4.9 Break and continue

A ***break statement*** in a loop causes an immediate exit of the loop. A break statement can sometimes yield a loop that is easier to understand.

Figure 4.9.1: Break statement: Meal finder program.

```
#include <iostream>
using namespace std;

int main() {
   const int EMPANADA_COST = 3;
   const int TACO_COST     = 4;

   int userMoney;
   int numTacos;
   int numEmpanadas;
   int mealCost;
   int maxEmpanadas;
   int maxTacos;

   mealCost = 0;

   cout << "Enter money for meal: ";
   cin >> userMoney;

   maxEmpanadas = userMoney / EMPANADA_COST;
   maxTacos = userMoney / TACO_COST;

   for (numTacos = 0; numTacos <= maxTacos; ++numTacos) {
      for (numEmpanadas = 0; numEmpanadas <= maxEmpanadas; ++numEmpanadas) {

         mealCost = (numEmpanadas * EMPANADA_COST) + (numTacos * TACO_COST);

         // Find first meal option that exactly matches user money
         if (mealCost == userMoney) {
            break;
         }
      }

      // If meal option exactly matching user money is found,
      // break from outer loop as well
      if (mealCost == userMoney) {
         break;
      }
   }

   if (mealCost == userMoney) {
      cout << "$" << mealCost << " buys " << numEmpanadas
      << " empanadas and " << numTacos << " tacos without change." << endl;
   }
   else {
      cout << "You cannot buy a meal without having change left over." << endl;
   }

   return 0;
}
```

```
Enter money for meal: 20
$20 buys 4 empanadas and 2 tacos without change.

...

Enter money for meal: 31
$31 buys 9 empanadas and 1 tacos without change.
```

Feedback?

The nested for loops generate all possible meal options for the number of empanadas and tacos that can be purchased. The inner loop body calculates the cost of the current meal option. If equal to the user's money, the search is over, so the break statement immediately exits the inner loop. The outer loop body also checks if equal, and if so that break statement exits the outer loop.

The program could be written without break statements, but the loops' condition expressions would be more complex and the program would require additional code, perhaps being harder to understand.

PARTICIPATION ACTIVITY 4.9.1: Break statements.

Given the following while loop, what is the value assigned to variable z for the given values of variables a, b and c?

```
mult = 0;

while (a < 10) {
   mult = b * a;
   if (mult > c) {
      break;
   }
   a = a + 1;
}
z = a;
```

1) a = 1, b = 1, c = 0

Check Show answer

2) a = 4, b = 5, c = 20

Check Show answer

Feedback?

A ***continue statement*** in a loop causes an immediate jump to the loop condition check. A continue statement can sometimes improve the readability of a loop. The example below extends the previous meal finder program to find meal options for which the total number of items purchased is evenly divisible by the number of diners. The program also outputs all possible meal options, instead of just reporting the first meal option found.

Figure 4.9.2: Continue statement: Meal finder program that ensures items purchased is evenly divisible by the number of diners.

```
#include <iostream>
using namespace std;

#include <stdio.h>

int main() {
   const int EMPANADA_COST = 3;
   const int TACO_COST     = 4;

   int userMoney;
   int numTacos;
   int numEmpanadas;
   int mealCost;
   int maxEmpanadas;
   int maxTacos;
   int numOptions;
   int numDiners;

   mealCost = 0;
   numOptions = 0;

   cout << "Enter money for meal: ";
   cin >> userMoney;

   cout << "How many people are eating: ";
   cin >> numDiners;

   maxEmpanadas = userMoney / EMPANADA_COST;
   maxTacos     = userMoney / TACO_COST;

   numOptions = 0;
   for (numTacos = 0; numTacos <= maxTacos; ++numTacos) {
      for (numEmpanadas = 0; numEmpanadas <= maxEmpanadas; ++numEmpanadas) {

         // Total items purchased must be equally
         // divisible by number of diners
         if ( ((numTacos + numEmpanadas) % numDiners) != 0) {
            continue;
         }

         mealCost = (numEmpanadas * EMPANADA_COST) + (numTacos * TACO_COST);

         if (mealCost == userMoney) {
            cout << "$" << mealCost << " buys " << numEmpanadas
                 << " empanadas and " << numTacos
                 << " tacos without change." << endl;
            numOptions = numOptions + 1;
         }
      }
   }

   if (numOptions == 0) {
      cout << "You cannot buy a meal without "
           << "having change left over." << endl;
   }

   return 0;
}
```

```
Enter money for meal: 60
How many people are eating: 3
$60 buys 12 empanadas and 6 tacos without change.
$60 buys 0 empanadas and 15 tacos without change.

...

Enter money for meal: 54
How many people are eating: 2
$54 buys 18 empanadas and 0 tacos without change.
$54 buys 10 empanadas and 6 tacos without change.
$54 buys 2 empanadas and 12 tacos without change.
```

Feedback?

The nested loops generate all possible combinations of tacos and empanadas. If the total number of tacos and empanadas is not exactly divisible by the number of diners (e.g., `((numTacos + numEmpanadas) % numDiners) != 0)`, the continue statement proceeds to the next iteration, thus causing incrementing of numEmpanadas and checking of the loop condition.

Break and continue statements can avoid excessive indenting/nesting within a loop. But they could be easily overlooked, and should be used sparingly, when their use is clear to the reader.

PARTICIPATION ACTIVITY 4.9.2: Continue.

Given:

```
for (i = 0; i < 5; ++i) {
   if (i < 10) {
      continue;
   }
   // Put i to output
}
```

1) The loop will print at least some output.

- True
- False

2) The loop will iterate only once.

- True
- False

Feedback?

CHALLENGE

ACTIVITY | 4.9.1: Simon says.

"Simon Says" is a memory game where "Simon" outputs a sequence of 10 characters (R, G, B, Y) and the user must repeat the sequence. Create a for loop that compares the two strings starting from index 0. For each match, add one point to userScore. Upon a mismatch, exit the loop using a break statement. Assume simonPattern and userPattern are always the same length. Ex: The following patterns yield a userScore of 4:

```
simonPattern: RRGBRYYBGY
userPattern:  RRGBBRYBGY
```

```
#include <iostream>
#include <string>
using namespace std;

int main() {
   string simonPattern;
   string userPattern;
   int userScore;
   int i;

   userScore = 0;

   cin >> simonPattern;
   cin >> userPattern;

   /* Your solution goes here  */
   for(i = 0; i<simonPattern.size(); i++){
      if(simonPattern.at(i) != userPattern.at(i))
      {
         break;
      }else{
```

Run ✓ All tests passed

✓ Testing simonPattern: RRGBRYYBGY and userPattern: RRGBBRYBGY

Your output
```
userScore: 4
```

✓ Testing simonPattern: RRRRRRRRRR and userPattern: RRRRRRRRRY

Your output
```
userScore: 9
```

✓ Testing simonPattern: YBYBYBRRRR and userPattern: GGGGGGGGGG

Your output
```
userScore: 0
```

Feedback?