C continue in switch

C 루프 C 언어의 continue 문은 break 문과 약간 비슷합니다. 그러나 강제 종료는 아니며 continue는 현재 루프의 코드를 건너뛰고 다음 루프를 강제로 시작합니다. for 루프의 경우 continue 문이 실행된 후에도 자체 추가 문이 계속 실행됩니다. while 및 do의 경우... while 루프, continue 문은 조건부 판단 문을 다시 실행합니다. C 언어의 continue 문의 구문: continue; #include <stdio.h>main() { /* 지역 변수 정의 */ int a = 10; /* do 루프 실행 */ do { if(a == 15) { /* 반복 건너뛰기 */ a = a + 1; continue; } printf(a 값: %d, a); a++; }while(a < 20=); = = = return= 0; When the above code is compiled and executed, it produces the following result: the value of the value The value of: s 16 s a s: s 17 s a s value: s 18 s a #include s value s char c; for(k=1,c='A'; c < 'f'; = k++)= {= switch(++c)= {= case'a':= k++; = printf(%c= %d,c,k); break; = case'b':= k= *=2; printf(%c= %d,c,k); break; - Jump out of switch() to execute the statement that follows: case'c': s k---; = printf(%c= %d,c,k); Regardless of the value of the condition, continue with the statement after the next case 'd':): s k s % s 3; printf(%c= %d,c,k); continue; Do not execute the statement after the switch block, jump out of the this loop directly to the outer loop , case'e': s k s 2; = printf(%c= %d,c,k); = case'f':= k++; = printf(%c= %d,c,k); = default:= k= +=2; printf(%c= %d,c,k); All conditions do not meet, execute the statement after default. = printf(**********); = }= printf(%d,= k); = return= 0; = }= 1. In a switch statement, the case-constant expression is only equivalent to a statement label, and the value of the expression and a label are equal to that label, but the entire switch-statement cannot be automatically jumped out after the statement of that label is executed, so there will be a case in which all later case statements continue to execute. 2.? If you use content in switch, the continue effect is for the while loop . . . if you use breakbreak in switch, break is effective for switch. 3. If you use continue and break outside of switch, the effect is for the while loop. When using while or for loops, you can use the break orcontinue keyword if you want to end the loop early (ending the loop without meeting the end criteria). In the section C language switch s case statement, we talk about break and use it to jump out of the statement. When the .break?keyword is used for the ?while, for=loop, the loop is terminated and the code following the entire loop statement is executed. The break-keyword is usually used in the same as the if-statement, which jumps out of the loop when the condition is met. The value of 1 plus 100 is calculated by using the while cycle: s.#include.-gt; slt;stdio.h>int main()?int i=1, sum=0; while(1){ //루프 조건은 데드 루프 sum+=i입니다. i++; if(i>100) break; } printf(%d, sum); </stdio.h></stdio.h></stdio.h>0; Execution result: 5050 while loop condition is 1 and dead loop. i++ is calculated when you run 100 cycles. The latter i value is 101, in which case the condition i> 100 of the if statement is set and break is executed. Exit the loop. In a multi-layer loop, the break statement moves outside only one layer. For example, output a 4*4 integer matrix: <stdio.h>#include int main(), int i=1, j; while(1){ // outer loop j=1; while(1){ // Inner loop printf (%-4d, i*j); j++; if(j>4) break; jump from inner loop - printf (;); i++; if(i>4) break; jump out of the outer loop } return 0; run result: 1 2 3 4 2 4 6 8 3 6 9 12 4 8 12 16 j>4 is set, the brake runs; jumps out of the inner loop, and the outer loop runs with i> 4 set and protrudes from the outer loop. The inner loop was executed a total of four times and the outer loop was executed a total of one time. The continue statement skips the rest of the statement in the loop body and forces it to move to the next loop. The continue statement is used frequently with only while, for loops, and is often used with if conditional statements to determine whether the condition is true. for example: #include <stdio.h>int main(), char c = 0; while (c!=''){ //return key end loop c=getchar (); if (c=='4' || c=='5') { // pressed numeric key 4 or 5 continue. Skip the current loop and move to the next loop . } return 0; } Execution results : 0123456789↙ 01236789 When the program meets while, the value of variable c is '\0', the loop condition c!=' is set, and the first loop starts. getchar() prevents the program from running and reading characters until the user presses Enter and waits for the user to type. This example enters 0123456789, and when the if condition reads as 4 or 5, c=='4'|| If c=='5' is set, the continue statement is executed. It does not run. When reading other numbers, the conditions of if are not set, and the continue statement is not executed, putchar (c); The read characters are output. Comparison of break and continue: break is used to terminate all loops, and loop statements are no longer executable. In the last topic, we studied different types of loops. We also saw how loops can be nested. Normally, if we have to choose one case among many choices, if-else is used. But if the number of choices is large, switch.. case makes it a bit easier and less complex. Let's control Case Wise switch... case is another way to control and decide the execution of statements other than if/else. This is used when we are given a number of choices (cases) and we want to perform a different task for each choice. Let's first have a look at its syntax. switch(expression) { case constant1: statement(s); break; case constant2: statement(s); break; /* you can give any number of cases */ default: statement(s); } In switch... case, the</stdio.h> </stdio.h> 。() The representation of the enclosed expression in parentheses is confirmed according to the following switch. If the expression value matches the constant value in all cases, the statement corresponding to that case is executed. If the expression does not match a constant value, the statement corresponding to the default value is executed. Let's take an example. #include <stdio.h>int Main () { char grade; printf (grade input); Scan (%c, And grades); switches (grades) { case 'A': printf (excellent!); rest; case 'B': printf (outstanding!); brakes; case 'C': printf (good!); rest; case 'D': printf (can do better); rest; case 'E': printf (just passed); rest; case 'F': printf (failure); rest; The default print (invalid rating); output input your Grade DCan better break is used to break or exit the loop whenever we want and is also used with a switch; in this example, the value of 'grade' is 'D'; the constant value of the first three cases is not 'D', so 'D' is executed and 'can do better' is printed. The break statement then ends execution without checking the rest of the cases. If there is no interruption in the statement, all cases are executed after executing the correct case. For example, look at the following code. #include <stdio.h>int main () { char grade and printf (grade entry); scan (%c) , and ratings); Switch (grade) { case 'A': printf (excellent!); case 'B': printf (outstanding!); case 'C'); : printf (good!); Case 'D': printf (can do better); Case 'E': printf (just passed); Case 'F': printf (failed); Basic: printf (invalid rating); } Return 0; } Input output input gradeDCan performs betterThe better passyeon fail in the example above, the value of the rating is 'D', so the control jumps to 'D' if. All statements of the case 'D' are also executed because there is no door to rest after any case. As you can see, all cases since Case D have been executed. Use break statements to run only those cases where a constant value is the same as the expression value of the switch statement. Always enclose the character value within ''. You can now view the expression value as an integer. #include <stdio.h>int main () { int i = 2; switch (i) { case 1: print (number 1); break; case 2: print (number 2); rest; default: print (number is greater than 2); } Return 0; } Use interruptions with output loops to exit the loop in the middle of execution using interruptions. Just a break type; To break the loop after that, after the statement. It's as simple as that! Let's take an example. #include <stdio.h>int Main () { int a; <= 10; a++) { printf (Hello World); if (a == 2) { //loop will now stop break; } } return 0; } Output In this example, after the first iteration of the loop, a++ increases the value of 'a' to 2 and 'Hello 10;= a= ++)= {= printf(hello= world);= if(a== 2)= {= loop= will= now= stop= break;= }= }= return= 0;= }= output= in= this= example,= after= the= first= iteration= of= the= loop,= a++= increases= the= value= of= 'a'= to= 2= and= 'hello=></= 10; a ++) { printf(Hello World); if(a == 2) { //loop will now stop break; } } return 0; } Output In this example, after the first iteration of the loop, a++ increases the value of 'a' to 2 and 'Hello > (a = 1;</stdio.h> </stdio.h> </stdio.h>인쇄되었습니다. 이 때 에 앉아 있는 경우의 조건이 실행되고 루프가 종료됩니다. 계속 문은 중단 문과 유사하게 작동합니다. 유일한 차이점은 중단 문이 루프를 종료하는 반면 계속 문은 조건이 검사되는 조건부 테스트(예: 루프의 나머지 문)를 건너뛰는 조건부 테스트로 컨트롤을 전달한다는 것입니다. #include <stdio.h>int 메인 () { int a; <= 10; a ++) { printf(Hello World); if (a == 2) { //this time further statements will not be executed. Control will go to for continue; } printf(a is not 2); } return 0; } Output Hello World a is not 2 Hello World Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Notice that at the second time, 'a is not 2' is not printed. It means that when 'a' was 2, then 'continue' got executed and control went to for loop without executing further codes. Knowing is not enough, we must apply. Willing is not enough, We must do. - Bruce Lee Lee 10;= a= ++)= {= printf(hello= world);= if= (a== 2)= {= this= time= further= statements= will= not= be= executed.= control= will= go= to= for= continue;= }= printf(a= is= not= 2);= }= return= 0;= }= output= hello= world= a= is= not= 2= hello= world= hello= world= a= is= not= 2= hello= world= a= is= not= 2= hello= world= a= is= not= 2= hello= world= a= is= not= 2= hello= world= a= is= not= 2= hello= world= a= is= not= 2= hello= world= a= is= not= 2= hello= world= a= is= not= 2= notice= that= at= the= second= time,= 'a= is= not= 2'= is= not= printed.= it= means= that= when= 'a'= was= 2,= then= 'continue'= got= executed= and= control= went= to= for= loop= without= executing= further= codes.= knowing= is= not= enough ,= we= must= apply.= willing= is= not= enough,= we= must= do.= -= bruce= lee= lee=></= 10; a ++) { printf(Hello World); if (a == 2) { //this time further statements will not be executed. Control will go to for continue; } printf(a is not 2); } return 0; } Output Hello World a is not 2 Hello World Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Hello World a is not 2 Notice that at the second time, 'a is not 2' is not printed. It means that when 'a' was 2, then 'continue' got executed and control went to for loop without executing further codes. Knowing is not enough, we must apply. Willing is not enough, We must do. - Bruce Lee Lee > (a = 1;</stdio.h>