Common C/C++ Errors

The most common errors can be broadly classified as follows

1. Programming errors
These are generated when typographical errors are made by users.

2. Compiler errors
These are errors detected by the compiler that make the program un-compilable.

3. Linker error
These are errors generated when the executable of the program cannot be generated. This may be due to wrong function prototyping, incorrect header files.

4. Execution error
These errors occur at the time of execution. Looping and arithmetic errors falls under this category.

5. Logical errors
These errors solely depend on the logical thinking of the programmer and are easy to detect if we follow the line of execution and determine why the program takes that path of execution.

Listed below are some common programming errors
1. Misuse of the Include Guard.
A common mistake is to use same symbol in multiple files for #ifndef.
2. Typo's : Using ">" for ">>" and "<" for "<<"
3. Trying to directly access the private variables in the main program.
4. Switch Statments without break.
5. Wrong usage of postfix and prefix operator (diff between i++ and ++i)
6. Undeclared and Unitialised Variables and functions.
7. ALWAYS USE MAKE FILE if you have more than one C++ program. The order of
compilations matters a lot too.
8. Trying to include "INCORRECT" header fuction.
9. Marking a member function as const in the class definition but not in the member function implementation.
10. Returning a value in a void function.
11. Confusing the name of an array with the contents of the first element.
12. Cstring array Errors - Arr[10] = Arr[0] to Arr[9]. Trying to access Arr[10] element.
13. Using "=" (assignment operator) instead of "= =" (comparison operators)
`scanf()` without '&' and wrong format.(IN C)
14. Trying to divide a number by Zero.
15. Poor Loop Exiting comparisons will tend to either loop being not executed at all or goes to an infinite loop.

16. Not using string functions and treating the strings are integer . Say trying to compare string by (string1= = string2) , rather than using `strcmp` command
17. CString not terminated by '\0'- Null character
18. Mismatched "{" or IF-ELSE statements or for that matter any looping statment.
19.Namespace errors

Some times the smallest errors are the hardest to find. One way to avoid these kinds of error is Systematic programming. Always follow one set of conventions. Say starting from naming the variables (using upper case to denote `CONST`) to order in which you write the program (START WITH HEADER FILE -function declaration . Then TEMPLATE -function definition- Then proceed to -function call at the calling function).
The following are the set of programming conventions given by Dr. Chelberg. Read the document. Proper systematic programming will save you hours of debugging.

Style Guidelines for programming -Dr. Chelberg

The nightmare for the students who are programming in C/C++ are the real time experiences with terms like Core Dumps, Bus error and Segmentation Fault. But these are generally the easiest errors to identify and debug.

1. Core Dumps

In the Unix environment, if the program that is currently running has certain errors, then it dumps core. “Core” is nothing but the entire state of the program at the time of failure. These files are the true state of the program that are extremely useful in identifying where the program actually went wrong. We can use the debugger on these core files to detect errors.

2. Running Out of Disk Space

Core files can be very big, and can quickly wipe out quota of disk space alloted to the user. This can block writing other files, for example when compiling a program. If the user sees a file named “core” lying around, it should usually just be deleted. This frees up storage space. The temporary files “Emacs” creates could also be deleted too. Those are the files that end with "~" (Eg. File.c~) or starts with '#' (Eg. #file.c#).

3. Segmentation Fault and Bus Errors

If the program displays these messages, it means is the program was trying to access a memory location outside its address space. The computer detects this and sends the error message back to the user. They are mainly due to out-of-bounds array references or accessing null pointer or unintialized pointers.

Error Detection Techiniques

1. Printf /cout statements

This is the easiest method to detect these error. Identify the point of the program or the sub-routine it fails, by giving a cout statement. It is always easier to debug the program once the point at which the crash occurs is known.

2. Use debugger -GDB :

It allows you to see the statr of the program when it is being executed--or what the program was performing at the moment it crashed. To use this effectively, you have to compile the program with `-g` option. This option causes the compiler to produce symbol table information that gdb needs.

USAGE :

```
<machine>% gdb file_name.c core
```

On more information on this debugger,look at the man page of GDB

```
<machine>% man GDB
```

(or)

http://sources.redhat.com/gdb/onlinedocs/gdb.html

PREVIOUS HOME NEXT