Q1. When should a type cast not be used?
Ans. A type cast can be used to change the data from one type to another type. For example, it can be used to change the data from double data type to integer data type. But it cannot be used to convert integer data type to double data type. In other words, we can say that type cast can be used to convert larger data types to smaller data types. But note it that it cannot convert smaller data types to larger data types. Doing this will give an error.
Also, type cast cannot be used to override constant. Overriding constants can cause runtime errors
For eg. #include
#include
main()
{
double a=6.567;
clrscr();
int b=(double)a;
printf("%d",a);
printf("%d",b);
getch();
}
The output will be: 6.567
6
Q2. Can math operations be performed on a void pointer?
Ans. No math operations cannot be performed on a void pointer. This is because addition and subtraction of pointers is based on advancing the pointer by number of elements. Now if we have a void pointer, we don't know what is it pointing to and so we don't know the size of the location it is pointing to. Hence, we can't apply math operations on a void pointer.
We can use character pointers, if we want pointer arithmetic to work on raw addresses.
NOTE:
We can cast void to a char, do arithmetic and cast it back to a void.
Q3. Why n++ executes faster than n+1?
Ans. This is because the expression n++ requires single machine instruction such as INR to carry out increment operation whereas n+1 requires multiple instructions to carry out increment operation. So, the expression n++ gets executed faster than the expression n+1.
Q4. What is the use of bit wise operators?
Ans. Operating systems requires the manipulation of data at addresses, and this requires manipulating individual bits or groups of bits. For this purpose bit wise operators are used. These operators are found in C, C++ and Java. Bitwise operators allow us to read and manipulate bits in variables of certain types.
NOTE: Bit wise operators only work on limited number of types. These are int and char.
Q5. What is a modulus Operators in C?
Ans. The modulus(%) operator computes the remainder that results from performing integer division. The modulus operator is useful in variety of circumstances. It is commonly used to take a random number and reduce that number to a random number on a smaller range. And it can also tell if any number is a factor of another.
We can use modulus operator to find if any number is even or odd.
#include
#include
main()
{
int x;
for(x=0;x<10;x++)
if (x % 2 == 0)
{
printf("%d is even",x);
}
return 0;
}
Q6. What are the restrictions of a modulus operator in C?
Ans. There is a restriction in C language while using the modulus operator. There are three types of numerical data types namely integer, float and double. But modulus operator can operate only on integers and cannot operate on floats or double. If anyone tries to use the modulus operator on floats then the compiler would display error message as ‘Illegal use of Floating Point’.
For example: main ()
{
float x=3,y=2;
int z;
z = x % y;
printf("%d”, z);
}
The output will be an error message: Illegal use of Floating Point
Q7. How can we find size of a variable without using sizeof() operator?
Ans. We can use the following macro, given below:
#define sizeof_var(var) ((size_t)(&(var)+1)-(size_t)(&(var)))
The idea is to use pointer arithmetic ((&(var)+1)) to determine the offset of the variable, and then subtract the original address of the variable, yielding its size. For example, if you have an int16_t i variable located at 0x0002, you would be subtracting 0x0002 from 0x0006, thereby obtaining 0x4 or 4 bytes.
Q8. How can we add two numbers in C language without using Arithmetic operators?
Ans. There are many methods to add two numbers without using Arithmetic operators. One such method is given below:
PROGRAM
#include
#include
int add(int,int);
void main()
{
int a,b;
clrscr();
printf("Enter the two Numbers: ");
scanf("%d%d",&a,&b);
printf("Addition of two num. is : %d",add(a,b));
getch();
}
int add(int num1,int num2)

```
{
int i;
for(i=0;i<num2;i++)
num1++;
return num1;
}
```

Q9. How to swap two numbers using bit wise operators?

Ans.PROGRAM

```
#include
int main() {
int i = 65;
int k = 120;
printf("n value of i=%d k=%d before swapping", i, k);
i = i ^ k;
k = i ^ k;
i = i ^ k;
printf("n value of i=%d k=%d after swapping", i, k);
return 0;
}
```

Explanation:

i = 65; binary equivalent of 65 is 0100 0001

k = 120; binary equivalent of 120 is 0111 1000

i = i^k;

i...0100 0001

k...0111 1000

---

val of i = 0011 1001

---

k = i^k

i...0011 1001

k...0111 1000

---

val of k = 0100 0001 binary equivalent of this is 65

---(that is the initial value of i)

i = i^k

i...0011 1001

k...0100 0001

---

val of i = 0111 1000 binary equivalent of this is 120

---(that is the initial value of k)

Q10. Which bit wise operator is suitable for checking whether a particular bit is ON or OFF?

Ans. Bit wise AND operator is suitable for checking if a particular bit is ON or OFF. AND operator works as shown below:

ANDing operation :

10101101 original bit pattern

00001000 AND mask

00001000 resulting bit pattern

The resulting value we get is 8. We get 8 because the 4th bit of the operand was ON. If the 4th bit was OFF, the resulting bit pattern would have been 00000000. "0" represents OFF state and "1" represents ON state