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

Introduction

• What is The FORCE?

The FORCE is a tool intended for novice programmers that aids in the process of developing simple C programs by allowing users to manipulate graphical objects which correspond to C semantics. The FORCE easily expresses powerful concepts such as flow of control and parameter passing. Programs generated by The FORCE are always syntactically correct, allowing beginners to focus on implementation rather than sifting through endless obscure compiler errors.

• What isn't The FORCE?

Graphical flow-based programming represents a complete paradigm shift from traditional text-based coding. However, The FORCE is not intended for experienced programmers to build large scale software. Rather, its unique flow-based "diagrams" allow novices and intermediate programmers to easily move from the design process to implementation.

Requirements Overview

• **Program Builder**: This is the environment used to "write" the program. It involves creating a flow chart-like graph of regulated C expressions. This *flow graph* represents the operators, conditionals, loops, functions, and data. The FORCE should be able to traverse the graph, executing all of the appropriate code. (Priority: 1)

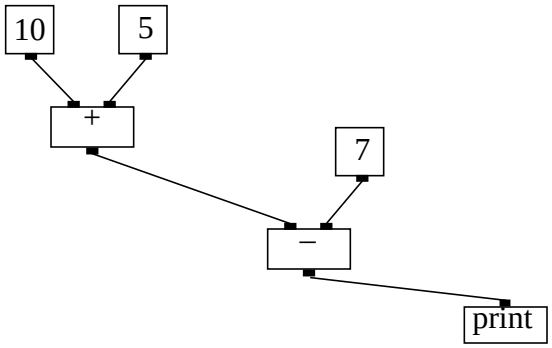
• **Code Builder:** Additionally, the user must have the option to generate a working C program which is completely independent of The FORCE and all but indistinguishable from a hand written program. The code must be readable, well structured, and commented. (Priority: 2)

Program Builder

The Flow Graph: The flow graph is the fundamental core of The FORCE. It allows the user to build software without writing a single line of code. Instead, the user programs by creating and connecting graphical elements which represent basic C operations, functions, and data. These elements are then connected to form a series of operations, conditions, and loops that make up a complete program.

Each element of the program should be represented as a box with input and output "plugs." These plugs are strongly typed and correspond to parameters and return values. By connecting output plugs to input plugs, the user can pass data from one operation to the next. For example,

the following diagram illustrates simple addition and subtraction.



This snippet would output “8”.

The types of parameters and return values must be checked as the user creates connections between elements. This will ensure that the code created by The FORCE constitutes a syntactically valid C program.

An additional type of connection, known as a “flow-of-control” connection, will be used to connect disparate elements of the graph without the passing of data. These flow-of-control connections will be used to represent branches and loops in the graph. Flow-of-control within a graph will begin at a special “start” terminal and end and a special “end” terminal.

Execution: When a program is executed, the user is shown the traversal of the graph beginning at the highest’s level’s start terminal. To show this traversal, each element will become highlighted when the program reaches it. When traversal reaches an element that requires input values it will go into a semi-highlighted state to represent that more information is needed. Traversal will then proceed up each input from left to right, evaluating each element in turn. Once all input values have been evaluated, the current element becomes fully highlighted, and traversal proceeds down the output value(s). The element reverts to its default color. If there are no more output values to follow, execution continues down the appropriate flow-of-control connection, until the “end” terminal is reached.

Language Features: Some language features that must be supported are base types (priority 1), basic function calls (1), literals (1), loops (1), conditionals (1), operators (1), arrays (2), and recursion (5). There must be three types of subroutines. The first type is a call to a library function, such as strcmp() or exp(). The program must include the most common standard ANSI C functions (3). Secondly, the user must be able to define subroutines using the program graph exclusively, effectively creating “sub-graphs” (4). Lastly, an advanced user must be able to insert actual C code directly into special code elements of the graph (6).

Code Builder:

The code builder generates the C code for the program graph and the code from the interface builder. The code must be commented, and the names should be consistent with those given by the user. In addition, the code should be well organized and formatted in a clear fashion so that the code can be easily explored.

A low priority requirement is the generation of a Makefile to accompany the user’s code.

Additional Requirements:

The user must be able to save a project and to reload it later. (7) There should be global preferences and advanced options that the user can manipulate (8). At a lower priority, the user must be able to print out a flow-graph representing a function (10). Finally, special attention should be given to creating a user friendly application, as this system is intended for novice users.