Command line utility library for C++

By John Lawson

Version 1.11 January 2014

Introduction

Cmd– is an open-source utility library for C++, created to streamline the creation of console-type programs with multiple menus, functions, and purposes. The library itself currently consists only of standard C++ header and source files, and can be compiled on any modern compiler supporting the C++ language.

Cmd– was born out of frustration with constantly writing command line menu structures every time I wanted to create a program. It now also allows for several other useful features, like the ability to quit from any menu, an automatic help system, automatic menu headings, and includes a library of useful functions for writing command line programs in general.

How it works

The basic concept behind Cmd– is that menus are treated as individual objects containing data that defines the behaviour of the menu with respect to other menus, and any functions which it may have assigned to it. A menu always has a string id by which other menus will identify it as the target of a user's input. This string must be passed to the menu at creation, and to any menus which have it as their parent menu. Functions are created as definitions of an Fx() function inherited from a parent CProgFx base class, allowing each function to be treated as a generic pointer type which may be passed to one or more menus which can then call it after receiving its input keyword from the user.

How to use Cmd--

Creating a program in Cmd-- is a little bit of work, but worthwhile for any project with more than one or two menus. The first step is to add the library files to your project, just as you would with any other C++ source or header file that you may have created. The Testrun.cpp file is not strictly needed, but may be useful as a reference for the menu creation calls.

Once you've added these files to your project, create your projects main cpp file, if you haven't created one already. This will contain your main function, along with the functions that your program will use. At the top of this file, insert the following along with any other file includes that your project may need:

```
#include <iostream>
#include <new>
#include <exception>
#include "GenericMenu.h"
#include "TMessageBox.h"
#include "ProgData.h"
#include "Functionalize!.h"
using namespace std;
```

CGenericMenu is the parent class of TCmdMenu, not strictly necessary for this system to work, but I wanted to write it this way in case I wanted to create some sort of an additional GUI menu class also inheriting from CGenericMenu in the future. Functionalize!.h and Functionalize!.cpp store the function library, containing functions which may be useful for your project at some point in the future. For the moment though, we will focus on the ProgData.h file which you will need to use regularly for your program.

A typical ProgData.h file will look something like this:

```
// Program Settings header file ////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

#include <string>
#include <iostream>

#define Program_Title "C++!"
#define Stylechar "-"

#define sNULL "NULL"

void Output_program_title(std::string Name);
void InitFxObjects();

// Function Pointer ids ////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

class ThisFunction: public CProgFx
{   public:
    DvFunction(std::string ina){FxId = ina;}
    void Fx();
};

class ThatFunction: public CProgFx
{   public:
    MpFunction(std::string ina){FxId = ina;}
    void Fx();
};


CProgFx * Fx1, * Fx2;


void InitFxObjects()
{   Fx1 = new ThisFunction("THIS");
    Fx2 = new ThatFunction("THAT");
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

void Output_program_title(std::string Name)
{   int l = Name.length();
    l = 80 - l;
    double bracketing = l/2;
    l = bracketing;
    std::string Greetings;
    for (int cy = 0; cy <= l; ++cy)
    {   Greetings.insert(Greetings.begin(), Stylechar);
    }
    Greetings.insert(l, Name);
    l = Greetings.length();
    l = 80 - l;
    for (int cy = 0; cy <= l; ++cy)
    {   Greetings.insert(Greetings.end(), Stylechar);
        if (Greetings.length() >= 80)
        {   break;
        }
    }
    std::cout << Greetings << NEWLINE << std::endl;
}
```

What each part does is fairly simple. The #defines for Program Title and Stylechar control what the program prints into the command line at program start, and the character that is used to bracket it on the first line, respectively. This can be changed as you see fit for your program, although I still personally prefer the "-" bracketing style in my programs. The character wrapping for each menu can be edited similarly at the top of TCmdMenu.cpp (look for Stylechar2).

Functions in Cmd-- are currently handled as specific implementations of an inherited function member of the parent class CProgFx. In order to use a function in Cmd--, you will need to create a class derived from CProgFx like this one:

```
class ThisFunction: public CProgFx
{   public:
    ThisFunction(std::string ina){FxId = ina;}
    void Fx();
};
```

Quite simple really, just a constructor to take the string id of the function, and the function itself that the class wraps. Once you have created this, you will need to add a specific CProgFx pointer for it, and its initialization in InitFxObjects() like this:

```
void InitFxObjects()
{   Fx1 = new ThisFunction("THIS");
    Fx2 = new ThatFunction("THAT");
}
```

The whole point of this exercise being to make each functions location accessible in a generic manner to the program at runtime.

If none of this is even vaguely making sense for you, I would have to recommend that you try reading a good tutorial on the C++ language before continuing (the one at cplusplus.com is particularly good in my opinion). There's nothing inherently wrong with learning a programming language by trial and error, but eventually almost every programmer would agree that the time wasted on trial and error will end up being more than you spend reading the tutorial.

In your main cpp source file, the function can now be defined by a call like this:

```
void ThisFunction::Fx ()
{
    cout << "'ello World!";
}
```

And you can then set up the menu structure like this:

```
int main()
{   Output_program_title(Program_Title);
    InitFxObjects();
    CGenericMenu * main = new TCmdMenu ("MAIN", "Main-Menu");
    CGenericMenu * hello = new TCmdMenu ( "MAIN", "HELLO", "Sub-Menu");
    (hello)->Embed_function(Fx1);
    (main)->menu();
    return 0;
}
```

Output_program_title() and InitFxObjects() are just the calls defined earlier in ProgData.h. The pointers main and hello are used to create each menu object, and the function at pointer Fx1 is assigned to the hello menu. If you compile this program it should start at the main menu, and will print "'ello World!" if you type in "This" in the sub menu.

TCmdMenu has only two overloaded constructors. (a huge improvement in 1.10, as far as I am concerned) The first one is used for creating main menus, the only menu in the program that only needs to access down child menus, but not upwards to a parent menu. In the main example above, "MAIN is the access string which the program matches input to, and "Main-Menu" is the nice title that gets printed onscreen for the users benefit. The second constructor is similar in all respects, save that it takes a string id of the parent menu that it can access upwards. Menus constructed either way can access functions added to the MFxPointers container of that menu by the Embed_function method, as shown above.

When typing the string ids of both menus and functions, it is very important that they are added in all caps. The input handler of each TCmdMenu automatically makes user input uppercase for comparison against possible actions; if you type your id strings in lowercase, they cannot be accessed by any user input, a rather irritating bug for your program. This limitation will be fixed soon.

That should pretty much cover everything needed for basic use of Cmd–. I will likely add more features at some point in the future, but the core of how the system works should (hopefully) remain unchanged for quite a while.

I would like to thank everyone who has helped with the creation of this library, including, but not limited to, Kuddel and Enjo from the Orbiter Forums, L B, cire, computerquip, helios, DTSCode, Superdude, ne555, and many others who frequent the cplusplus.com forums.

And, now that this library is fairly useable, I suppose I might as well get back to writing some actual programs.

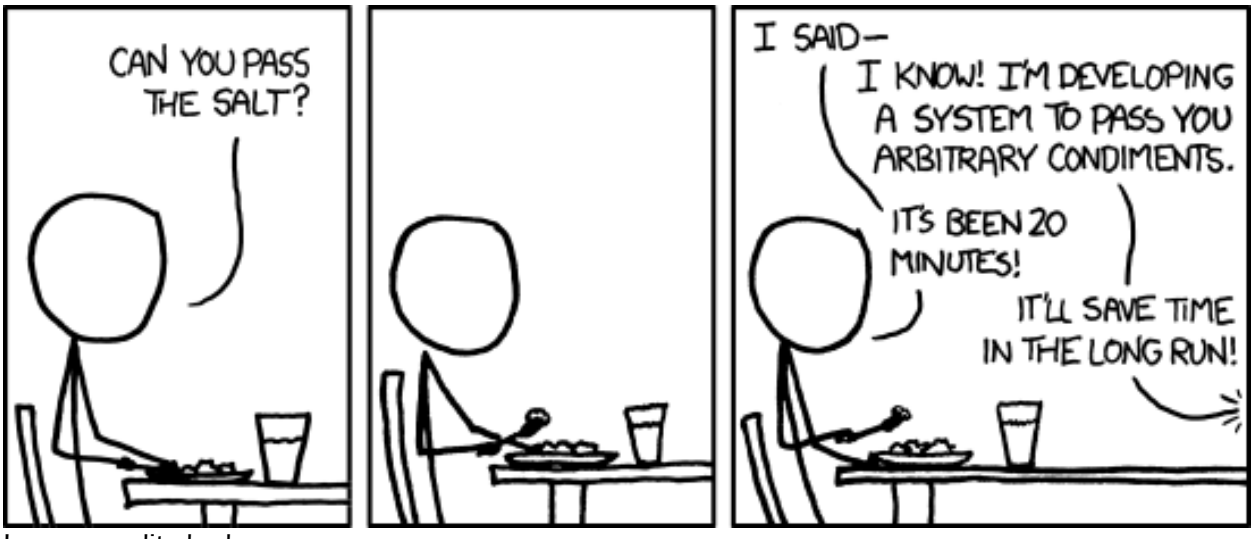

Image credit xkcd.com