General Problem Solving Strategy

There are likely to be two kinds of problems (generally, obviously there could be more kinds) 1) Problems that read data incrementally (read a little, write a little, read a little, write a little), and 2) Problems that read everything in and then process the data and write everything out. That intermediate class would read a block of data in, and write a block of data out. Call these bit by bit, block by block, and everything in and everything out types.

Figure out what kind of problem it is.

Figure out the algorithm that will solve it. Generally you should be able to sketch it out on paper and work through a simple example on paper.

Write pseudocode as comments that describe your solution procedure. This is easily done as
//Step1: … what you do in step 1
//Step2: … what you do in step 2 etc.

Generic steps will be things like:
Put you include files in
Declare your variables
Stub Out your main() and any functions that you think you need. Put the pseudocode into your stubs.

Now incrementally build out your functions compiling and running them incrementally.

To run the program build some test files using the procedure:

cat > filename.txt
lines of file …
^D …that's a control D which is eof (end of file)

Write you file as filename.cpp
Compile as **g++ -o fileout filename.cpp** and then run it with your filename.txt input file As:
fileout < filename.txt If you want to send your output to a file you can do that by typing

fileout < filename.txt > filenameout.txt

Obviously you can name the input and output files anything you want, but typically they will be text files. Only after being confident that you've solved the problem should you submit it.

Programming On-the-Fly

Input —> Process —> Output is the flow of information in most programs.

Essentially this means:
1) Read some information
2) Do some processing, and
3) Output some information

The exact flow will depend on the application or problem being solved.
Programs are put together from pieces. The father of Pascal, Niklaus Wirth, wrote a programming book titled: Algorithms + Data Structures = Programs and that's about the size of it.

When you want to build programs fast you need a process that works well. Typically you need a set of pre-fabs and a quick way to put them together. I'll call the pre-fabs, Templates, and the way to put them together pseudocode.

Top Level Prefab is the **PROGRAM STUB**

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

//PROTOTYPES AS NECESSARY

int main(){

// DATA DECLARATIONS
// INPUT OPERATIONS
// PROCESSING OPERATIONS
// LOOP AS NECESSARY

return 0;
}

//PROTOTYPE EXPANSIONS AS NECESSARY
```

DATA DECLARATIONS
Study the problem and make a list of the data variables you will need and the kind.

Most of the time you will be using standard input and standard output for your input and output so you should use cin and cout in C++ as the easiest approach and use IO redirection to test your program. What's that? Well you have to find out.

http://en.wikipedia.org/wiki/Redirection_(Unix)#Redirecting_standard_input_and_standard_output

`command < inputFile > outputFile` this will cause the inputFile named to be read as if it is the keyboard and any output will be directed to the outputFile.

So the obvious thing to do is to use `cin` and `cout`, but the >> and << operators don't usually read a character at a time so it is helpful to have some other tools for input in your toolkit.

Appendix D in Nyhoff discusses input and output stream operations . Also you can find information in Overland's *C++ Without Fear*. (172-191) for string input.

Whole Entity Input
If you're just trying to read in whole entities, such as numbers in isolation then `cin >>` will probably work just fine.

Character by Character Input
In instead you want to read in character by character because you're supposed to do things with the characters then you have to kinds of options. You can actually read in character by character using get or using getline you can read in an entire line as a string and then work through the line character by character. The latter is often the better option, but it can vary from problem to problem.

http://www.cplusplus.com/reference/iostream/istream/get.html
http://www.cplusplus.com/reference/iostream/istream/
http://www.cplusplus.com/reference/iostream/istream/getline.html

Do Some Processing
Once you've sorted out the data structures you will need and figure out how you're going to handle input in general, you have to figure out what kind of processing you're going to do. One of the most direct and easy ways to do that is to use *PSEUDOCODE*.

Ok, what is pseudocode?
The short answer is that it is a set of directions you write for solving the problem but unconstrained by any particular computer language. You write the program solution in your own way, in your own language. It may be helpful if the language you pick is close to the language you're going to implement the program in, but that isn't really necessary.

One way is just to write a series of numbered steps using indentation for numbering substeps. So let's look at an example for that. We'll use a simple problem used as a warm-up problem at the CCSCE 2006 contest.

PROBLEM STATEMENT
Write a program that reads in a positive integer value and prints out one row of stars equal in length to each digit of the input number, starting from the least significant digit. For example, if the input number is 2078, then the output would look like:

****
****

**

INPUT & OUTPUT REQUIREMENTS

Your program should first read in an integer indicating the number of test cases to be executed. Then the program should read in an integer for each test case and output the row(s) of stars. After all of your test case result are output, output the word "DONE" on the last line of output. All output must be left-justified and include no additional spaces.

SAMPLE INPUT

```
3
1234
970
101505
```

SAMPLE OUTPUT

```
****
***
**
*

*******
*********
*****

*****
*

*
DONE
```

ANALYSIS

DATA

We need the following data (or we could in some cases expect to implement an element either procedurally or with a data element. For example we could create strings of asterisks of lengths 1 to 9 to handle the asterisks, or we could simply have a loop which outputs the necessary number of asterisks.

PSEUDOCODE

Now we need to solve the problem by writing our pseudocode.

1. Read in the number of test cases, ex. cin>>numcases;
2. For the number of test cases
   2.1 Read in the test case
   2.2 From Least Significant Digit to Most Print Asterisks
3. When finished output "DONE"

So what's tricky here. Probably 2.2. There are two obvious ways to do 2.2. The first is to use the modulus operator and the integer divide to progress through the number. The second way is a bit different, that's to read the number in as a string and then read the characters out from the end of the string to the front. That might be a little trickier. Now HAVE AT IT!