Chapter - 6 Decision and Control Statements

What are decision and control statements

We've been working on linear programs. That is we start the program and execute each statement in a straight line until we reach the end.

Decision and control statements allow us to change the flow of the program.

Branching statements cause one section of code to be executed or not depending on a *conditional clause*.

Looping statements allow a section of code to be repeated a number of times or until a condition occurs.

if Statement

General form:

```
if (condition)
    statement;
```

If the condition is true (non-zero), the statement is executed.
If the condition is false (zero), the statement is not executed.

Example:

```
if (total_owed <= 0)
    std::cout << "You owe nothing.\n";
```

Relational Operators

Operator	Meaning
<=	Less than or equal to.
<	Less than.
>	Greater than.
>=	Greater than or equal to.
==	Equal.
!=	Not equal.

Example:

```
if (oper_char == 'Q')
    std::cout << "Quit\n";
```

Logical Operators

Operator	Usage	Meaning
logical or (`||`)	`(`*`expr1`*`)|| (`*`expr2`*`)`	True if *expr1* or *expr2* is true
logical and (&&)	`(`*`expr1`*`) && (`*`expr2`*`)`	True if *expr1* and *expr2* are both true
logical not (`!`)	`!(`*`expr`*`)`	Returns false if *expr* is true Returns true if *expr* is false

Example:

```
if (total_owed <= 0) {
    ++zero_count;
    std::cout << "You owe nothing.\n";
}
```

Note the use of curly braces ({}) to group multiple statements together so they are treated as a single statement.

`Else` Statement

General form:

```
if (condition)
     statement;
else
     statement;
```

Example:

```
if (total_owed <= 0)
    std::cout << "You owe nothing.\n";
else
    std::cout << "You owe " << total_owed << " dollars\n";
```

Question: Which *if* does the *else* belong to?

```
if (count < 10)      // if #1
    if ((count % 4) == 2)   // if #2
        std::cout << "Condition:White\n";
  else   // (Indentation is wrong)
      std::cout << "Condition:Tan\n";
```

a. It belongs to **if** #1.
b. It belongs to **if** #2.
c. You don't have to worry about this situation if you never write code like this.

How not to use `std::strcmp`

The logic of the following code appears to be simple, yet it confuses many programmers.

```
if (std::strcmp(string1, string2))
    std::cout << ".....";
```

Does the `std::cout` statement executes if the two C style strings are equal or not equal.

A better use of `std::strcmp` is:

```
// Check for Equal
if (std::strcmp(string1, string2) == 0)
   std::cout << "Strings equal\n";
else
   std::cout <<"Strings not equal\n";
```

Better yet, stick to C++ strings.

`While` Statement

General format:

```
while (condition)
      statement;
```

Example:

```
counter = 0;
while (counter < 5) {
    total += balance[counter];
    counter++;
}
```

Fibonacci Sequence

The Fibonacci sequence is:

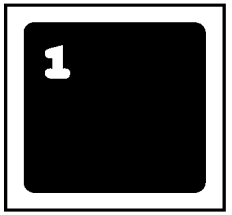
```
   1 1 2 3 5 8 . . .
```

The terms are computed from the equations:

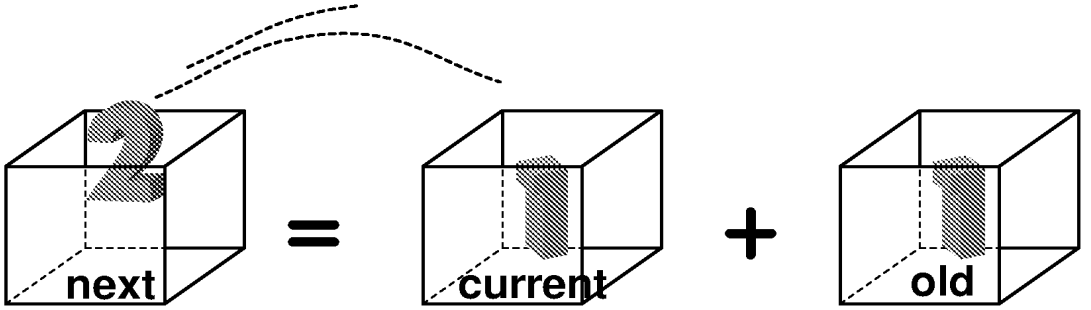
```
1
1
2 = 1 + 1
3 = 1 + 2
5 = 3 + 2
etc.
```

In general terms this is: $f_n = f_{n-1} + f_{n-2}$

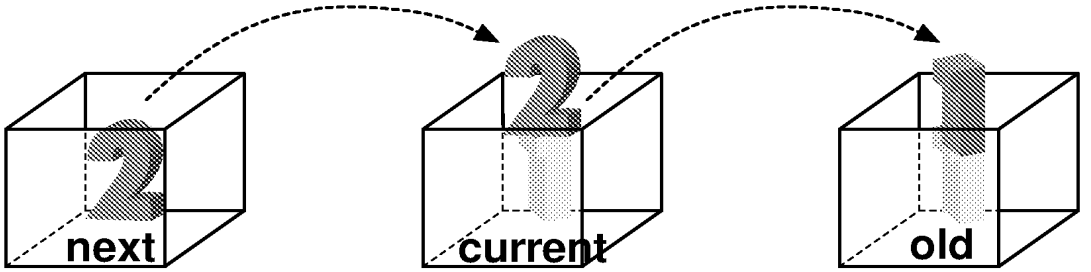
Fibonacci execution




```
std::cout << current_number << '\n';
next_number = current_number + old_number;
```




```
old_number = current_number; current_number = next_number;
```



Fibonacci Program

```
#include <iostream>
int   old_number;     // previous Fibonacci number
int   current_number; // current Fibonacci number
int   next_number;    // next number in the series


int main(){
    // start things out
    old_number = 1;
    current_number = 1;


    std::cout << "1\n"; // Print first number


    while (current_number < 100) {
        std::cout << current_number << '\n';
        next_number = current_number + old_number;


        old_number = current_number;
        current_number = next_number;
    }
    return (0);
}
```

Break Statement

The **break** statement causes the program to exit the innermost loop.

Example:

```
if (item == 0)
    break;
```

Break Example

```
#include <iostream>
int   total;  // Running total of all numbers so far
int   item;   // next item to add to the list


int main()
{
    total = 0;
    while (1) {
        std::cout << "Enter # to add \n";
        std::cout << "  or 0 to stop:";
        std::cin >> item;


        if (item == 0)
            break;


        total += item;
        std::cout << "Total: " << total << '\n';
    }
    std::cout << "Final total " << total << '\n';
    return (0);
}
```

Assignment Anywhere Side Effect

In C++ you can use assignment statements almost anyplace.

```
// don't program like this
average = total_value /
      (number_of_entries = last - first);
```

This is the equivalent of saying:

```
// program like this
number_of_entries = last - first;


average = total_value / number_of_entries;
```

You can even put an assignment statement in a conditional. *Please don't!!*

```
// do not program like this
while ((current_number = last_number + old_number) < 100)
    std::cout << "Term " << current_number << '\n';
```

Question: Why does everyone owe 0 dollars?

```
#include <iostream>
int   balance_owed;     // amount owed


int main()
{
    std::cout << "Enter number of dollars owed:";
    std::cin >> balance_owed;


    if (balance_owed = 0)
        std::cout << "You owe nothing.\n";
    else
        std::cout << "You owe " <<
                     balance_owed << " dollars.\n";


    return (0);
}
```

Sample output

```
Enter number of dollars owed: 12
You owe 0 dollars.
```