C Programming

Baby Steps

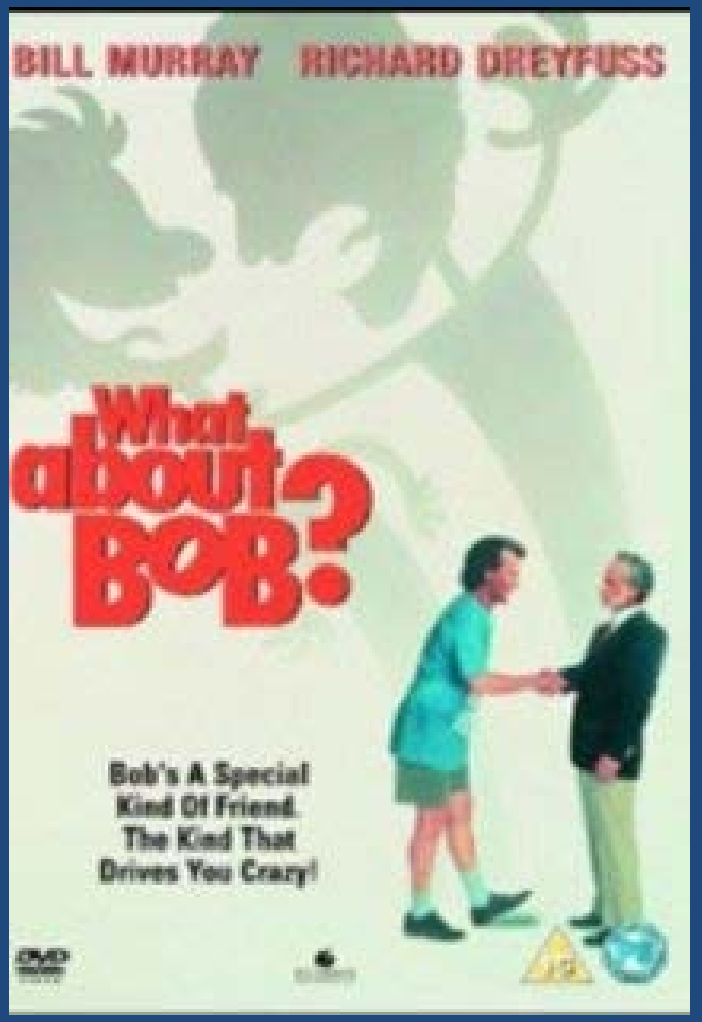
Another Design Method

Why?

It's an alternative to pseudocode

What is it?

Let
Bill Murray
and
Richard Dreyfuss
tell you



Bottom Line:

Do something that you can do,

then add to it

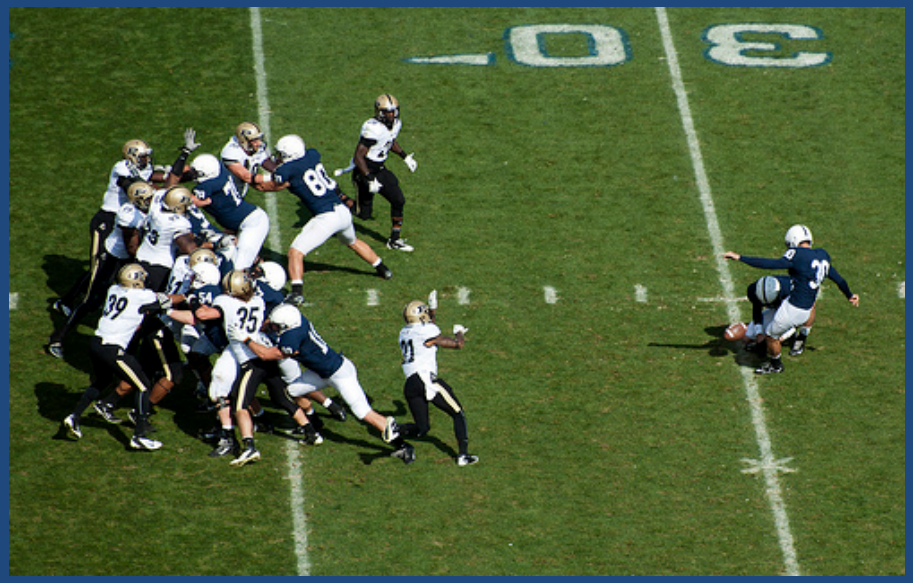
Uses

"Do I know how to do this easily?"

with code from scratch

From the movie:

"Setting small, reasonable goals for yourself, one day at a time, one tiny step at a time"



"All I have to do is take one little step at a time and I can do anything"

Applying Baby Steps

1. Do what you know how to do
2. Get that working
3. Add something to that program
4. Get that working
5. Etc.

We're going to do
some examples,
but first ...

Getting numbers from a user

Assignment #2 (and #1a) has a function that can be used to get an int from a user

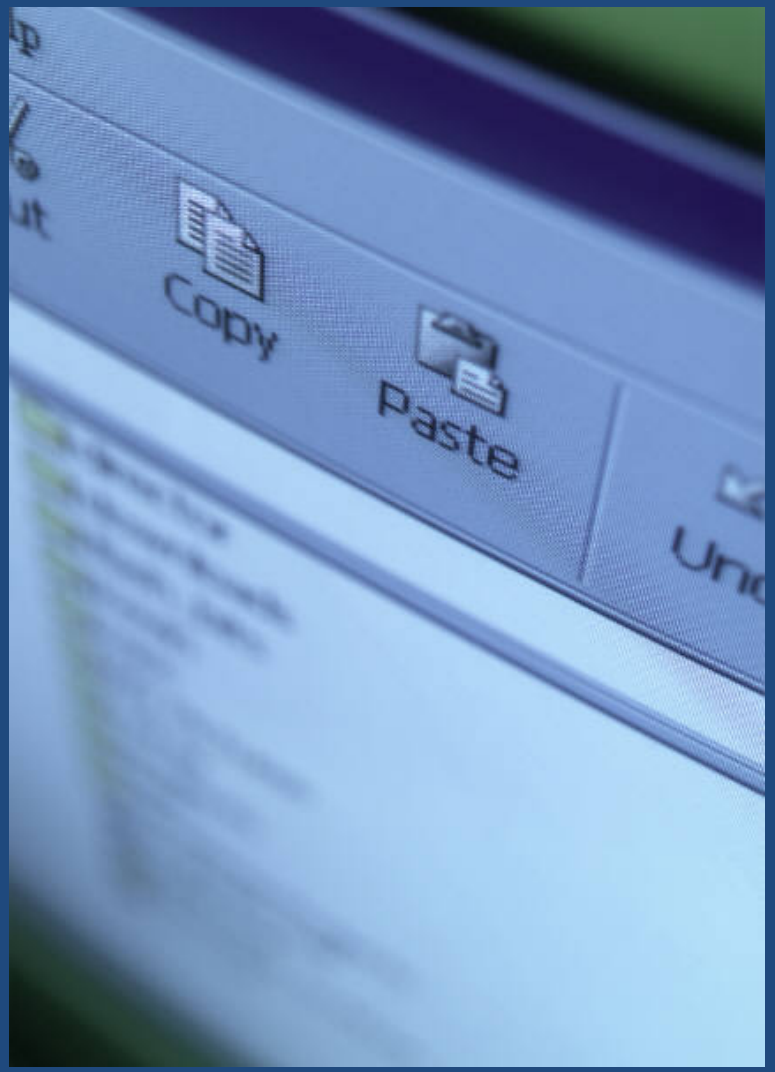
It's called getNum()

We'll go over it
in more detail
in another class

For now, use

variable = getNum();

You must also copy-and-paste the code from the Assignment into your source code (and reformat it).



Example

Given a series of numbers entered by a user, count the number of occurrences of negative numbers. Stop when the user enters 0.

Put each step in a main() function like in the first example in the first week.

Start with something easy:

- Write a program that gets a single number from the user and displays it back to them.
  1. Get a number from the user.
  2. Display that number.
- Turn that into code:
  - int number = getNum(); // from assignment #2
  - printf("%d\n", number);

Once you have that, add to it:

- Write a program that gets an infinite number of numbers from the user and displays them back to them.

1. While condition is true
    1. Get a number from the user.
    2. Display that number.

- Turn that into code:

```
int  number = 0;
while(  1 )         // remember, true is any non-zero value
{
        number = getNum(); // from assignment #2
        printf("%d\n", number);
}
```

Once you have that, add to it:

- Write a program that gets ten numbers from the user and displays them back to them.

1. Initialize counter to 0
2. While counter is less than 10
    1. Get a number from the user.
    2. Display that number.
    3. Increment counter.

- Turn that into code:

```
int counter = 0, number = 0;
while(  counter < 10 )
{
        number = getNum(); // from assignment #2
        printf("%d\n", number);
        counter++;
}
```

Once you have that, add to it:

- Write a program that gets ten numbers from the user and displays only the negative numbers.

1. Initialize counter to 0
2. While counter is less than 10
    1. Get a number from the user.
    2. If the number is negative, then
        1. Display that number.
    3. Increment counter.

- Turn that into code:

```
int counter = 0, number = 0;
while(  counter < 10 )
{
        number = getNum(); // from assignment #2
        if( number < 0 )
        {
                printf("%d\n", number);
        }
        counter++;
}
```

Once you have that, add to it:

- Write a program that gets ten numbers from the user and counts the negative numbers.

1. Initialize counter to 0 and negativeCounter to 0.
2. While counter is less than 10
    1. Get a number from the user.
    2. If the number is negative, then
        1. Increment the negativeCounter.
    3. Increment counter.
3. Display negativeCounter.

- Turn that into code:

```
int counter = 0, number = 0, negativeCounter  = 0;
while(  counter < 10 )
{
        number = getNum(); // from assignment #2
        if( number < 0 )
        {
                negativeCounter++;
        }
        counter++;
}
printf("negative count is %d\n", negativeCounter);
```

Once you have that, add to it:

- Write a program that gets numbers from the user (until they enter 0) and counts the negative numbers.
  1. Initialize counter to 0 and negativeCounter to 0.
  2. Initialize number to something that is not 0 (so the loop will start).
  3. While number is not 0
     1. Get a number from the user.
     2. If the number is negative, then
        1. Increment the negativeCounter.
  4. Display negativeCounter.

- Turn that into code:

```
int number = -1, negativeCounter  = 0;
while(  number  !=  0)
{
        number = getNum(); // from assignment #2
        if( number < 0 )
        {
                negativeCounter++;
        }
}
printf("negative count is %d\n", negativeCounter);
```

Problem solved!

Walkthrough Exercise

Design a program to repeatedly take in (from the user) odometer readings (in km) and gas consumption (in litres) values and display the litres per 100 km calculation applicable since the last fillup.

When the user enters gas consumption of 0, the program will end.

Assumption #1: the user will enter positive floating point numbers for odometer readings and gas consumption (until 0 is entered to exit).

Assumption #2: the user will always enter an odometer value that is greater than the previous one.

Assumption #3: all gas consumption figures assume that the tank is filled and that any error due to differences between gas pump "full“ sensors is OK.

Solution

Step 0: Examine the statement of the problem, noting the word "repeatedly" and user input.

Step 1: Come up with a project that compiles but does nothing.

Step 2: Display a message on the screen prompting the user for odometer.

Step 3: Use the getNum() function from assignment #2 to get an integer for odometer from the user. Display it.

Step 4: Change getNum() to getFloat() function to get a float for odometer from the user. Display it.

Since we haven't looked at sscanf() yet, it is OK if you don't feel comfortable with this step. If that's the case, continue with the solution using int variables.

Step 5: Prompt the user for gas consumption.

Step 6: Use getFloat() to get a float for gas consumption from the user. Display it.

Step 7: Use an infinite loop to get and display the two values repeatedly.

Step 8: Change the loop to end when the user enters 0 for the odometer.

Thinking Steps:

- Step 9a: Think about, but do not code, how you would calculate the litres per 100 km.
- Step 9b: Conclude that litres per 100 km is 100 times litres per km.
- Step 9c: Conclude that the litres per km is consumption divided by distance.
- Step 9d: Conclude that you need to calculate distance.
- Step 9e: Conclude that distance is the current odometer reading minus the previous one.

Step 9: Prompt the user for the previous odometer reading and display it.

Step 10: Calculate and display the distance.

Step 11: Calculate and display the consumption divided by distance.

Thinking Steps:

- Step 12a: Realize that you had the previous odometer reading the last time through the loop.
- Step 12b: Realize that the previous odometer reading is still in the odometer variable until you get the next one.

Step 12: Create a variable that will store the previous odometer value before you get the next one and display it.

Step 13: Calculate and display distance with your new variable.

Thinking Step:

- Step 14a: Realize that this now gives bad calculation values since the calculation shouldn't be done when exiting.

Step 14: Add an if statement that skips the calculation part if you're about to leave the program.

Thinking steps:

- Step 15a: Realize that this doesn't give the right values the first time through the loop.
- Step 15b: Realize that this means that you need a starting odometer value before entering the loop.

Step 15: Prompt the user for the very first odometer reading and display it.

Thinking Step:

- Step 16a: Realize that we're still displaying litres per km instead of litres per 100 km.

Step 16: Multiply litres per km calculation by 100.

Thinking Step:

- Look over the statement of the problem and make sure that you've satisfied the requirements.
  - Nope! We're exiting on an odometer value of 0 instead of a consumption value of 0. Make changes and test it.

Problem solved!

Summary

1. Baby Steps can be used as a good way of doing program design.