

BRANCHING CONTROL STRUCTURES*

Kenneth Leroy Busbee

This work is produced by OpenStax-CNXX and licensed under the
Creative Commons Attribution License 3.0[†]

Abstract

An introduction to the common types of branching control structures and which ones are allowed in good structured programming.

1 Discussion

The branching control structures allow the flow of execution to jump to a different part of the program. The common branching control structures that are used with other control structures are: break, continue and goto. These are rarely used in modular structured programming with one exception. That exception is in relation to creating the case within the selection category of control structures. Within C++ the break is used with the switch to create a structure that acts like the traditional case structure. There is one other branching control structure that is often not viewed as branching control structure. It is: return; which is used with functions. Thus, there are two commonly used branching control reserved words used in C++; break and return. Additionally, we will add to our list of branching items a pre-defined function commonly used in the C++ programming language of: exit; that is part of the C standard library (stdlib). Some definitions:

1.1 Definitions

Definition 1: branching control structures

Allow the flow of execution to jump to a different part of the program.

Definition 2: break

A branching control structure that terminates the existing structure.

Definition 3: continue

A branching control structure that causes a loop to stop its current iteration and begin the next one.

Definition 4: goto

A branching control structure that causes the logic to jump to a different place in the program.

Definition 5: return

A branching control structure that causes a function to jump back to the function that called it.

Definition 6: exit

A pre-defined function used to prematurely stop a program and jump to the operating system.

We will discuss each item indicating which ones are allowed or not allowed within good structured programming practices.

*Version 1.4: Oct 23, 2012 10:03 am -0500

[†]<http://creativecommons.org/licenses/by/3.0/>

2 Examples

2.1 break

The break is used in one of two ways; with the switch (a C++ programming structure) to make it act like a case structure (it's more common name within most programming languages) or as part of a looping process to break out of the loop. The first usage is allowed in good structured programming and the second is not allowed in good structured programming.

Example 1: C++ source code

```
switch (age)
{
  case 18: cout << "\nYou can vote.";
           break;
  case 39: cout << "\nYou are middle aged.";
           break;
  case 65: cout << "\nYou are at retirement age.";
           break;
  default: cout << "\nYour current age is not important.";
}
}
```

The following is an unauthorized use of break in a loop and it gives the appearance that the loop will execute 8 times, but the break statement causes it to stop during the fifth iteration.

Example 2: C++ source code

```
counter = 0;
while(counter < 8)
{
  cout << counter << endl;
  if (counter == 4)
  {
    break;
  }
  counter++;
}
```

2.2 continue

The continue structure is not allowed in good structured programming. The following gives the appearance that the loop will print to the monitor 8 times, but the continue statement causes it not to print number 4.

Example 3: C++ source code

```

for(counter = 0; counter < 8; counter++)
{
  if (counter == 4)
  {
    continue;
  }
  cout << counter << endl;
}

```

2.3 goto

The goto structure is not allowed in good structured programming. It is with a certain amount of hesitancy that we even show it. Many textbooks do not cover the goto. Within the C++ programming language you create a label with an identifier name followed by a colon. You use the command word goto followed by the label. A label can be used before it is declared.

Example 4: C++ source code

```

some lines of code;
goto mynewspot;           //jumps to the label
some lines of code;
some lines of code;
some lines of code;
mynewspot: some statement; //Declared label
some lines of code;

```

2.4 return

The return is allowed in good structured programming, but only at the end of a function. A function should not pre-maturely end by having the logic of the function have it terminate by jumping back to the function that called it.

Example 5: C++ source code

```

//*****
// get data
//*****

void get_data(void)
{
  // Input - Test Data - 5678.9, 5432.1
  cout << "\nEnter the length of the property in feet --->: ";
}

```

```
cin >> property_length;
cout << "\nEnter the width of the property in feet ---->: ";
cin >> property_width;
return;
}
```

2.5 exit

Although `exit` is technically a pre-defined function, it is covered here because of its common usage in programming. A good example is the opening a file and then testing to see if the file was actually opened. If not, we have an error that usually indicates that we want to pre-maturely stop the execution of the program. Within the C++ programming language the `exit` function terminates the running of the program and in the process returns an integer value back to the operating system. It fits the definition of branching which is to jump to some other place in the program. In our example the value returned to the operating system is the value of the constant named: `EXIT_FAILURE`.

Example 6: C++ source code

```
inData.open(filename); //Open input file
if (!inData)           //Test to see if file was opened
{
    cout << "\n\nError opening file: " << filename << "\n\n";
    pause();           //Pause - user reads message
    exit(EXIT_FAILURE); //Allows a pre-mature jump to OS
}
```