

Take 1, 2 or 3 Human vs. Human

Using Bloodshed Dev C++

By Janine Bouyssounouse

Bloodshed Dev C++ is a free program to make writing and compiling C++ programs easy to do. It makes executable files quickly so programs can be shared with others as soon as they are written and debugged. Bloodshed Dev C++ can be downloaded from the website:

<http://www.bloodshed.net/devcpp.html>.

There is more than one way to start writing programs in Dev C++. This document discusses using a project for each program. This encourages using an organized file structure to keep each program in a separate location for future use. Remember that reusing code is a good idea and it is nice to be able to find the folder with everything for a program in one place.

Click on the File menu and select New – Project. Choose Console Application for this program. Type in a name for the project, such as Display Input, in the name field in the bottom left corner of the window. Click on the OK button. Choose a place on the computer to save the project. It's a good idea to make a folder for each project. Once the project is saved, some basic items show up on the screen. These items give a shell of a C++ program and include things that are unique to Bloodshed Dev C++ as well.

```
#include <iostream>      - This tells the compiler to include files.
#include <stdlib.h>      - This tells the compiler to include files.

using namespace std;    - This tells the compiler that you will be using key
                        words that are included in namespace std.

int main(int argc, char *argv[])    - This starts the main function.
{                                     - Main is contained between { and }.

    system("PAUSE");                - This leaves the console window open until
                                    you are ready to close it. Otherwise you
                                    wouldn't be able to see the results of the
                                    program. Not all compilers require this.

    return 0;                       - This is how the main program ends.
}                                     - This signifies the end of the code for main.
```

The code for the main part of the program will be typed between the first curly brace ({} and the system("PAUSE") line of code. The input and print functions for this program will be outside of this area, but the main program will refer to them.

The Programming Challenge: Write a program to simulate a game with a pile of 21 matches. This will be a two player game using two human players. Each player takes 1, 2 or 3 matches from the pile on their turn. The player to take the last match loses. Don't forget to include a welcome message that explains the game to the users.

Hints: Work on the different parts of the program, one at a time. Work on the welcome message first, since that is the easiest part.

Then try working on a function to take a turn for one player. This can be called each time a player needs to make a play. It will stay the same if you use the player1 and player2 as a variable that is passed to it as well as the total number of matches still in the pile.

Then work on the main game loop in the main part of the program. A do while loop will work great for a game loop. It can keep going until the matches are done. Don't forget to update the number of matches taken from the pile each turn and change which player has their turn next.

Try making a function to announce the winner of the game when the game is over. It can be called when the game loop exits.

The Walk Through: Starting with the welcome message...

The welcome message needs to explain the rules of the game to the users so they know what is happening and how to play the game. It would be good to include information on how the winner is determined.

Type in the first lines of the program by clicking at the beginning of the first line and pressing enter a couple of times, then moving to the first blank line. Type in these first lines of code:

```
// This program is a game with two human users. 21 matches are in a
// pile and players take turns taking 1, 2 or 3 matches. The program
// loops until the matches are gone and announces who won.
```

The next line of code can be a comment stating your name as the programmer and the date the program was written.

```
// Written by Janine Bouyssounouse on 10/19/08
```

Type the following line of code after the using namespace statement:

```
void welcome();
```

A welcome statement will be displayed to explain what this program does. There is nothing sent to or received from the welcome function, so the void and empty parentheses are used in the function prototype.

On the last line of the program, we will comment the line to show that the main function is finished, so that it is not confused with the other functions listed after it.

The last line of code should look like this:

```
} // end main
```

Next we will start typing the welcome function at the end of the program, outside of the curly braces for the main function.

Skip a line and type:

```
// welcome function displays an opening message to
// explain the program to the user
void welcome()
```

Notice the comments are listed on the lines before the start of the function. The first line of the function looks exactly like the function prototype, except for the missing semicolon at the end.

On the next line of code, type in the function:

```
{
    cout << "Imagine a pile of 21 matches.\n";
    cout << "There are two human players.\n";
    cout << "Each player takes one, two or three matches on their turn.\n";
    cout << "The player who takes the last match loses.\n\n";
} // end of welcome function
```

This function uses no variables at all.

Call the welcome function in the main function to display the information to the user.

To do this, type the following code after the first curly brace in the main function and before the system("PAUSE") line of code:

```
welcome(); // This calls the welcome function
```

The Walk Through: Initializing the variables...

Variables are going to be passed to functions and need to be kept track of in the main function to know when to end the game (when matchTotal is zero) to know whose turn it is (whoseTurn = Player 1 or Player 2) and to know how many matches were taken from the pile on each turn (turnAmount).

It is good practice to initialize variables when they are created. In this game it helps make the rest of the program flow well. matchTotal will be initialized to the starting amount of 21 matches. whoseTurn will be initialized to Player1. turnAmount will be initialized to 0, since no turns

have been taken yet and there should be no amount until a player decides how many they will take. The variables will be initialized in the main function while they are being created.

Type the following lines of code before the void welcome; statement in the main function:

```
int matchTotal = 21; // initializing matchTotal for start of game
string whoseTurn = "Player 1"; // initializing whoseTurn for the first turn
int turnAmount = 0; // initializing matches taken on each turn
```

The Walk Through: Taking turns function...

Since each turn will be basically the same, except for whose turn it is and the total available matches in the pile, it makes sense to make a function out of it. This function will be called each time through the game loop in the main function.

Type the following line of code after the void welcome(); statement in the function prototype section before the main function:

```
int takeTurn(int matchTotal, string whoseTurn);
```

int means there is an integer returned to the main program when the takeTurn function finishes. This integer is the number of matches taken on the turn. This value will then be subtracted from the matchTotal variable in the main game loop. This is how the main game loop knows when to break out of looping. The function needs to know what the current matchTotal is so it can be displayed to the next player. The function also needs to know whose turn it is so that it can be displayed on the screen to keep the players on track with whose turn it is to take matches.

This function needs to display the current match total, display whose turn it is and get the amount of matches taken for that turn. It would also be nice if the function only accepted 1, 2 or 3 as the correct values. A do while loop in the function that checks for this would be great.

At the end of the program, type in the takeTurn function:

```

// takeTurn function displays current number of matches in the pile,
// displays whose turn it is and gets a number of matches taken
// by the user for that turn. This amount is passed back to the main function.
int takeTurn(int matchTotal, string whoseTurn)
{
    int howMany = 0; // initializes turn taking amount to be passed back

    // the loop looks for an acceptable amount of matches to be taken
    do
    {
        cout << "\n" << whoseTurn << ", there are " << matchTotal;
        cout << " matches in the pile.\nHow many do you want? ";
        cout << " (1, 2 or 3): ";
        cin >> howMany;
    } while ((howMany != 1) && (howMany != 2) && (howMany != 3));
    // logical and used in do while condition to test user response

    return howMany; // sends the amount of matches taken on the turn back
} // end of takeTurn function

```

The Walk Through: Testing the takeTurn function...

We aren't technically ready to use this function in the program yet because it gets called from the main game loop which isn't written yet. But it is a good idea to test the function to make sure each function is working the way you want it to as you go along through the program. To do this, a test line of code can be added to the main function. This line of code will be deleted once the function is working fine.

Type the following line of code after the welcome function is called in the main function:

```
turnAmount = takeTurn(matchTotal, whoseTurn); // testing function
```

Save, compile and run the program to see if the functions are working. Choose Compile and Run from the Execute menu.

Here is a sample output of the program so far:

Imagine a pile of 21 matches.
There are two human players.
Each player takes one, two or three matches on their turn.
The player who takes the last match loses.

Player 1, there are 21 matches in the pile.
How many do you want? (1, 2 or 3): 5

Player 1, there are 21 matches in the pile.
How many do you want? (1, 2 or 3): 3
Press any key to continue . . .

Notice that the 5 was used to test if the program would stay in the loop or fall out of the loop. Once an acceptable choice was given, then the program ended because we didn't ask it to do anything else yet. The function returned the integer 3 back to the main function and did nothing with it. The test worked and we can delete the last line of code added. We now know the function is working.

Delete the following line of code from the main function:

```
turnAmount = takeTurn(matchTotal, whoseTurn); // testing function
```

The Walk Through: The main game loop...

This program will have several turns taken by the two users. The program must loop through these turns until the game is over. A playGame function can be used to contain the main game loop. The matchTotal and whoseTurn variables will need to be passed to it and it will need to return the winning player of the game. This function will have to take care of changing the total amount of matches to determine when the game is over as well as changing whose turn it is for the next turn. The takeTurn function will be called from within the playGame function.

Type the following line of code after the int takeTurn(int matchTotal, string whoseTurn); line of code in the function prototypes section above the main function:

```
string playGame(int matchTotal, string whoseTurn);
```

Type the following code at the end of the program:

```
// playGame function takes care of the main game loop of the game.
// It will stay in the loop until the game is won (matchTotal is 0).
// The function is passed the matchTotal and whoseTurn variables
// and returns the winning player back to the main loop.
string playGame(int matchTotal, string whoseTurn)
{
    int turnAmount = 0; // initializing amount taken on each turn

// the main game loop
do
{
    // start play
    turnAmount = takeTurn(matchTotal, whoseTurn); // take one turn

    // change total of matches in the pile
    matchTotal = matchTotal - turnAmount;

    // change which player's turn it is
    if (whoseTurn == "Player 1")
        whoseTurn = "Player 2";

    else
        whoseTurn = "Player 1";
} while (matchTotal > 0); // allows for negative pile total to also end game

return whoseTurn; // returning winner of game
} // end of playGame function
```

Now the new function needs to be called from the main function. Type the following code into the main function after the welcome function has been called:

```
whoseTurn = playGame(matchTotal, whoseTurn); // Calls playGame
```

Test to see if the program is working so far. Choose Compile and Run from the Execute menu.

The Walk Through: Displaying the winner of the game...

There is only one function left to write. It's the function to tell the users who won the game. The function will be a void function like the welcome function because it will not need to return anything to the main program. It will need to have the whoseTurn variable passed to it so it can display the winner.

Type the following code after the string playGame(int matchTotal, string whoseTurn); statement in the function prototypes before the main function:

```
void endOfGame(string whoseTurn);
```

At the end of the program, type the following code:

```
// endOfGame displays the winner of the game.
// The winning player is passed to the function.
void endOfGame(string whoseTurn)
{
    cout << "\n\nEnd of Game!\n\n";
    cout << whoseTurn << " is the winner!\n\n";
} // end of endOfGame function
```

Please note that in the playGame function the last person to take a turn was the loser, but after that person took a turn, the whoseTurn was changed to the other player (the winner). This makes it easy to display the winner without changing the value of the whoseTurn variable.

Now the function needs to be called from the main function. Type the following code after the whoseTurn = playGame(matchTotal, whoseTurn); statement in the main function:

```
endOfGame(whoseTurn); // calls the endOfGame function
```

Please note that the turnAmount variable did not end up being used in the main function. It was reinitialized as a local variable inside of the playGame

function. Since this line of code isn't needed in the main function, it has been removed from the listing of the final code.

The program is finished. Here is the code:

```
// This program is a game with two human users. 21 matches are in a
// pile and players take turns taking 1, 2 or 3 matches. The program
// loops until the matches are gone and announces who won.
// Written by Janine Bouyssounouse on 10/19/08

#include <iostream>
#include <stdlib.h>

using namespace std;
void welcome();
int takeTurn(int matchTotal, string whoseTurn);
string playGame(int matchTotal, string whoseTurn);
void endOfGame(string whoseTurn);

int main(int argc, char *argv[])
{
    int matchTotal = 21; // initializing matchTotal for start of game
    string whoseTurn = "Player 1"; // initializing whoseTurn for the first turn

    welcome(); // This calls the welcome function
    whoseTurn = playGame(matchTotal, whoseTurn); // Calls playGame
    endOfGame(whoseTurn); // calls the endOfGame function

    system("PAUSE");
    return 0;
} // end main

// welcome function displays an opening message to
// explain the program to the user
void welcome()
{
    cout << "Imagine a pile of 21 matches.\n";
    cout << "There are two human players.\n";
    cout << "Each player takes one, two or three matches on their turn.\n";
    cout << "The player who takes the last match loses.\n\n";
}
```

```

} // end of welcome function

// takeTurn function displays current number of matches in the pile,
// displays whose turn it is and gets a number of matches taken
// by the user for that turn. This amount is passed back to the main function.
int takeTurn(int matchTotal, string whoseTurn)
{
    int howMany = 0; // initializes turn taking amount to be passed back

    // the loop looks for an acceptable amount of matches to be taken
    do
    {
        cout << "\n" << whoseTurn << ", there are " << matchTotal;
        cout << " matches in the pile.\nHow many do you want? ";
        cout << " (1, 2 or 3): ";
        cin >> howMany;
    } while ((howMany != 1) && (howMany != 2) && (howMany != 3));
        // logical and used in do while condition to test user response

    return howMany; // sends the amount of matches taken on the turn back
} // end of takeTurn function

// playGame function takes care of the main game loop of the game.
// It will stay in the loop until the game is won (matchTotal is 0).
// The function is passed the matchTotal and whoseTurn variables
// and returns the winning player back to the main loop.
string playGame(int matchTotal, string whoseTurn)
{
    int turnAmount = 0; // initializing amount taken on each turn

    // the main game loop
    do
    {
        // start play
        turnAmount = takeTurn(matchTotal, whoseTurn); // take one turn

        // change total of matches in the pile
        matchTotal = matchTotal - turnAmount;

        // change which player's turn it is

```

```

    if (whoseTurn == "Player 1")
        whoseTurn = "Player 2";

    else
        whoseTurn = "Player 1";
} while (matchTotal > 0); // allows for negative pile total to also end game

return whoseTurn; // returning winner of game
} // end of playGame function

// endOfGame displays the winner of the game.
// The winning player is passed to the function.
void endOfGame(string whoseTurn)
{
    cout << "\n\nEnd of Game!\n\n";
    cout << whoseTurn << " is the winner!\n\n";
} // end of endOfGame function

```

Save, compile and run the program to see if it works. Choose Compile and Run from the Execute menu.

Here is a display of the program:

```

Imagine a pile of 21 matches.
There are two human players.
Each player takes one, two or three matches on their turn.
The player who takes the last match loses.

Player 1, there are 21 matches in the pile.
How many do you want? (1, 2 or 3): 5

Player 1, there are 21 matches in the pile.
How many do you want? (1, 2 or 3): 1

Player 2, there are 20 matches in the pile.
How many do you want? (1, 2 or 3): 1

Player 1, there are 19 matches in the pile.

```

How many do you want? (1, 2 or 3): 3

Player 2, there are 16 matches in the pile.

How many do you want? (1, 2 or 3): 1

Player 1, there are 15 matches in the pile.

How many do you want? (1, 2 or 3): 3

Player 2, there are 12 matches in the pile.

How many do you want? (1, 2 or 3): 2

Player 1, there are 10 matches in the pile.

How many do you want? (1, 2 or 3): 1

Player 2, there are 9 matches in the pile.

How many do you want? (1, 2 or 3): 3

Player 1, there are 6 matches in the pile.

How many do you want? (1, 2 or 3): 1

Player 2, there are 5 matches in the pile.

How many do you want? (1, 2 or 3): 1

Player 1, there are 4 matches in the pile.

How many do you want? (1, 2 or 3): 3

Player 2, there are 1 matches in the pile.

How many do you want? (1, 2 or 3): 1

End of Game!

Player 1 is the winner!

Press any key to continue . . .

Enjoy playing. Try to find a strategy that works to win the game.