?
22. How does the insertion sort algorithm work and what is its time complexity?
23. What are the advantages and disadvantages of using different sorting algorithms?
24. How can you sort an array in descending order?
25. How can you sort an array in ascending order?

Unit 4 – Pointers

1. What is pointer? Explain how the pointer variable declared and initialized?
2. Explain the array of pointes with example? or explain how pointers and arrays are related with example
3. What is pointer? give the advantages and disadvantages of pointer data type
4. What is the difference between a pointer and an array in C?
5. How do you pass a pointer to a function in C?
6. What is the difference between a null pointer and a void pointer?

How do you allocate memory dynamically using pointers?

Ans:- There are two main ways to allocate memory dynamically using pointers in C:

malloc(): The malloc() function is used to dynamically allocate a block of memory of a specified size. It takes a single argument, which is the number of bytes of memory to be allocated. The function returns a void pointer to the start of the allocated memory block. The allocated memory is not initialized and its value is indeterminate.

```
int *ptr = (int *) malloc(sizeof(int) * 5);
```

calloc(): The calloc() function is also used to dynamically allocate memory, but it differs from malloc() in that it initializes the allocated memory to zero. It takes two arguments: the number of elements to be allocated and the size of each element. Like malloc(), it returns a void pointer to the start of the allocated memory block.

```
int *ptr = (int *) calloc(5, sizeof(int));
```

It's important to note that when you use malloc or calloc to allocate memory dynamically, it's the programmer's responsibility to free the allocated memory when it's no longer needed by using free() function.

How do you compare two pointers?

In C, you can compare two pointers using the relational operators (>, <, >=, <=, ==, !=). When comparing two pointers, the comparison is based on the memory addresses they point to, not the values they point to.

For example, you can compare two pointers as follows:

```
int a = 5, b = 10;

int *ptr1 = &a, *ptr2 = &b;

if (ptr1 < ptr2) {

   printf("ptr1 points to a lower memory address than ptr2\n");

} else if (ptr1 > ptr2) {

   printf("ptr1 points to a higher memory address than ptr2\n");

} else {

   printf("ptr1 and ptr2 point to the same memory address\n");

}
```

It's important to note that the result of comparing two pointers that don't point to the same array or that have been allocated dynamically using malloc() or calloc() is undefined.

Also, you can't compare pointers that point to different data types.

Also, it's important to note that the result of comparing two pointers that point to different data types is undefined.

Unit-5 Pre-processor

What is preprocessor directive? Explain #define and #include preprocessor directives

Ans: In C, preprocessor directives are lines of code that are executed before the actual compilation of the program. These directives are used to perform tasks such as defining constants, including header files, and controlling conditional compilation. Preprocessor directives begin with the # symbol.

#define: The #define preprocessor directive is used to define constants in C. It can be used to define constants in the form of macro definitions. For example, the following code defines a constant named **PI** with a value of **3.14**:

```
#define PI 3.14
```

#include: The #include preprocessor directive is used to include the contents of one file into another. It's typically used to include header files that contain declarations for functions and variables used in the program. For example, the following code includes the contents of the **stdio.h** header file:

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

You can also include header files in the form of double quotes instead of angle brackets, for example:

```
#include "myheader.h"
```

This is useful when the header files are not in the system include path but in the same directory of the source file.

In both cases, the preprocessor will replace the directive with the content of the corresponding file.

The #define directive can be used for simple replacements like constants, but also for more complex macro-function definition.

Explain any five preprocessor directives in C

#define: The #define preprocessor directive is used to define constants and macro in C. It can be used to define constants in the form of macro definitions. For example, the following code defines a constant named PI with a value of 3.14:

```
#define PI 3.14
```

#include: The #include preprocessor directive is used to include the contents of one file into another. It's typically used to include header files that contain declarations for functions and variables used in the program. For example, the following code includes the contents of the stdio.h header file:

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

#ifdef, #ifndef and #endif : These are used for conditional compilation. The #ifdef directive is used to check whether a specific macro is defined, if it is defined then the code written after this directive will be executed otherwise not. #ifndef is opposite of #ifdef, it check whether a macro is not defined and executes the code if it's not defined.

```
#ifdef DEBUG

   printf("Debug mode is on");

#endif
```

#pragma: The #pragma preprocessor directive is used to enable or disable certain features of the compiler. For example, the following code disables the warning for unused variables:

```
#pragma GCC diagnostic ignored "-Wunused-variable"
```

#error : The #error preprocessor directive is used to stop the compilation process and print a user-defined error message. For example, the following code will stop the compilation process and print the specified error message if the DEBUG macro is not defined.

```
#ifndef DEBUG

   #error "DEBUG macro is not defined"

#endif
```

These are some of the commonly used preprocessor directives in C, but there are others like #undef, #line, #elif, etc. which have their specific uses.

What is a macro ?Write a macro to determine whether the given number is odd or even

A macro in C is a fragment of code that has been given a name. When the name is used, it is replaced by the contents of the macro. Macros are typically used to define constants, but they can also be used to define more complex functionality in the form of macro functions.

Here is an example of a macro that can be used to determine whether a given number is odd or even:

```
#define IS_EVEN(x) ((x % 2) == 0) ? "Even" : "Odd"
```

This macro takes a single argument x and checks if it is divisible by 2 using the modulus operator(%) and returns "Even" if it is and "Odd" if it's not.

You can use this macro in your code like this:

```
int num = 5;

printf("%d is %s", num, IS_EVEN(num));
```

This will print "5 is Odd"

It's important to note that macros are replaced by the preprocessor with their expanded form before the actual compilation of the code, so they are not as efficient as functions, and also they do not have type checking and scope, so they can cause some problems if not used carefully.

Also, you can use ternary operator in the macro which makes it more readable, and the code more concise.

Unit 7 – Functions and Recursion

1. What is function ? Explain the difference between user defined and library functions
2. Explain the different elements of user defined functions in detail
3. Explain function call, function definition and function prototype with examples
4. What is function? Write a function to find the sum of two numbers
5. Differentiate between call by value and call by reference with examples
6. Explain the type of functions based on parameters
7. Define global variable with example
8. Define Static Variable with example
9. Define Local Variable with example
10. wright down the advantage and dis-advantage of global variable
11. Difference between global vs local and static variable
12. what do you mean by scope of the variable in C?Explain different type scope with example
13. what is static function? Explain Function with no arguments and no parameters
14. what is function prototype in C?
15. what is recursion?
16. what is Recursive Function?
17. what are the advanatge and dis-advanatge of Recusrion?

Unit-8 – String

1. what is string in C? Explain with example
2. Define String Literal?
3. Explain Following String Function with Example in C
    1. gets()
    2. puts()
    3. fgets()
    4. strlen()
    5. strcpy()
    6. strcat()
    7. strcmp()
    8. strrev()
4. How do you declare and initialize a string in C?
5. How do you concatenate two strings in C?
6. How do you find the length of a string in C?
7. How do you compare two strings in C?
8. How do you reverse a string in C?
9. What is the difference between character array and string literal?

Unit-9 – Storage Class in C

1)How does the register storage class affect the performance of a program in C?

The register storage class in C is used to indicate that a variable should be stored in a register, rather than in memory. This can potentially improve performance by reducing the number of memory accesses required to access the variable, as registers are generally faster to access than memory. However, it is important to note that the compiler is free to ignore the register storage class, and whether or not a variable is actually stored in a register will depend on the specific implementation and the availability of registers at the time the program is executed. In general, the use of the register storage class should be considered a hint to the compiler rather than a strict instruction. It's also worth noting that not all variables are suitable to be stored in register, for example large arrays or structures may not fit into a register, it's also important to be aware of the number of available registers in the target architecture.

2)How do the storage classes in C (auto, static, register, extern) affect the memory allocation of variables?

The storage classes in C (auto, static, register, extern) affect the memory allocation of variables in the following ways:

1. **Auto:** Variables declared with the auto storage class (also known as automatic or local variables) are stored on the stack. The memory for these variables is allocated when the block in which they are defined is entered and deallocated when the block is exited.
2. **Static:** Variables declared with the static storage class are also stored in memory, but unlike auto variables, the memory for these variables is allocated at program startup and remains allocated until the program exits. The value of a static variable is also initialized only once.
3. **Register:** Variables declared with the register storage class are stored in a register, rather than in memory. However, the compiler is free to ignore this storage class and place the variable in memory instead, depending on the specific implementation and the availability of registers at the time the program is executed.
4. **Extern:** Variables declared with the extern storage class are not allocated memory by the compiler, but are used to provide a reference to a variable that is defined elsewhere in the program.

It's also worth noting that the size of the variables in each storage class is the same, and the difference is in how they are allocated and accessed.

3)Define Storage Class?

4)Difference between auto vs register vs extern vs static storage class in C

Unit-10 – Structure and Union

1. How do you define and declare a structure in C?
2. How do you access the individual members of a structure in C?
3. How can you pass a structure to a function in C?
4. Can you explain the concept of nested structures in C and give an example of when they might be used?
5. Define Union in C?
6. What is the main difference between a structure and a union in C?

Unit – 11 Files in C

Follow My Notes