|  |
| --- |
| **Operators** |
| 1.Describe the purpose and usage of the ternary conditional operator(?:) in c programming. Provide an example demonstrating its usage. |
| A:The conditional operator in C is kind of similar to the if-else statement as it follows the same algorithm as of if-else statement but the conditional operator takes less space and helps to write the if-else statements in the shortest way possible. It is also known as the ternary operator in C as it operates on three operands.  variable = (condition) ? Expression2 : Expression3;  int main()  {  int x = 10;  int y = 20;  int max;  max = (x > y) ? x : y;  printf("The maximum value is: %d\n", max);      return 0;  } |
| 2. Discuss the bitwise operators available in C programming. Explain their usage with suitable examples. |
| A: Bitwise operators in C programming are tools for performing bit-level operations on integer data. These bit operators, including AND (&), OR (|), XOR (^), and NOT (~), allow programmers to manipulate individual bits within binary sequences/ representations.  AND Operator (&): The bitwise AND operator performs a logical AND operation on each pair of corresponding bits. It returns 1 if both bit values are 1; otherwise, it returns 0.  operand1 & operand2  OR Operator (|): The bitwise OR operator performs a logical OR operation on each pair of corresponding bits. It returns 1 if at least one of the bits is 1; otherwise, it returns 0.  operand1 | operand2  XOR Operator (^): The bitwise XOR (exclusive OR) operator returns 1 if the bits being compared are different; otherwise, it returns 0.  operand1 ^ operand2  Complement Operator (~): The bitwise complement operator inverts the bit values. That is, this bit operation turns 0s into 1s and 1s into 0s and returns the complement representation of the number.  ~operand  Left Shift Operator (<<): The binary left shift operator moves the bits of a number to the left by a specified number of bit positions. It effectively multiplies the number by 2 to the power of the shift count.  operand1 << operand2  Right Shift Operator (>>): The right shift operator moves the bits of a number to the right by a specified number of positions. It effectively divides the number by 2 to the power of the shift count.  operand1>>operand2 |
| 3. Explain the difference between the postfix and prefix increment operators (++) in C programming. Provide examples to illustrate. |
| A: Prefix Increment Operator (++variable): This operator increments the value of the variable and then returns the updated value.  Postfix Increment Operator (variable++): This operator returns the current value of the variable and then increments it.  Prefix Increment Operator (++variable):  #include <stdio.h>  int main()  {  int num = 5;  int result = ++num;  printf("Result: %d\n", result);  printf("Updated num: %d\n", num);  return 0;  }  Result=6, num=6  Postfix Increment Operator (variable++):  #include <stdio.h>  int main()  {  int num = 5;  int result = num++;  printf("Result: %d\n", result); // Output: 5  printf("Updated num: %d\n", num); // Output: 6  return 0;  }Result=5, num=6 |
| 4. What is the significance of the logical AND (&&) and logical OR (||) operators in C programming? How are they used in conditional expressions? |
| A: Logical AND (&&): The logical AND operator returns true if both of its operands are true; otherwise, it returns false. It short-circuits the evaluation, meaning that if the left operand evaluates to false, the right operand is not evaluated because the overall result will be false regardless.  Logical OR (||): The logical OR operator returns true if at least one of its operands is true; otherwise, it returns false. Similar to logical AND, it also short-circuits the evaluation. If the left operand evaluates to true, the right operand is not evaluated because the overall result will be true regardless.  #include <stdio.h>  int main()  {  int num = 10;  if (num > 5 && num < 15) {  printf("num is between 5 and 15\n");  }  if (num == 0 || num == 10) {  printf("num is either 0 or 10\n");  }  return 0;  } |
| 5. Discuss the concept of operator precedence and associativity in C programming. Provide examples to demonstrate how they affect expression evaluation. |
| A: Operator Precedence:  Operator precedence determines the order in which different types of operators are evaluated in an expression. Operators with higher precedence are evaluated before operators with lower precedence. For example, multiplication has higher precedence than addition, so in the expression 2 + 3 \* 4, the multiplication (\*) is evaluated first, resulting in 2 + (3 \* 4), which equals 14.  Associativity:  Associativity determines the order in which operators of the same precedence level are evaluated. It can be left-to-right or right-to-left.  #include <stdio.h>  int main()  {  int a = 5, b = 3, c = 2;  int result = a + b \* c;  printf("Result: %d\n", result);  int result2 = (a + b) \* c;  printf("Result2: %d\n", result2);  int x = 5, y = 3, z = 2;  x += y += z;  printf("x: %d, y: %d, z: %d\n", x, y, z);  return 0;  }  Result=11, Result2=16, x=10, y= 5, z= 2 |