Creating a program -- A 4-step process.

Before you can attempt to compile a program, you must first create it. This section discusses the 4-steps required to get a program up-and-running. Of course, this all

assumes that you are starting off with a good design!(*see section on algorithms*)

Step 1: Create (edit) the source code.

During this phase you will enter your source code. Depending on the development environment you are using, you must start up some sort of text-editor so you can enter your code. An editor is an application program, similar to a word-processor, except the information (code) is stored in text (ascii) format, with no additional graphics. With the Turbo environments, once you start the Turbo program, you are automatically put into a text processing window. With UNIX, you must start up a text editor yourself (such as vi, or emacs).

Once you have your text editor up-and-running, you can start to type in your code. Some text editors are specifically setup for you to enter C code (such as Turbo C and MS Visual C). These editors can be very helpful as you are typing in your code, as they give color-coded words and segments of code so you can more easily detect a "syntax error". (More on this later.)

Once you have entered all of you code, you must save your file on disk, giving it a name. It is a C standard to give your "source" file a descriptive name, and end it with a ".c" extension. That is, if you want to call your program "fun", you should name your source file: **fun.c**. (Your compiler may want to 'automatically' place **.cpp** at the end, please *save* your file *as* **fun.c** with a **.c** extension so that it will compile in standard ANSI C and not in C++, which is how .cpp programs compile)

Now, you have your "source code", in your source file (**fun.c**), you are ready to "compile" your program, which is step 2 of the process, and described next.

Step 2: Compile your source file.

Recall, the computer only understands a series of 1's and 0's. However, our source file contains "english-like" instructions in it. So, we must convert the source code into "binary", or machine code. Luckily, there is a software program written to do this for us, otherwise we would need another (huge) cup of coffee. (Or perhaps at this point we would grab the vodka! :-)) The program that is available to us to convert our high-level code to machine code is called the: ?????????? Yes. You guessed it. It is called the compiler! A **compiler** is simply an **application program which converts high-level code to a particular computer's machine code**. So, compilers are machine (computer) dependant. That is, each computer (with associated operating system running on it) has its own compiler. So, if you have a microcomputer running MS-Windows, then you will buy a compiler which runs on MS-Windows. If you have a

mainframe computer running UNIX, then you need a compiler which runs on UNIX. If you have a Mac computer, then you need to purchase a compiler that runs on a Mac.

How to start up your compiler will again vary depending on which compiler you are using! For example, with the Turbo C compiler and many others, you simply can click on the "COMPILE" menu. If it compiles with no errors it can then be run(executed). This is also usually just a menu pick - RUN (or Execute!). (You may need to create project space such as in MS Visual Studio)

In C-Free simply select the ***"Build"*** tab, and from the drop down menu select ***"Run".*** The compiler will compile, link, and execute the program. If compile errors are encountered there will be errors in red highlights below the code window. Your cannot always tell what your error is from the error listing. You will have to go back and correct the errors, and try again. That's it! You may read the rest of this section for what will happen in other typical compilers. Or jump down to "Step 4…"

After the compiler is invoked (started), and it will convert your source code, to machine code. (AKA - binary code, or object code.)

The compiler actually does the following:

1. Invokes (starts) the "preprocessor" (another program which checks a few things - more later).
2. Checks for errors. -- If any of the source code does not follow ANSI C Standards (remember those?), the compiler will alert you that there is an error. This type of error is known as a "**compiler error**" or "**syntax error**". At this point, you must go back to step 1 and fix any of the problems that you might have. Typically the compiler is pretty good at giving you a hint as to where your problems lie. After you make your changes, you must re-compile your source code to see if there are any additional syntax errors. Don't be surprised if you start off with only 3 syntax errors, fix those, and when you recompile, you end up with 10 different syntax errors! This is a normal process in the computer programming world!
3. Once the compiler no longer detects any syntax errors, it will then go ahead and convert your source code to object code (AKA: binary, or machine -- they all mean the same thing). Typically a file with a ".o" or ".obj" extension is created. So, in our example, after compiling file **fun.c**, you would end up with a file called fun.o (or **fun.obj**). This object code is not yet executable. That is, it is not yet in its final form for execution. Now the "linker" program must be called to put together (link) multiple files that make up your program. In most cases, you cannot put all of your source code into 1 file because normally, programs are very large, and your source file would be huge and unmanageable. So, we split the program into parts and each programmer gets a portion of the program to work on and will create his/her own source files. Each source file is compiled separately, and then "linked" when they are all complete. This brings us to the

next step in the process, which is to "link" all of the object files together. (We will briefly discuss "compiling larger programs" in a later chapter.) ***Warning!*** You may still have runtime problems with your code. Certain compilers are notorious for ignoring sections of code it does not understand. You must read the compiler *warnings* as well as the *error messages*.

4. **Step 3 - Link your program.**

Normally the linker is invoked (started) automatically by the compiler. Again, the linker, like the compiler (and the editor) is simply an application program which creates an executable file. You can choose whether or not to automatically invoke the linker after compilation.

Now that I've completely confused you...

Please do not get too stressed out about this particular phase. It will make more sense as we go along. What I want you to understand at this point is not necessarily HOW the linker is invoked, but WHAT happens when the linker is invoked. Once the linker is invoked, it checks if there are other program files that need to be combined with your program file to make a completed executable file. Once all of the "pieces of the puzzle" (so to speak) are located by the linker, the linker will then make another binary file called the "executable" file. This is the program file that is executed (started) so that you can see your results.

Typically, an executable file will be the same name as the source (and object) file, with an extension of ".exe". So, in our example, the linker would create the file: **fun.exe** (This is the default executable filename if you are using a Windows based compiler/linker like C-Free).

Like with the compiler, you may get errors during this link phase. The types of errors you might get are different than "compiler" or "syntax" errors that you get during compilation. The types of errors you might get during the link phase are normally "undefined" functions. In this case, you need to go back to Step 1 and fix your problem. We will discuss functions in more detail when we get to chapter 8. Another type of "**linker error**" is one that is caused as a result of not having enough hard-disk space for the linker to store the executable file. This should not be the case in our class, as our programs will be relatively small. However, think about the size of some of the MS-Windows based application programs, for example. MS-Word 2000 is 8.6 MB (megabytes)! That is a hefty sized program. So, as you can see, the linker may have problems trying to fit the executable file on a crowded hard-drive. Once the linker successfully creates an executable file, you can now go ahead and attempt to execute (run) it!

Step 4 - Execute (Run) the Program.

"HOW" you run your program depends once again on the operating system you are using. If you are using a Windows-based environment, you would double-click on the executable filename. However, if you are still within the C-Free environment, you can just click on the "Build" menu, select “Run”, and your program will run.

Once your figure out how to actually run the program, the executable file created by the linker is copied from hard-disk to **RAM** (Random Access Memory). The **CPU** (Central Processing Unit -- which executes your program 1 instruction at a time) wants to run the program from memory, instead of from the hard-drive because it is much faster if the program resides in RAM.

FYI: The hard drive is a mechanical device. Memory and the CPU are electronic devices (integrated circuit chips). So, if the CPU had to run the program from the hard-drive, it would be slowed down by the mechanics of the hard drive. If the program is instead copied into RAM, the CPU can run at its most efficient pace, since both the CPU and RAM are electronic devices.

If all goes well, (and it probably WON'T the first time you run your newly created program), the program should run to completion, presenting you with the desired results. However, as in the last 2 phases, things can go wrong during program execution. ***There are 3 types of errors that occur during run-time:***

1. **Fatal error** -- This occurs when your program "crashes" during runtime. You know this has happened when you see some sort of nasty message on your screen from the operating system such as: 7FFE 12A4 Core Dump. Don't panic. You simply go back to step 1 and try to figure out what went wrong. (Sometimes you may actually have to reboot your computer!)
2. **Infinite Loop** -- This occurs when your program "hangs" or gets stuck at a particular segment of code. You know this is happening when your program does not ever finish. This typically happens when you incorrectly use one of the repetition structures (loops). Sometimes your loop is generating output, and you will see lots and lots of text scrolling on your screen. Some of the time you will need to re-boot your computer and return to step 1.
3. **Logic error** -- This occurs when your program runs to completion, however your results are incorrect. For example, if you wrote a program that prompts the user for 2 numbers, and the program then calculates and displays the "sum" of the two numbers, and if you ran your program and the user typed 15 and 20, and your program displayed 55 for the sum, then you know you have a logic error. (Was that a run-on-sentence or what?!?!) These errors are sometimes difficult to find if the program logic is complex. This is the type of error you want to resolve during the "design" phase (hint, hint).

Now, throughout all of these steps, you will be making lots of mistakes. Not that I don't have faith in you, but I speak from my very own experience. Just know that the BEST thing you can do is make lots of mistakes. ***Yes!*** I have not lost my mind. The reason I say this is because when you make a mistake, you will troubleshoot your program and

find the solution to the problem. This process (known as "debugging" your code) is where you actually learn how to program. You will quickly become a good troubleshooter, easily fixing your own, and even other people's code. Because, once you make a mistake, you rarely forget how to fix that particular mistake. So, when you repeat that mistake, or if someone else has made the same mistake, you will know how

to fix it. Trust me.

In Summary...

4-Steps to creating programs:

1. Create (edit) your source code ---> fun.c
2. Compile your source code ---> fun.o
3. Link your object code ---> fun.exe (usually links automatically after compile)
4. Run your program (steps 2,3,4 happen as one single step in C-Free if there are no compiler errors)

Types of errors:

- Compilation (or Syntax) Error (.exe file is not created)
- Linker Error (.exe file is not created)
    - Undefined functions.
    - Insufficient disk space.
- Runtime Error (occurs when you run the program and it fails to execute/executes incorrectly)
    - Fatal error.
    - Infinite Loop.
    - Logic error.

Please also note any *warnings* that are generated - these may be indicative of possible errors you will encounter when you run the program. Also, always remember to *save* your program source. If you do get a run-time error, in some cases, you may have to re-boot your computer. If you haven't saved the source - it may be gone after re-booting.

What you will submit to me is the ***fun.c*** source file **only!** (**not** the *fun.exe* file, or any other files created by the system you use.) I will then compile and run the program on my system.

NEXT WEEK: -- Now that you (hopefully) understand some of the basic fundamental concepts of computers and programming, we will be pressing on and typing in our first program!