The FORCE

The Flow of Regulated C Expressions

A graphical programming environment for beginners learning the C language.

Authors: Jeff Alexander, Leonard Kirschen, Michael Pellauer

Specification Document

Introduction

The FORCE is a graphical development environment designed to aid novice programmers. One of the hardest barriers for beginners to overcome is learning to express their ideas in terms of the arcane pigeon-English semantics of modern programming languages. The visual flow graphs of The FORCE aid users in making the transition from the ideas in their head to the implementation on the screen. The FORCE is not intended as a crutch to lean on, rather as a safe, exploratory environment where novices can experiment and devlop software quickly, without the fear of unintelligable compiler errors. Eventually, it should aid in transitioning the user smoothly to the world of fully text-based coding.

From a specifications point of view, this means that The FORCE must strike a delicate balance between features and limitations. While the environment must be diverse and offer many features of C, the whole language cannot be supported unilaterally. Similarly, while The FORCE includes a GUI builder, it is not intended to give the user unlimited control over every widget and event. In the end, if a user becomes tired of the limitations of The FORCE and moves on succesfully to a standard text-based environment, then our program has been a success.

The Flow Graph

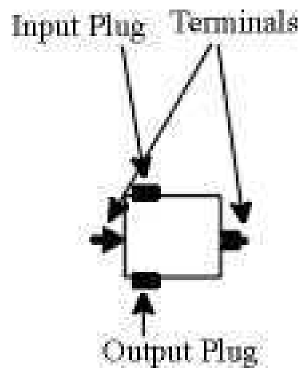
In The FORCE users program visually by constructing a graph which resembles a flow-chart. Nodes on the graph represnt data, and operations on data. The edges which connect the nodes determines the logical order in which operations are performed, as well as the flow of data from one operation to the next. The Flow Graph is the central feature of The FORCE, and must be flexible, yet robust; powerful, yet easy to use.

Flow

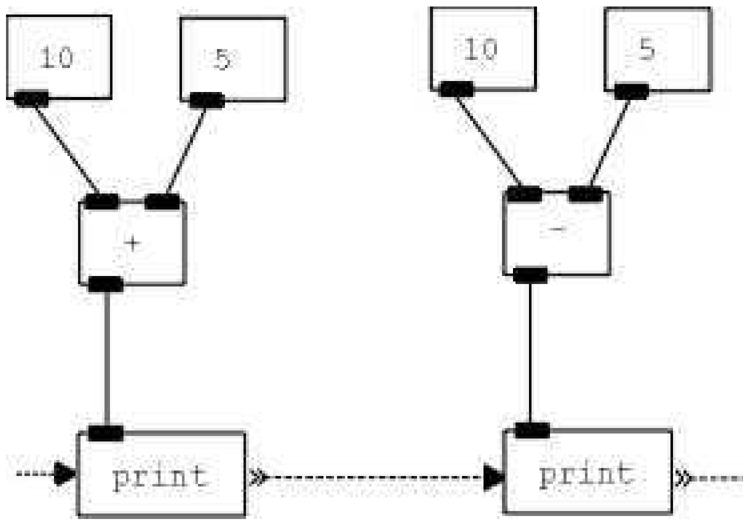
One of the central concepts novice programmers must grapple with is flow-of-control. In The FORCE flow is represented by connections between between nodes on a graph. The user literally maps out the path the program is going to take, including all branches, loops, and recursion. Flow of control is represented by dotted lines with arrows to indicate which direction control is flowing. These lines are connected to "terminals" on the sides of operations, data, and functions, mapping a linear path the program will take.

First Function 1 is executed, then function 2, then 3

Perhaps more important than the flow-of-control is the flow-of-data. In a computer program the result of one calculation is often fed into another function, the result of both of which is fed into a third function which prints it to the screen or stores it in a variable. This data-flow, as it is referred to in The FORCE, is represnted by solid lines which connect data to functions that will be performed on it. Each node on the graph may include input and output "plugs." the user creates a connection by clicking on the input plug of one node, and dragging a line to the output plug of another (or vice versa). The user may click the mouse button again to put bends in the line, allowing for a cleaner layout. The FORCE immediately does checks to make sure the connection is legal (ie that the user is not connecting two input nodes, or connecting a string to a parameter that requires an int). If the connection is illegal The FORCE beeps and does not create the connection. The user may later "cut" connections either by selecting them and pressing delete, or with the use of a special scissors tool.



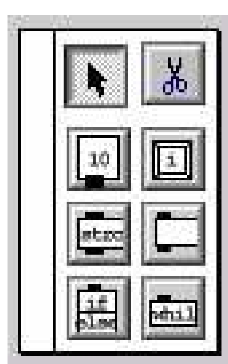
By combining flow-of-control and data-flow, it is possible for the user to construct some quite advanced programs. Undertanding the difference between the two types of flows is vital for succesful understanding and use of the flow graph paradigm. This is facilitated visually by the different types of lines, and the clear difference between "terminals" on the sides of nodes, and "plugs" on the top and bottom. In addition, the user must be guaranteed that when the flow-of-control brings the program to a place where data-flow occurs, all data operations will be performed before the program continues along the control path.



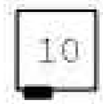
The user is guaranteed that 15 will be printed before 5.

Nodes

Nodes are the building blocks which the user uses to construct the flow graph of their program. Specifically, there are nodes to represent data, operations, functions, variables, conditionals, and loops. The different types of nodes are created through the use of a special tool "palette," or floating window. The user selects a tool from the palette, then clicks on the graph area of the flow-graph window, and a node is created at that point. Once a node is created, the user has full control over its size and location through the use of a special selection tool, which may also be used to delete nodes. The user may select multiple nodes by holding down the shift key. When a node is moved all connecting lines move to stay attached to it. When a node is deleted, all connections to it are also deleted. The user may cut, copy, and paste nodes and connections at will.

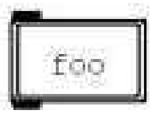


The most basic node is the value node. Value nodes are just that - values. The user types in a value, either an integer, double, char, or string. Connecting a line from the output terminal of the node passes the value unchanged to the input of the next node.



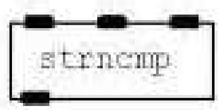
Value Node

More complex is a variable node. Users create variables with a special create variable button, allowing them to specify information such as type and default value. When the user creates a new variable node they indicate which variable it represents. Variable nodes have an output node just like value nodes, which outputs the present value of the variable. But they also have an input node, allowing the user to change the variable's value.



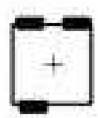
Variable Node

The most important type of node is the function node, which represents predefined functions and operations from standard C libraries. Function nodes have a variable number of input plugs - one for each parameter the function requires. They also can have a variable number of output plugs, to account for situations such as returns by reference. Function nodes also include user-defined functions, which are discussed later.



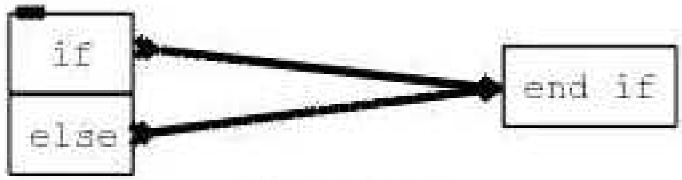
Function Node

An important subset of functions are operator nodes, which represent basic mathmatical operations such as addition and subtraction, greater-than and less-than. Operator nodes always have two input plugs and one output plug.



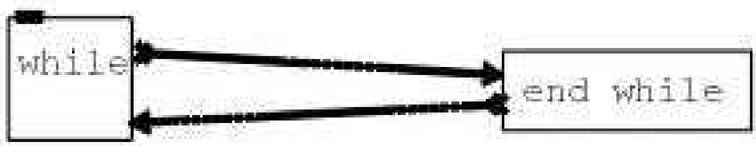
Operator Node

The FORCE inlcudes special types of nodes for flow-of-control. First is the if/else node. Notice that this node is very different because it allows the flow of control to branch. The if/else node takes one input - a boolean value. If the value is true the "if" branch is followed. If it is false, the "else" branch is taken. The two branches reconverge at the special "end if" node. This node, which has no input or output plugs, is created when the if/else box is created, and removed if it is removed - its only purpose is to reconnect the branch, acting as "}" in C.



if/else Branch

The while-loop node is similar in that it also alters the program's flow of control. Like the if/else node, it always is connected to an "end-while" node. However, notice that the flow of control loops back from the "end while" to the while. As long as the boolean value given to the loop's input is true the loop continues. When it is false, execution proceeds from the point of the "end while" node.



While Loop

Finally, users may annotate their graphs with the use of special comment objects. These comments have no bearing on the function of the flow graph as a whole, but must be incorporated into the final generated code.

User-Defined Functions

User-Defined functions used within a flow graph behave the same as the library functions discussed above. The function is represented by a single node, with input and output plugs, as well as flow terminals. The number of plugs will vary with the number assigned by the user.

Two types of user-defined functions exist within The FORCE. The sub-graph class of functions consists of the specification of parameters, return values, and a flow graph describing the function. When a user creates a new sub-graph function, a new window is presented divided into a few sections. First, an area on the side of the window provides a scrolling list of all variables available within the current scope. An area is also provided to enter a name for the function being defined. Next, at the top and bottom of the window, thin scrolling areas exist for specifying parameters and return values. These consist of a series of rows of type/name entry fields. A popup menu allows the user to choose an existing type for a parameter or return value, and a text edit field allows the entry of a name for the item. Variables created in this method are made available in the variable list of the Graph Builder. Finally, the largest part of the window is available for constructing the flow graph for the function.

Start and End terminal nodes are provided to define the beginning and end of the function. Any pre-built nodes or user-defined nodes may be used in the sub-graph, including a node representing the function being built, allowing for recursion.

Once the user has finished editing a sub-graph, checks must be performed to assure that the function signature does not match that of any existing signature, within the bounds of The FORCE.

The other type of user-defined function available is the code function. As above, the user is presented with a window providing a means of entering both a function name, as well as parameters and return values. Unlike the sub-graph editor, no variable list is present, and the canvas for constructing a sub-graph is replaced with a text edit area. This area is used to enter C code for the body of a function. Since parameters and return values are specified using the GUI, only the body of the function is written in the text edit area. Upon code generation, the code entered will be written directly into a code file.

Like in the sub-graphs, any user-defined function (graphical or code-based) may be called, as well as any C accessible function available on the system. If additional libraries are needed, fields will be provided in a project settings window to list additional compiler flags. Restrictions on the signatures of sub-graphs also apply to code-based functions.

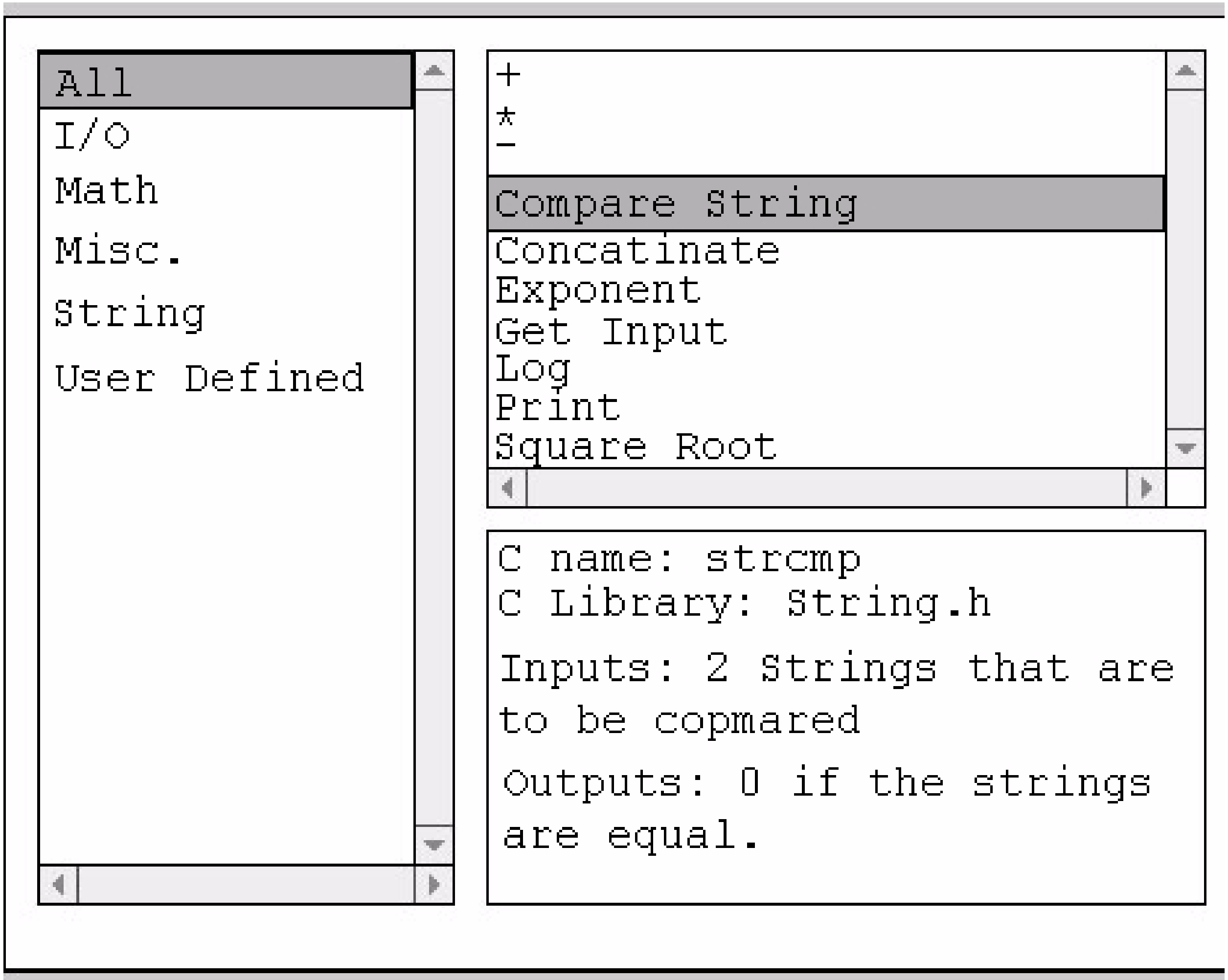
Once any user-defined function has been written, it becomes available in the Function Library (described below). The user creates calls to the function by choosing it from its listing in the Function Library, thus avoiding repeated “code”.

Function Library

The Function Library is the user's means of accessing all pre-built, and user-defined function nodes available for use in a flow graph. The Function Library is responsible not only for providing a means of choosing a function, but also for informing the user of what each function is used for.

The Function Library window is divided into three sections. Along the left side of the window, a scrolling list presents the user with a number of categories of functions to chose from. Clicking an item in this list fills in another list, occupying the top right-hand side of the window with the functions available in the selected category. Example categories include Math, Strings, Operators, and User-Defined. Below the function listing, another region, filling in the bottom right, shows a brief description of a function when it is chosen from the function list.

Each function present in the Function Library is internally associated with a singleton class representing the code to be created for that function.



Internal Representation Of Program

To store the program that a user generates, The FORCE must track a large data structure internally. This data structure will match closely the visual layout of each flow graph. It will consist of nodes and connections between nodes, as well as a system for correlating functions embedded within nodes with actual functions (and code for functions) themselves.

The data structure should be object oriented in nature, allowing for great flexibility in the type of information stored within it. Nodes in the data structure will represent flow control constructs such as “while” and “if”, as well as generic function nodes.

The actual classes representing any user-defined function will be singletons, and each instance of, or call to, that function will simply be represented as a node in the graph with a pointer to the singleton. It is important that graphs not be copied, as this could lead to very large memory requirements. Classes representing pre-built functions will have keys which can be used to look up actual code in the function library when code generation time comes. These will also be used to look up parameter types, as described above in the section describing Connections.

Another important piece of data stored in the data structure for a flow graph is the physical layout of the graph. Rather than incorporating graph layout code, the absolute position of each element in the graph will be stored in the node and saved when the program is saved.

Two Ways To Build Programs

There are two different types of programs that can be created with The FORCE: A command line based program and a GUI based program. When users start up The FORCE, they will be asked which of these two options they would like to use.

The CLUI Builder

The more straightforward way to build programs is from the command line. The FORCE will maintain a function graph representing the main routine of the program. The main graph will be the same as any other procedure except that there will be indicators at both the start and end of the main routine indicating where the program begins and ends.

Arguments from the command line will also be supported by The FORCE. The arguments will be stored in an array of strings which will be visible to the programmer on the variable list for the main routine's scope.

The GUI Builder

The GUI Builder allows users to create event driven programs. When users choose to use the GUI Builder, they are given a window. They can then place standard GTK widgets on their window by dragging them from a palette of supported widgets. They can also create additional windows as they see fit.

Supported widgets will include push buttons, radio buttons, labels, text fields, slider, scroll bars, multiple windows, dialog boxes, and menus.

Knowing the widgets selected by the user, The FORCE creates a list of the callbacks that the user can define. The user can then create a graph for the each callback routine. For these graphs, the indicators at the start and of the routines indicate which widget and which callback routine the graph has been created to respond to.

All details of entering the main loop will be hidden from the programmer using The FORCE. All that the user is aware of is that he or she is defining routines for the events he or she would like the program to respond to.

The Code Builder

The Code Builder takes all the program's function graphs and creates the C code for the user's program. The builder, given the data structure corresponding to the graph, creates the code for each node. The C code should be highly readable, using the same variable names and supporting user comments. In the case where The FORCE's reference to a function does not have the same name as the corresponding C function (i.e. if The FORCE calls a function print(), which appears as printf() in the code), a comment will be inserted which says what that the C function corresponds to on the graph and how the C function works (i.e. why we need a "!" before strcmp()). There will be an option for more advanced users to turn off such explanatory comments.

For programs created with the GUI builder, The FORCE's output will have a main routine which creates all the widgets and then defines the callback methods according to the function graphs specified by the user. For programs created with the CLUI builder, The FORCE creates the main routine as written by the user. In both cases, The FORCE generates the code corresponding to the appropriate nodes on the function graphs created by the user.

Each different graph will be defined as a different routine with an appropriate header comment. Subroutines will be called just as they are in a regular C program, which means they should have their prototypes declared at the beginning of the program. All such declarations will also be commented. By seeing such comments and how routines are separate, the user will get a feel for such concepts as modularity and scope.

Also created with the C code will be a makefile. The make file should allow support the standard make command for compilation as well as the traditional "make clean" command.

Menus

• **File**: The File menu should allow the user to open a new project (either GUI or CLUI), to load, to save, to save as, and to quit The FORCE.

• **Edit**: The Edit menu allows standard cut, copy, and paste commands and will also allow the user to undo and redo. Select all will also be implemented so that a user can easily include a whole program as part of another program. We will also have a project settings option. This option, which may be under an "advanced" section, will allow the user to specify include paths, library paths, or anything else that might be necessary for compilation. Different options can be made available to specify the level of commenting.

• **Code**: The Code menu is where the user specifies that he or she would like to generate the C code for his program. There will also be a "clean" option-- the equivalent of the traditional "make clean" command-- and a run option. The run option will execute the executable after compilation.

• **Windows**: The Windows menu allows the user to display or hide any windows he or she may or may not want shown. Such windows will include the tool palette, the function library, and the variable list.

File Format

In order to load and save files, a file format for representing function graphs must be created. It is important that this file format keep track of the location on the screen of all its components, as graphs that look different can represent the C program. The user should see his or her graphs when The FORCE loads a file.