Important questions of CPA

Very Short type questions

**Ques1**. What is the difference between Call by Value and Call by Reference?

**Sol.** When using Call by Value, you are sending the value of a variable as parameter to a function, whereas Call by Reference sends the address of the variable. Also, under Call by Value, the value in the parameter is not affected by whatever operation that takes place, while in the case of Call by Reference, values can be affected by the process within the function.

**Ques2**. **What is variable initialization and why is it important?**

**Sol.** This refers to the process wherein a variable is assigned an initial value before it is used in the program. Without initialization, a variable would have an unknown value, which can lead to unpredictable outputs when used in computations or other operations.

**Ques3**. **What is the modulus operator?**

**Sol. The modulus operator outputs the remainder of a division. It makes use of the percentage (%) symbol. For example: 10 % 3 = 1, meaning when you divide 10 by 3, the remainder is 1.**

**Ques4**. **What is the difference between the = symbol and == symbol?**

**Sol. The = symbol is often used in mathematical operations. It is used to assign a value to a given variable. On the other hand, the == symbol, also known as “equal to” or “equivalent to”, is a relational operator that is used to compare two values.**

**Ques5. What are preprocessor directives?**

**Sol. Preprocessor directives are placed at the beginning of every C program. This is where library files are specified, which would depend on what functions are to be used in the program. Another use of preprocessor directives is the declaration of constants. Preprocessor directives begin with the # symbol.**

**Ques6.  What are actual arguments?**

**Sol. When you create and use functions that need to perform an action on some given values, you need to pass these given values to that function. The values that are being passed into the called function are referred to as actual arguments.**

**Ques7. Define algorithm.**

**Sol. a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.**

**Ques8. Define assembler.**

**Sol. a program for converting instructions written in low-level symbolic code into machine code.**

**Ques9. What are pointers?**

**Sol. Pointers point to specific areas in the memory. Pointers contain the address of a variable, which in turn may contain a value or even an address to another memory.**

**Ques10. Define program**

**Sol. A computer program is a set of instructions for a computer to perform a specific task.**

**Short type questions**

**Ques1. Compare and contrast compilers from interpreters.**

**Sol. Compilers and interpreters often deal with how program codes are executed. Interpreters execute program codes one line at a time, while compilers take the program as a whole and convert it into object code, before executing it. The key difference here is that in the case of interpreters, a program may encounter syntax errors in the middle of execution, and will stop from there. On the other hand, compilers check the syntax of the entire program and will only proceed to execution when no syntax errors are found.**

**Ques2. What are header files and what are its uses in C programming?**

**Sol. Header files are also known as library files. They contain two essential things: the definitions and prototypes of functions being used in a program. Simply put, commands that you use in C programming are actually functions that are defined from within each header files. Each header file contains a set of functions. For example: stdio.h is a header file that contains definition and prototypes of commands like printf and scanf.**

**Ques3. What are variables and it what way is it different from constants?**

**Sol. Variables and constants may at first look similar in a sense that both are identifiers made up of one character or more characters (letters, numbers and a few allowable symbols). Both will also hold a particular value.  Values held by a variable can be altered throughout the program, and can be used in most operations and computations. Constants are given values at one time only, placed at the beginning of a program. This value is not altered in the program. For example, you can assigned a constant named PI and give it a value 3.1415.  You can then use it as PI in the program, instead of having to write 3.1415 each time you need it.**

**Ques4. What is a program flowchart and how does it help in writing a program?**

**Sol. A flowchart provides a visual representation of the step by step procedure towards solving a given problem. Flowcharts are made of symbols, with each symbol in the form of different shapes. Each shape may represent a particular entity within the entire program structure, such as a process, a condition, or even an input/output phase.**

**Ques5. What are multidimensional arrays?**

**Sol. Multidimensional arrays are capable of storing data in a two or more dimensional structure. For example, you can use a 2 dimensional array to store the current position of pieces in a chess game, or position of players in a tic-tac-toe program.**

**Ques6. What are Programming Languages?**

 **Sol. Programming languages are classified as:**

**1 Machine language:**

**2 Assembly language:**

**3 High level language :**

**MACHINE LANGUAGE :- the language of 0s and 1s is called as machine language. The machine language is system independent because there are different set of binary instruction for different types of computer systems.**

**LIMITATIONS OF MACHINE LANGUAGES :**

**It is very tedious and error prone process of writing programs in machine languages.**

**ASSEMBLY LANGUAGES: it is low level programming language in which the sequence of 0s and 1s are replaced by mnemonic (ni-monic) codes. Typical instruction for addition and subtraction.**

**Example: - ADD for addition, SUB for subtraction etc**

**HIGH LEVEL LANGUAGE: - high level languages are English like statements and programs. Written in these languages are needed to be translated into machine language before to their execution using a system software compiler.**

**Ques7. Difference between structure and union.**

**Sol.**

|  |  |
| --- | --- |
| **Structure** | **Union** |
| **1.The keyword  struct is used to define a structure** | **1. The keyword union is used to define a union.** |
| **2. When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is greater than or equal to the sum of  sizes of its members. The smaller members may end with unused slack bytes.** | **2. When a variable is associated with a union, the  compiler allocates the  memory by considering the size of the largest memory. So, size of union is equal to the size of largest member.** |
| **3. Each member within a structure is assigned unique storage area of location.** | **3. Memory allocated is shared by individual members of union.** |
| **4. The address of each member will be in ascending order This indicates that memory for each member will start at different offset values.** | **4. The address is same for all the members of a union. This indicates that every member begins at the same offset value.** |
| **5 Altering the value of a member will not affect other members of the structure.** | **5. Altering the value of any of the member will alter other member values.** |
| **6. Individual member can be accessed at a time** | **6. Only one member can be accessed at a time.** |
| **7. Several members of a structure can initialize at once.** | **7. Only the first member of a union can be initialized.** |

**Long Answer type Questions**

**Ques1. Explain data types in C**

**Sol. Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.**

**The types in C can be classified as follows −**

|  |  |
| --- | --- |
| **S.N.** | **Types & Description** |
| **1** | **Basic Types****They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.** |
| **2** | **Enumerated types****They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.** |
| **3** | **The type void****The type specifier void indicates that no value is available.** |
| **4** | **Derived types****They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.** |

**The array types and structure types are referred collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see the basic types in the following section, where as other types will be covered in the upcoming chapters.**

**Integer Types**

**The following table provides the details of standard integer types with their storage sizes and value ranges −**

|  |  |  |
| --- | --- | --- |
| **Type** | **Storage size** | **Value range** |
| **char** | **1 byte** | **-128 to 127 or 0 to 255** |
| **unsigned char** | **1 byte** | **0 to 255** |
| **signed char** | **1 byte** | **-128 to 127** |
| **int** | **2 or 4 bytes** | **-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647** |
| **unsigned int** | **2 or 4 bytes** | **0 to 65,535 or 0 to 4,294,967,295** |
| **short** | **2 bytes** | **-32,768 to 32,767** |
| **unsigned short** | **2 bytes** | **0 to 65,535** |
| **long** | **4 bytes** | **-2,147,483,648 to 2,147,483,647** |
| **unsigned long** | **4 bytes** | **0 to 4,294,967,295** |

**To get the exact size of a type or a variable on a particular platform, you can use the sizeof operator. The expressions sizeof(type) yields the storage size of the object or type in bytes.**

**Floating-Point Types**

**The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision −**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Storage size** | **Value range** | **Precision** |
| **float** | **4 byte** | **1.2E-38 to 3.4E+38** | **6 decimal places** |
| **double** | **8 byte** | **2.3E-308 to 1.7E+308** | **15 decimal places** |
| **long double** | **10 byte** | **3.4E-4932 to 1.1E+4932** | **19 decimal places** |

**he void Type**

**The void type specifies that no value is available. It is used in three kinds of situations −**

|  |  |
| --- | --- |
| **S.N.** | **Types & Description** |
| **1** | **Function returns as void****There are various functions in C which do not return any value or you can say they return void. A function with no return value has the return type as void. For example, void exit (int status);** |
| **2** | **Function arguments as void****There are various functions in C which do not accept any parameter. A function with no parameter can accept a void. For example, int rand(void);** |
| **3** | **Pointers to void****A pointer of type void \* represents the address of an object, but not its type. For example, a memory allocation function void \*malloc( size\_t size ); returns a pointer to void which can be casted to any data type.** |

**Ques 2. What is arrays? Explain different types of arrays.**

**Sol. In C language, arrays are reffered to as structured data types. An array is defined as finite ordered collection of homogenous data, stored in contiguous memory locations.**

**Here the words,**

**finite means data range must be defined.**

**ordered means data must be stored in continuous memory addresses.**

**homogenous means data must be of similar data type.**

**Since arrays provide an easy way to represent data, it is classified amongst the data structures in C. Other data structures in c are structure, lists, queues and trees. Array can be used to represent not only simple list of data but also table of data in two or three dimensions.**

**Declaring an Array**

**Like any other variable, arrays must be declared before they are used. General form of array declaration is,**

**data-type variable-name[size];**

**Ques3. Explain different types of operators in C**

**Sol. C language supports a rich set of built-in operators. An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations. Operators are used in program to manipulate data and variables.**

**C operators can be classified into following types,**

**Arithmetic operators**

**Relation operators**

**Logical operators**

**Bitwise operators**

**Assignment operators**

**Conditional operators**

**Special operators**

**Ques4. Write a program to swap two numbers.**

**Sol. #include<stdio.h>**

**#include<conio.h>**

**void main()**

**{**

 **int x=10, y=15, temp;**

 **temp = x;**

 **x = y;**

 **y = temp;**

 **printf("x= %d and y= %d", x, y);**

 **getch();**

**}**

**Ques5. Explain storage classes in detail.**

**Sol. In C language, each variable has a storage class which decides scope, visibility and lifetime of that variable. The following storage classes are most often used in C programming**

* **Automatic variables**
* **External variables**
* **Static variables**
* **Register variables**
1. **Automatic variables**

 **A variable declared inside a function without any storage class specification, is by default an automatic variable. They are created when a function is called and are destroyed automatically when the function exits. Automatic variables can also be called local variables because they are local to a function. By default they are assigned garbage value by the compiler.**

1. **External or Global variable**

**A variable that is declared outside any function is a Global variable. Global variables remain available throughout the entire program. One important thing to remember about global variable is that their values can be changed by any function in the program.**

1. **Static variables**

**A static variable tells the compiler to persist the variable until the end of program. Instead of creating and destroying a variable every time when it comes into and goes out of scope, static is initialized only once and remains into existence till the end of program. A static variable can either be internal or external depending upon the place of declaration. Scope of internal static variable remains inside the function in which it is defined. External static variables remain restricted to scope of file in each they are declared.**

1. **Register variable**

**Register variable inform the compiler to store the variable in register instead of memory. Register variable has faster access than normal variable. Frequently used variables are kept in register. Only few variables can be placed inside register.**