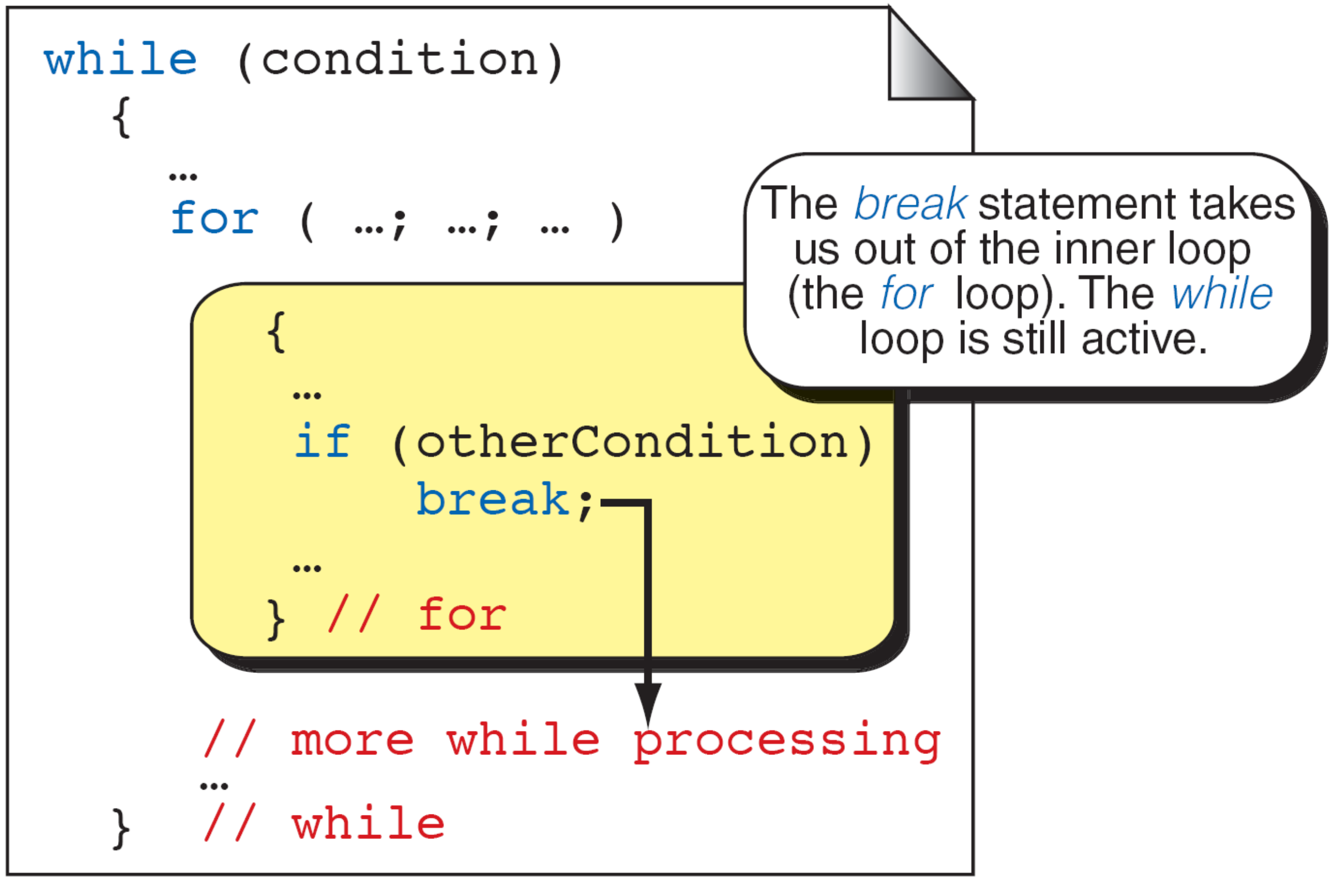
Control Structures

Part 4

Dr Bhanu

Break and continue Statements

- The `break` and `continue` statements are used to alter the flow of control.
- The `break` statement, when executed in a `while`, `for`, `do`…`while` or `switch` statement, causes an immediate exit from that statement.
- Program execution continues with the next statement.
- Common uses of the `break` statement are to escape early from a loop or to skip the remainder of a `switch` statement.



```
while (condition)
  {
    …
    for ( …; …; … )
      {
       …
       if (otherCondition)
           break;
       …
      } // for
     // more while processing
     …
  }  // while
```

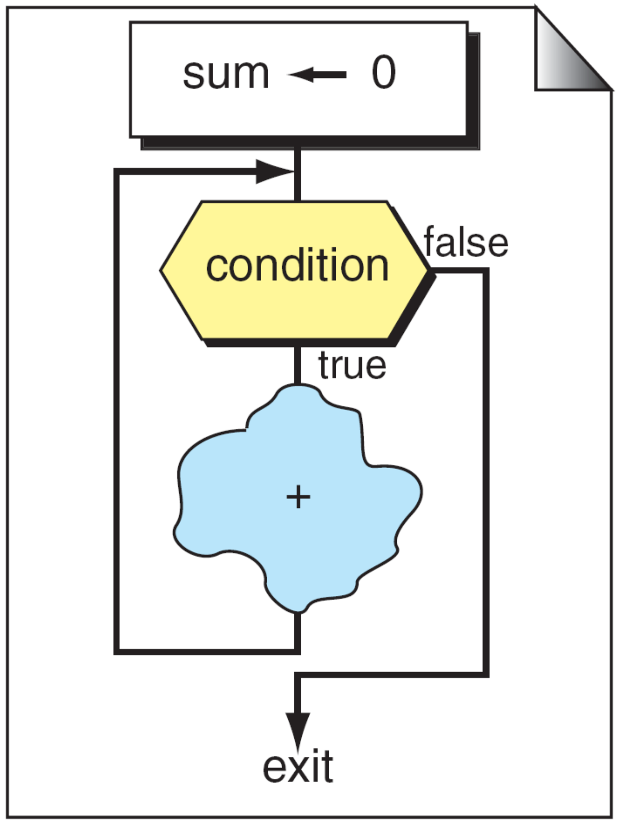
The *break* statement takes us out of the inner loop (the *for* loop). The *while* loop is still active.

break and Inner Loops

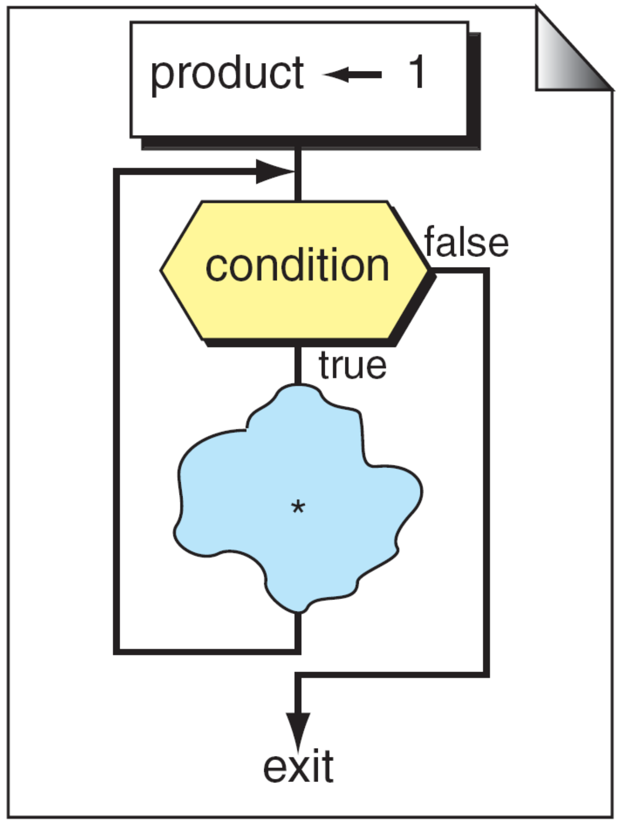
Looping Applications

There are four common applications for loops: summation, product, smallest and largest, and inquiries. Although the uses for loops are virtually endless, these problems illustrate many common applications.

Summation
product
Smallest and Largest
Inquiries



Summation



Product

Summation and Product Loops

Break and continue Statements

- Next example demonstrates the break statement in a `for` repetition statement.
- Using the `if` statement and a condition, `break` is executed to terminate the loop prematurely.
- This terminates the `for` statement, and the program continues with the `next statement` after the `for loop body`.

```
#include <stdio.h>
int main()
{
   int n, i, sum = 0;

   printf("Enter a positive integer: ");
   scanf("%d", &n);

   for (i = 1; i <= n; ++i)
   {
      sum += i;
   }

   printf("Sum = %d", sum);
   return 0;
}
```

Requirement:

Terminate the loop when the sum exceeds, for example 1000.

Can we use the break statement to overcome “divide-by-zero” error?

Can we use the break statement to overcome “overflow / Underflow” error?

- Factorial problem

Break and continue Statements

- The continue statement, when executed in a while, for or do…while statement, skips the remaining statements in the body of that control statement and performs the next iteration of the loop.
- In while and do…while statements, the loop-continuation test is evaluated immediately after the continue statement is executed.
- In the for statement, the increment expression is executed, then the loop-continuation test is evaluated.

Break and continue Statements

- The `while` statement could be used in most cases to represent the `for` statement.
- The one exception occurs when the increment expression in the `while` statement follows the `continue` statement.
- In this case, the increment is not executed before the repetition-continuation condition is tested, and the `while` does not execute in the same manner as the `for`.
- Next example shows the difference in execution of break and continue statements

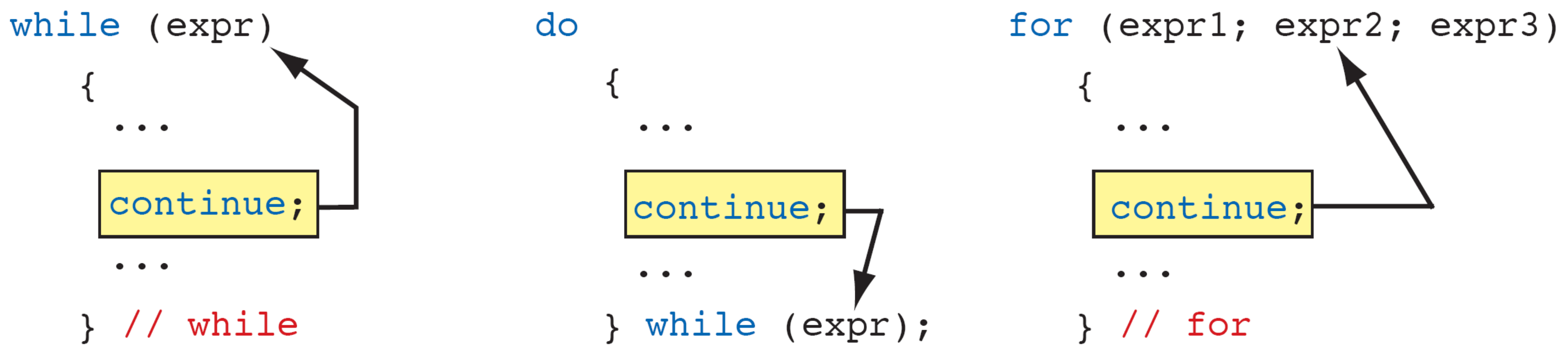
```
#include <stdio.h>
int main()
{
   int n, i, sum = 0;

   printf("Enter a positive integer: ");
   scanf("%d", &n);
   printf("n = %d\n", n);


   for (i = 1; i <= n; ++i)
   {
      sum += i;

      if (sum >= 100 && sum <= 200)
         //continue;
         break;
      printf("cnt = %d     Sum = %d\n", i, sum);
   }

   return 0;
}
```



```
while (expr)
  {
   ...
   continue;
   ...
  } // while
```

```
do
  {
   ...
   continue;
   ...
  } while (expr);
```

```
for (expr1; expr2; expr3)
  {
   ...
   continue;
   ...
  } // for
```

FIGURE 6-21 The *continue* Statement

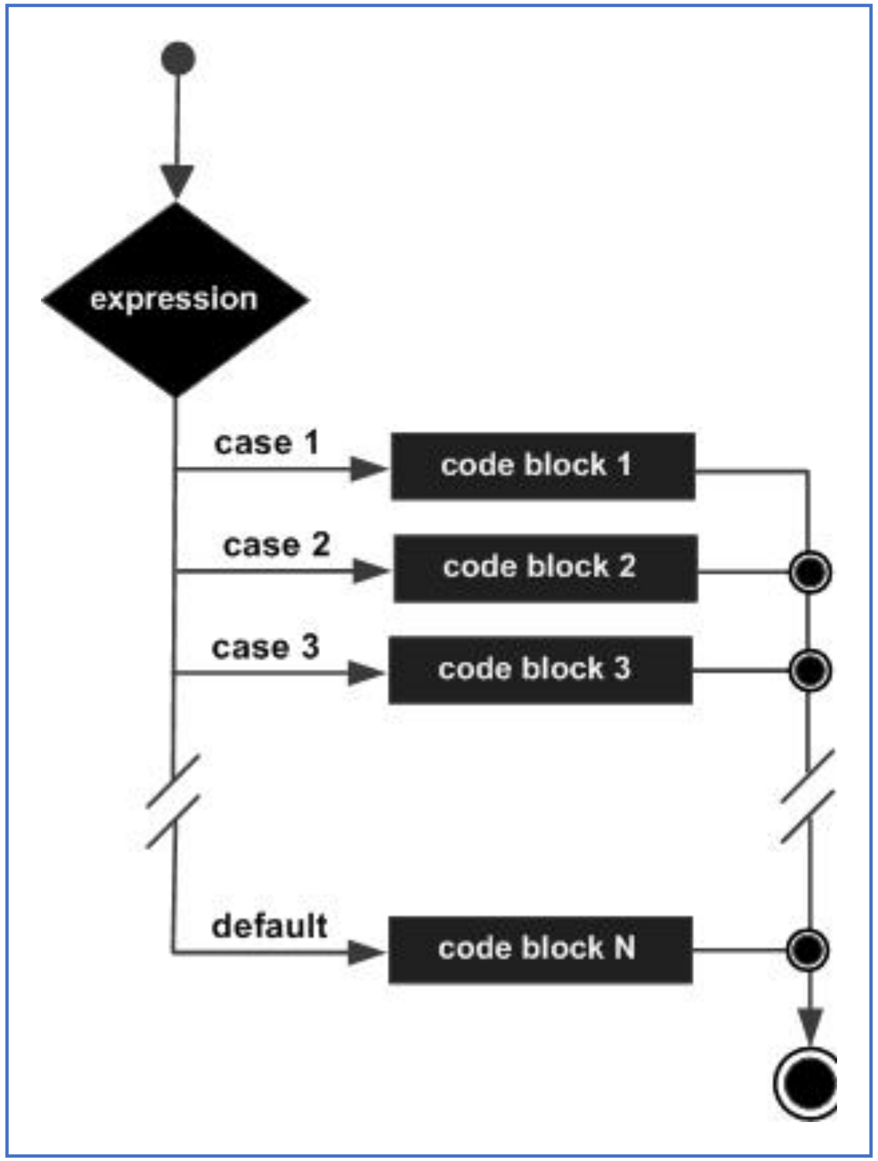
Key	Break	Continue
Functionality	The break statement is mainly used to terminate the loop such as while, do-while, for and also in switch statement	The Continue statement is used to skip the rest of loop and execute the next iteration.
Executional flow	The break statement causes program to resume at the end of loop. The program will continue outside the terminated loop.	The Continue statement resumes the control of the program to the next iteration of that loop.
Usage	Used to terminate the loop.	Used to cause early execution of the next iteration of the loop.
Compatibility	Break statement is compatible with 'switch' statement.	Not compatible with 'switch'

`switch` Multiple-Selection Statement

- Occasionally, an algorithm will contain a series of decisions in which a variable or expression is tested separately for each of the constant integral values it may assume, and different actions are taken.
- This is called multiple selection.
- C provides the `switch` multiple-selection statement to handle such decision making.
- The `switch` statement consists of a series of `case` labels, an optional `default` case and statements to execute for each case.

switch Multiple-Selection Statement

- A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
- A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.



```
#include <stdio.h>

int main ()
 {
   char grade;

   printf("Enter the Grade   :  ");
   scanf("%c", &grade);
   //printf("\nThe grade entered is : %c\n", grade);

   switch(grade) {
      case 'A' :
      case 'a' :
         printf("Excellent!\n" );
         break;

      case 'B' :
      case 'b' :
      case 'C' :
       case 'c' :
         printf("Well done\n" );
         break;
case 'D' :
      case 'd' :
         printf("You passed\n" );
         break;

      case 'F' :
      case 'f' :
         printf("Better try again\n" );
         break;

      default :
         printf("Invalid grade\n" );
   }

   printf("Your grade is  %c\n", grade );

   return 0;
}
```

```c
#include <stdio.h>

int main() {
    char operator;
    double n1, n2;

    printf("Enter an operator (+, -, *, /): ");
    scanf("%c", &operator);
    printf("Enter two operands: ");
    scanf("%lf %lf",&n1, &n2);

    switch(operator)
    {
        case '+':
            printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
            break;

        case '-':
            printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
            break;
case '*':
            printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
            break;

        case '/':
            printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
            break;

        // operator doesn't match any case constant +, -, *, /
        default:
            printf("Error! Invalid operator ");
    }

    return 0;
}
```

Why do we need a Switch case?

There is one potential problem with the if-else statement which is the complexity of the program increases whenever the number of alternative paths increases. If you use multiple if-else constructs in the program, a program might become difficult to read and comprehend. Sometimes it may even confuse the developer who himself wrote the program.

Rules for switch statement:

- A switch must contain an executable test-expression.
  Case labels must be constants and unique.
- Case labels must end with a colon (:).
- A break keyword must be present in each case.
- There can be only one default label.