|  |
| --- |
| **Control Strucutres:** |
| 1. Describe the purpose and usage of the switch statement in C programming. How does it differ from the if-else statement? |
| A: The switch statement in C programming provides a way to make decisions based on the value of a variable or expression. It's particularly useful when you have multiple conditions to check against the same variable. The switch statement evaluates the value of an expression and compares it to a list of case values. If a match is found, the corresponding block of code is executed.  It is different from if statement in following ways:  1.In a switch statement, the expression is evaluated once, and its value is compared against multiple constant values specified in the case labels. In contrast, an if-else statement evaluates each condition sequentially until a true condition is found.  2.In a switch statement, after a match is found, the corresponding block of code is executed, and then the switch statement is exited. In an if-else statement, if a condition evaluates to true, the corresponding block of code is executed, and then the program continues to evaluate subsequent conditions. |
| 2. Explain the concept of nested control structures in C programming. Provide an example demonstrating nested if-else statements. |
| A: Nested control structures in C programming refer to the situation where one control structure, such as an if statement, loop, or switch statement, is nested within another control structure. This allows for more complex decision-making and execution flows within a program.  Nested control structures enable developers to create logic that involves multiple levels of conditions or iterations. This approach is particularly useful when dealing with situations where decisions or iterations depend on multiple factors or conditions.  #include <stdio.h>  int main()  {  int num;  printf("Enter a number: ");  scanf("%d", &num);    if (num > 0)  {  printf("%d is positive.\n", num);  }  else {  if (num < 0)  {  printf("%d is negative.\n", num);  }  else {  printf("%d is zero.\n", num);  }  }    return 0;  } |
|  |
| 3. Discuss the role of the break and continue statements in loop control in C programming. Provide examples to illustrate their usage. |
| A: Break Statement:  The break statement is used to terminate the nearest enclosing loop.  When encountered within a loop, it immediately exits the loop, transferring control to the statement following the loop.  It's commonly used to exit a loop prematurely when a certain condition is met.  #include <stdio.h>  int main()  {  int i;  for (i = 1; i <= 10; i++)  {  if (i == 5)  {  printf("Encountered 5. Exiting loop.\n");  break; }  printf("%d ", i);  }    printf("\nAfter the loop\n");    return 0;  }  Continue Statement:  The continue statement is used to skip the rest of the current iteration of a loop.  When encountered within a loop, it immediately jumps to the next iteration, skipping any remaining code within the loop body.  It's commonly used to skip certain iterations based on specific conditions.  #include <stdio.h>  int main()  {  int i;  for (i = 1; i <= 10; i++)  {  if (i % 2 == 0) {  continue; // Skip even numbers  }  printf("%d ", i);  }    printf("\nAfter the loop\n");    return 0;  } |
| 4. What are the advantages of using the for loop over the while loop in C programming? Provide examples comparing the two. |
| A: Advantages of using the for loop over the while loop:  Initialization, Condition Checking, and Incrementing in One Place: The for loop allows initialization, condition checking, and incrementing of loop control variables to be done in a single line, making the code more compact and easier to read.  Visibility of Loop Control Variables: In a for loop, loop control variables are typically declared and initialized within the loop statement itself, making their scope more localized and improving code readability.  Suitability for Counter-Based Iterations: The for loop is well-suited for counter-based iterations, where the number of iterations is known beforehand and can be controlled using a loop counter.  Using for loop  #include <stdio.h>  int main()  {  int i;  printf("Using a for loop:\n");  for (i = 1; i <= 5; i++)  {  printf("%d ", i);  }  printf("\n");    return 0;  }  Using while loop  #include <stdio.h>  int main() {  int i = 1;  printf("Using a while loop:\n");  while (i <= 5)  {  printf("%d ", i);  i++;  }  printf("\n");    return 0;  } |
| 5. Explain the concept of short-circuit evaluation in C programming. How does it affect the evaluation of logical expressions in if statements? |
| A: Short-circuit evaluation is a concept in C programming where the evaluation of a logical expression stops as soon as the result of the entire expression is known based on the evaluation of its leftmost parts. This means that if the result of the expression can be determined without evaluating all of its components, the remaining components are not evaluated.  Logical AND (&&) Operator:  In an expression using the && operator, if the left operand evaluates to false (0), the right operand is not evaluated because the overall result will always be false regardless of the value of the right operand.  Only when the left operand evaluates to true (non-zero) does the evaluation proceed to the right operand. If the left operand is false, the overall result of the && expression is immediately known to be false, and there's no need to evaluate the right operand.  Logical OR (||) Operator:  In an expression using the || operator, if the left operand evaluates to true (non-zero), the right operand is not evaluated because the overall result will always be true regardless of the value of the right operand.  Only when the left operand evaluates to false (0) does the evaluation proceed to the right operand. If the left operand is true, the overall result of the || expression is immediately known to be tr1ue, and there's no need to evaluate the right operand. |