A TECHNIQUE FOR ANIMATING ALGORITHMS OVER THE WEB

Chittaranjan Mandal
Dept of Computer Sc & Engg
IIT Kharagpur, WB 721302, India
Email: chitta@iitkgp.ac.in

Chris Reade
Kingston Business School
Kingston University, KT2 7LB UK
Email: Chris.Reade@king.ac.uk

Keywords: Algorithm, animation, web, CGI.

Abstract: We discuss a novel technique for animating algorithms over the web. Although there are several existing software environments for the animation of algorithms, some of which are web-enabled, ours is designed specifically to simplify the process of adapting an algorithm for animation and delivering the animation over the web with a simple web interface. This provides a first stage for more advanced development of web-based interactions to support animation. Our goal is to provide general web-based support to enable much more widespread use of animation in teaching. In particular we want to address the active participation of the observer in using algorithm animation technology. We describe the current implementation of the animation engine which is based on a simple co-processing method with CGI implementation on a web-server, along with plans to use this as a base to include emerging technologies (web-services with XML to markup examples and asynchronous interaction). We also illustrate the current web interface with some examples.

1 INTRODUCTION

An important aid in teaching early courses in algorithms and many other process-related subjects is to animate the particular process that is being described. This is a challenging task that has been addressed many times before but solutions are still not sufficiently simple for widescale use. Regular class room teaching without computer aids can address the problem only partially, by illustrating the steps for a specific example (usually) on slides. However, such illustrations are limited when it comes to demonstrating a process with different sets of inputs. Even leading edge animation techniques usually provide a high quality rendering of only a specific problem instance. As recently noted in (Naps, 2005), the real benefits of animation for effective teaching come from active participation of the learner in using the animation rather than from high quality graphics. In this paper we describe a simple animation technique that provides a general solution within a limited context. It works over the web for maximum flexibility and does not require the student to have anything more than a web browser to get started. We have applied this technique to teach an initial course on 'C' programming to a class of nearly 700 students.

The target users were learners with minimum understanding of tools and techniques of the learning target of programming. Some of the objectives of this work were as follows:

- avoid reliance on tools that the user may not have or may have difficulty with using at the early stages
- make it a web-based technique so that it is usable just via a web browser
- it should be interactive
- it should be able to work with user supplied data sets
- it should be possible for the instructor to create tutoring examples with relative ease

Initial learners usually have a problem with grasping the basic paradigm of programming. A good way to get them started is to walk them through simple programs. That way they can understand how the execution of a program progresses through various statements. They get to see the working of various control flow constructs and also how variables get updated. Hard-to-grasp concepts such as parameter passing and function calls can be graphically illustrated. All of these can make the learning a lot easier. These aids are already available, but not easily reachable by the novice or early programmer be-

cause the use of these aids often require some initial knowledge of programming. The other option is to provide personal assistance, an increasingly scarce resource. This web-based technique was developed to make these aids available over the web, especially to the early programmer. These techniques also work well beyond the initial steps of programming as the student proceeds to learn basic techniques such as sorting, searching and so on.

The rest of the paper is organized as follows. In section 2 an outline of the technique is given. In section 3 some examples are given to illustrate the technique. The underlying protocol used for the animation is described in section 4. This is followed by the conclusions and references. The source of a complete animation example is given in the appendix.

2 OUTLINE OF TECHNIQUE

The technique essentially relies on running a program, which is to be demonstrated, at the server. If the program should require some data, that is collected from the user's browser and fed to the program. The trace of the program, as it executes, and any outputs that are produced are returned to the user's browser. The program proceeds from one chosen point to some other chosen point within the program. In addition to the start and the end points, numerous additional points may be chosen from within the program. If necessary, a point can be placed between every adjacent pair of statements. The points should be chosen carefully so that the exhibited trace is neither too monotonous nor too macroscopic. It is expected that the program being demonstrated has been rigorously tested in advance so that it does not throw up unexpected bugs or exceptions.

The basic program is augmented with additional statements to do the following as it reaches a chosen point in the program:

- return display matter to the user's browser, as appropriate. The display matter will include information regarding the:
  - outputs generated by the program
  - values stored in variables that need to be displayed
  - run-time data structures indicating currently active function calls
- collect inputs that may be required by the program
- proceed to the next point once the user is ready or after some period of time.

To make the overall scheme more practical, it should operate in user space rather than a superuser space. That way, an instructor can setup such animations without demanding special system privileges. Only a regular web server is required.

It may be noted that the user's browser does not interact with the program directly. Instead it interacts with the web server on the server – this the normal *modus operandi* for browser interaction. This system relies on the CGI programming interface and there are supporting methods for the following:

- start the program that is to be demonstrated
- collect display matter generated by the program and return that to the user's web browser
- collect inputs required by the demonstration program from the user and feed them to it
- close the interaction session once the demonstration program terminates

2.1 Algorithm encoding

The algorithm that is being animated needs to be coded in a special way so that the animation is made possible. Presently, the animation is done using Perl. In the future, 'C' programs will be animated directly in 'C'. This avoids the complications of trying to emulate one language with another. The full encoding of the algorithm to animate the comparison of two numbers is given in appendix A.1.

The text of the algorithm is placed in an array (the Perl array `@program` in this case). The program starts with a call to `initInteraction()` to make necessary initializations. Thereafter, the program continues in *mini-sessions* which involves opening the session, doing the required processing which may involve some combination of collecting inputs, executing and generating outputs, and finally closing the session. Once the mini-session is closed everything generated in the current session is returned to the client's bowser where the simulation is being viewed. A mini-session is opened by a call to `openInteraction`. Helper functions such as `retrieve`, `htmlStart`, `displayProg`, `htmlFinish`, etc. assist the input/output operations. A call to `closeInteraction` in conjunction with `htmlFinish` helps to close the interaction mini session. The encoding procedure is highly mechanical in nature and may be easily automated.

As a next step we plan to use XML to encode example programs and animation markup. This will allow us to develop an interaction to support the example developer in the generation of the animation with as much automation as possible. This could then become a web service with an option to add the result to the animation server. To enrich the interaction, we plan to use (AJAX-style) technology to allow the end-user more control over the layout of the animation.

This is particularly important when more complex examples need to be displayed, so the user has control of the focus.

3 ANIMATION EXAMPLES

3.1 Comparing two numbers

First, a simple example is presented to illustrate the working of the system in figure 1. This example compares two numbers `n1` and `n2`. Only the open screen, the first two steps and the last step have been shown. The following features of the system are illustrated via this example.

- interactive abilities; user inputs can be accepted, outputs are displayed
- ability to display values of selected variables
- program tracing; the statement being executed is shown in red
- buttons for stepping and running through

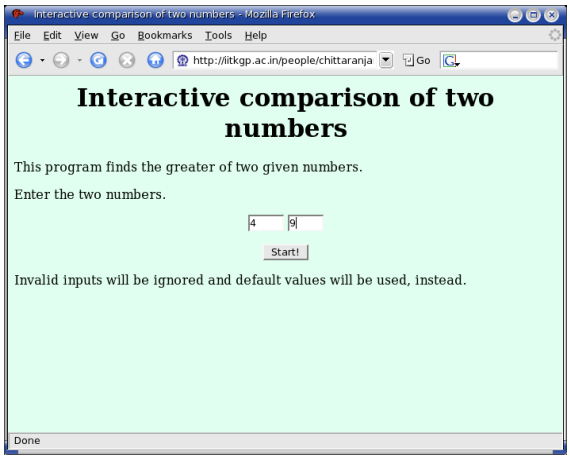

Figure 1: Screen shot - 1 of an example to compare two numbers

3.2 Selection sort

Next, a few screen shots are shown for a more complex example of selection sort in figure 5. This example illustrates additional features, such as

- function calls
- highlighting a set of statements
- distinctive highlighting of comments and program statements

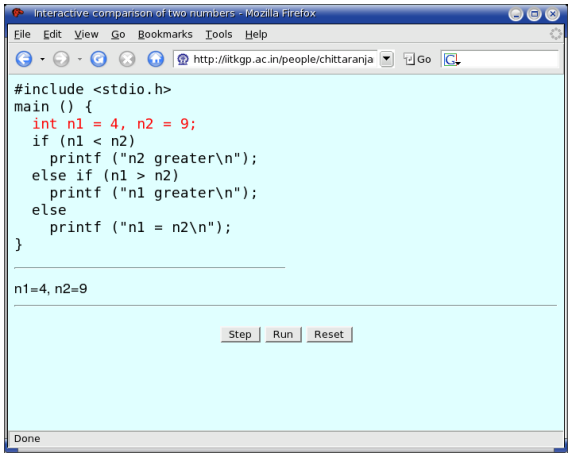

Figure 2: Screen shot - 2 of an example to compare two numbers

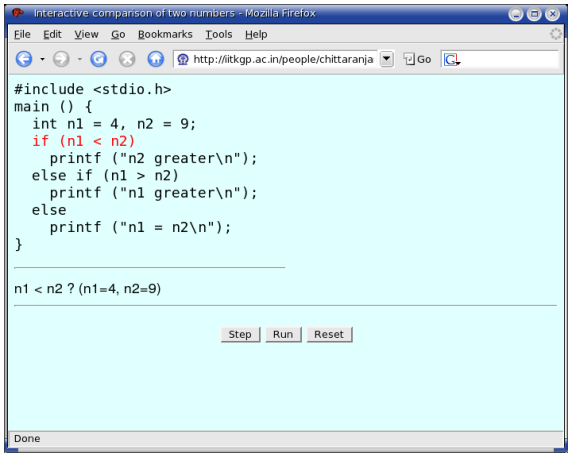

Figure 3: Screen shot - 3 of an example to compare two numbers

- call parameters of activation records to illustrate function calls
- more complex graphics; at present all graphics are implemented by means of *html* constructs; this is not essential

It may be noted that arrays in the activation record are denoted in a peculiar way. That is because the 'C' programs are emulated in Perl. The array address representation used in Perl has not been hidden.

4 PROTOCOL FOR ANIMATION

The protocol is illustrated in figure 7. The animation is initiated and continued over the web us-

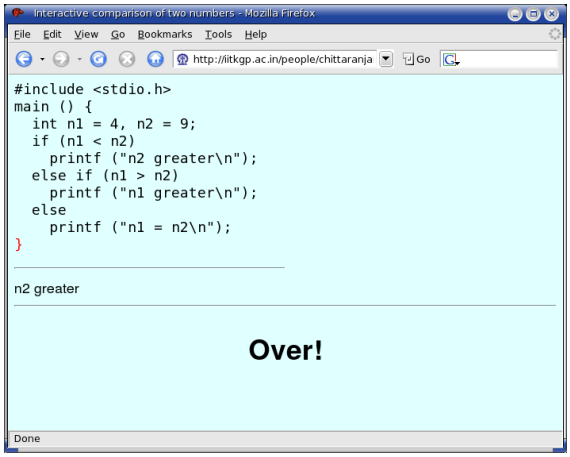

Figure 4: Screen shot - 4 of an example to compare two numbers

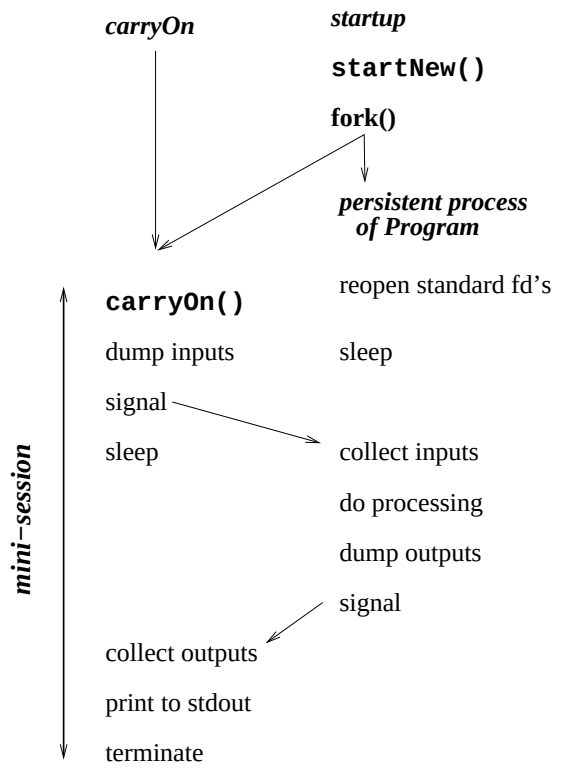

Figure 7: Animation protocol

ing the CGI(Robinson, 1996) gateway. Each CGI request is received by the HTTP server in a stateless manner. The protocol works by storing enough state information to be able to execute the program being demonstrated, receiving inputs from the user and returning appropriate output. The user supplies inputs via HTML FORMs.

There are two form actions, *startup* for initiating the animation and *carryOn* to continue with the animation until it ends. The HTML FORM to start the animation for the comparison example via *startup* is shown in appendix A.2. The *startup* receives a path to the executable and the necessary inputs. A call is made to a function called *startNew* which essentially forks to create a process for the program to be animated, and itself continues to form the first mini session. After a process is spawned for animation, its first job is to reopen its standard file descriptors of `stdin, stdout` and `stderr.` Thereafter it goes to sleep. If it is not awakened within a certain (reasonably long) interval it assumes that the user has abandoned the animation and it terminates.

The process participating in a mini session, continues with *carryOn,* whose job is to collect inputs sent by the user via the form and dump them into a file and then signal the sleeping animation process, and itself go to sleep. The process for the animation, on waking up, collects the inputs, does necessary processing, dumps necessary outputs into a file and then signals the sleeping co-process of the mini-session, and readies itself for the next mini-session by going to sleep. The co-process, on waking up, collects the output dumped by the animation process from the file, prints it to its `stdout` and terminates.

All subsequent mini-sessions are initiated by a separate script called *carryOn*. The form is submitted when the user clicks on the step button. The button click is performed automatically after a certain period of time if `Run` is enabled. This is achieved by a short piece of Javascript, which internally generates the effect of clicking the step button. This directly enters the mini-session, as shown in the diagram of figure 7 and as explained above. When the animation comes to an end, the last mini-session generates plain HTML output without the form. The animation process terminates once the animation comes to an end.

This system is based on the forward execution of a program and so replay of earlier steps, if desired, may only be achieved indirectly, with the help of the browser. This would, therefore, depend on the capabilities of the browser. Most modern browsers have "back" and "forward" buttons, using which, a few earlier frames may be revisited. If this browser feature is used, then the replay should be enacted only with the "forward" button and not via the "Step" or "Run" buttons. Their use in this context would be invalid.

Invalid use of the scripts is detected on the server side. The primary source of invalid use would be to continue with a fictitious trace of a process (as exemplified above). On the server side new keys are generated for every mini-session. A copy of the key is embedded in the form and another copy is retained for checking. If the check fails, the animation is aborted.

5 DISCUSSION AND RELATED WORK

Our goal is to provide general web-based support to enable much more widespread use of animation in teaching and to support more interactive use of animations in the learning process. The implementation technique described here provides support for simple adaptation of an algorithm so that an animation can be delivered over the web for viewing and controlling via a web browser. The advantages of this approach are two-fold. Firstly, the development of an animation does not require expertise in using complex animation software and is thus a much lower hurdle for teachers wishing to produce examples. Secondly, the delivery and viewing of animations requires a minimal, platform-independent capability to be available to the end user (i.e. just a web-browser) so that the potential audience is maximised.

At the moment, only examples in the 'C' programming language are being considered, and we do not provide full automation for the process of creating animations from example 'C' programs. In future work we plan to maximise the automation using XML to markup programs and a web-service to upload examples and provide fine tuning control of the animation generation through a web interface. We also recognise the need to support more graphical displays of complex data and objects for more widespread use of the animation tool. The preferred approach would be to develop document object models to update the client and keep issues of presentation separate from data as much as possible. This approach maximises flexibility and platform independence with modern standards-compliant browsers. However, we feel that a higher priority is to develop support for more interaction from the end user in running, controlling and changing animated algorithms. We believe that our simple approach to algorithm generation will enable more flexible and adaptable interactions than has been practical with traditional approaches.

Since the early days of algorithm animation, there has been an emphasis on the use of advanced graphics and support from large animation development environments (such as Zeus (Brown, 1991)). More recently the emphasis has returned to effective use in the learning process. With studies such as those discussed in (Hansen et al., 2000) and (Hundhausen et al., 2002) the importance of interaction rather than simple observation has been recognised. This new direction was also discussed in (Naps, 2005) where Jhavé was also described. Jhavé is a Java based animation tool which has been designed to take plugins for new animation techniques, and thus can act as a controlling shell for animations. This is relatively platform independent through the use of Java and can support an extensible set of interaction and animation styles. Another Java specific animation tool is Jeliot 3 (Sutinen et al., 2003) which has evolved from Jeliot 2000 and uses self-animating data types and is designed to allow easy uploading of examples by end-users. In (Bednarik et al., 2005) it is argued that smarter tools are needed to cater for and to adapt to the different abilities of users, referring to studies of taxonomies of software visualisation tools.

Our tool presently used mostly HTML and just a few lines of JavaScript on the client side. It is, nevertheless, a light weight and yet capable animation tool. We have coded over thirty six animation examples (manually) and used these to teach a first-level programming course. One important advantage of our approach is that the client CPU is not significantly loaded.

REFERENCES

Bednarik, R., Moreno, A., Myller, N., and Sutinen, E. (2005). Smart program visualization technologies: Planning a next step. *Proceedings of the 5th IEEE International Conference on Advanced Learning Technologies (ICALT 2005)*, pages 717–721.

Brown, M. H. (1991). Zeus: A system for algorithm animation and multi-view editing. *IEEE Workshop on Visual Languages*, pages 4–9.

Hansen, S., Narayanan, H., and Schrimpsher, D. (2000). Helping learners visualize and comprehend algorithms. *Interactive Multimedia Electronic J Computer-Enhanced Learning*, 2.

Hundhausen, C., Douglas, S., and Stasko, J. (2002). A meta-study of algorithm visualization effectiveness. *J. Visual Languages and Computing*, 13(3):259–290.

Naps, T. L. (2005). Jhavé: Supporting algorithm visualization. *IEEE Computer Graphics and Applications*, 25(5):49–55.

Robinson, D. (1996). The www common gateway interface version 1.1. In *http://weeble.lut.ac.uk/System-docs/Internet-drafts/draft-robinson-www-interface-01.txt and http://www.w3.org/pub/WWW/CGI/Overview.html*. IETF.

Sutinen, E., Tarhio, J., and Terasvirta, T. (2003). Easy algorithm animation on the web. *Multimedia Tools and Applications*, 19(2):179–194.

Appendix

A.1 Code to animate comparison of two numbers

```
#!/usr/bin/perl

require '../../cgi-bin/interactive/interactive.pl';
require strict;

# exit;

my $bgcolor = "#E0FFF";
my ( @fields );
my ( $outStr );

$n1 = 6; $n2 = 9;   # actual code of example

use POSIX qw(isdigit);

initInteraction ();

# starting interaction early to collect user inputs
openInteraction (); # start I/O
htmlStart ("Interactive greater of two numbers",
  $bgcolor);

my $pid = POSIX::getpid();
my $inputsRef = retrieve("${pid}Inputs.dat");

# get n1 with validation
if (exists $$inputsRef{'n1Raw'} ) {
  my $n1Raw=$$inputsRef{'n1Raw'};
  $n1Raw =~ s/ //g;
  if ("$n1Raw" =~ /-?[0-9]+/) {
    $n1=$n1Raw;
  }
}
# get n2 with validation
if (exists $$inputsRef{'n2Raw'} ) {
  my $n2Raw=$$inputsRef{'n2Raw'};
  $n2Raw =~ s/ //g;
  if ("$n2Raw" =~ /-?[0-9]+/) {
    $n2=$n2Raw;
  }
}


local @program = (
'<pre>',
'#include &lt;stdio.h&gt;',                # 2
'main () {',                              # 3
"  int n1 = $n1, n2 = $n2; ",             # 4
'  if (n1 &lt; n2)',                      # 5
'    printf ("n2 greater\n");',           # 6
'  else if (n1 &gt; n2)',                 # 7
'    printf ("n1 greater\n");',           # 8
'  else',                                 # 9
'    printf ("n1 = n2\n");',              # 10
'}',                                      # 11
'</pre>',
'<hr width="50%" align="left">'
);

displayProg (\@program, 4, "red");
print "n1=$n1, n2=$n2\n";
htmlFinish (closeInteraction (0, \@fields));

openInteraction (); # start I/O
htmlStart ("Interactive greater of two numbers",
  $bgcolor);
displayProg (\@program, 5, "red");
print "n1 &lt; n2 ? (n1=$n1, n2=$n2)\n";
htmlFinish (closeInteraction (0, \@fields));

if ($n1 < $n2) {

  $outStr = sprintf "n2 greater\n";

openInteraction (); # start I/O
htmlStart ("Interactive greater of two numbers",
  $bgcolor);
displayProg (\@program, 6, "red");
print $outStr;
htmlFinish (closeInteraction (0, \@fields));

} elsif ($n1 > $n2) {

openInteraction (); # start I/O
htmlStart ("Interactive greater of two numbers",
  $bgcolor);
displayProg (\@program, 7, "red");
print "n1 &gt; n2 ? (n1=$n1, n2=$n2)\n";
htmlFinish (closeInteraction (0, \@fields));

    $outStr = sprintf "n1 greater\n";

openInteraction (); # start I/O
htmlStart ("Interactive greater of two numbers",
  $bgcolor);
displayProg (\@program, 8, "red");
print $outStr;
htmlFinish (closeInteraction (0, \@fields));

} else {

openInteraction (); # start I/O
htmlStart ("Interactive greater of two numbers",
  $bgcolor);
displayProg (\@program, 7, "red");
print "n1 &gt; n2 ? (n1=$n1, n2=$n2)\n";
htmlFinish (closeInteraction (0, \@fields));

  $outStr = sprintf "n1 = n2\n";

openInteraction (); # start I/O
htmlStart ("Interactive greater of two numbers",
  $bgcolor);
displayProg (\@program, 10, "red");
print $outStr;
htmlFinish (closeInteraction (0, \@fields));
}
```

```
openInteraction (); # start I/O
htmlStart ("Interactive greater of two numbers",
  $bgcolor);
displayProg (\@program, 11, "red");
print $outStr;
htmlFinish (closeInteraction (1, \@fields));
```

A.2 HTML FORM to startup the comparison example

```
<html><head><title>
  Interactive greater of two numbers
</title></head><body bgcolor="#E0FFF">

<h1 align="center">
  Interactive greater of two numbers
</h1>

<p>
This program finds the greater of two given numbers.

<p>
Enter the two numbers.

<FORM METHOD="POST"
  ACTION="../../cgi-bin/interactive/startUp.pl">
<INPUT TYPE="hidden"
  name="tPath"
  value="~/public_html/interactive/2C/2C.pl">
<p align="center">
<input type="text" name="n1Raw" size="4"
  maxlength="4" align="center">
<input type="text" name="n2Raw" size="4"
  maxlength="4" align="center">
<p align="center">
<INPUT TYPE="submit" value="Start!" align="center">
</FORM>

<p>
Invalid inputs will be ignored and default values
will be used, instead.
</body></html>
```

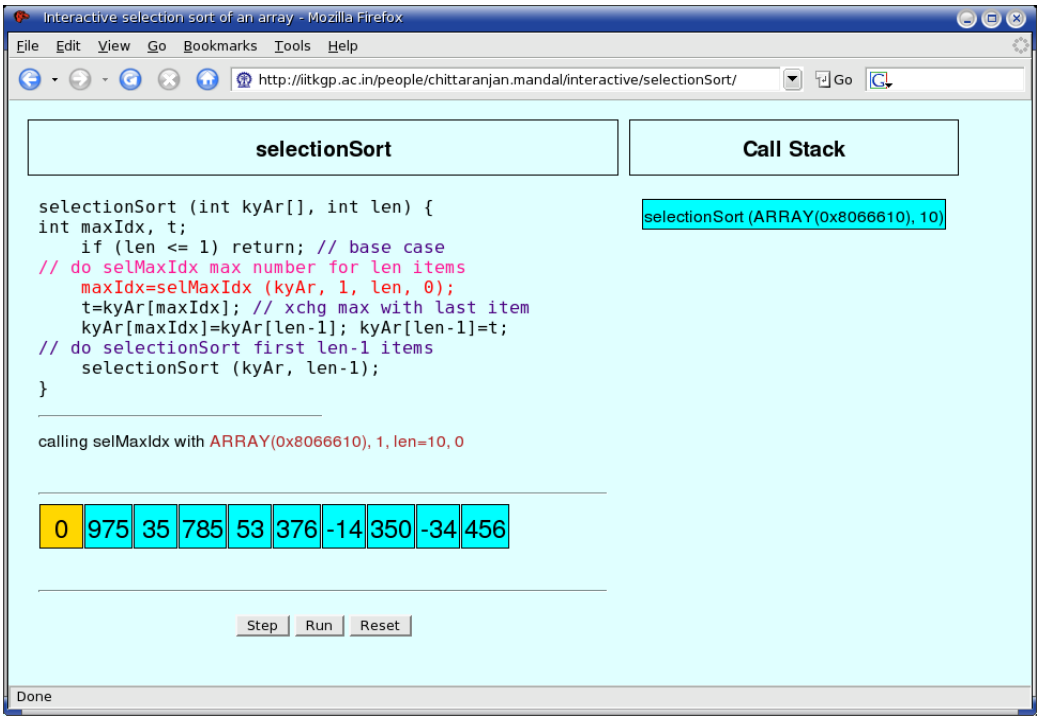

Figure 5: Screen shot - 1 of selection sort

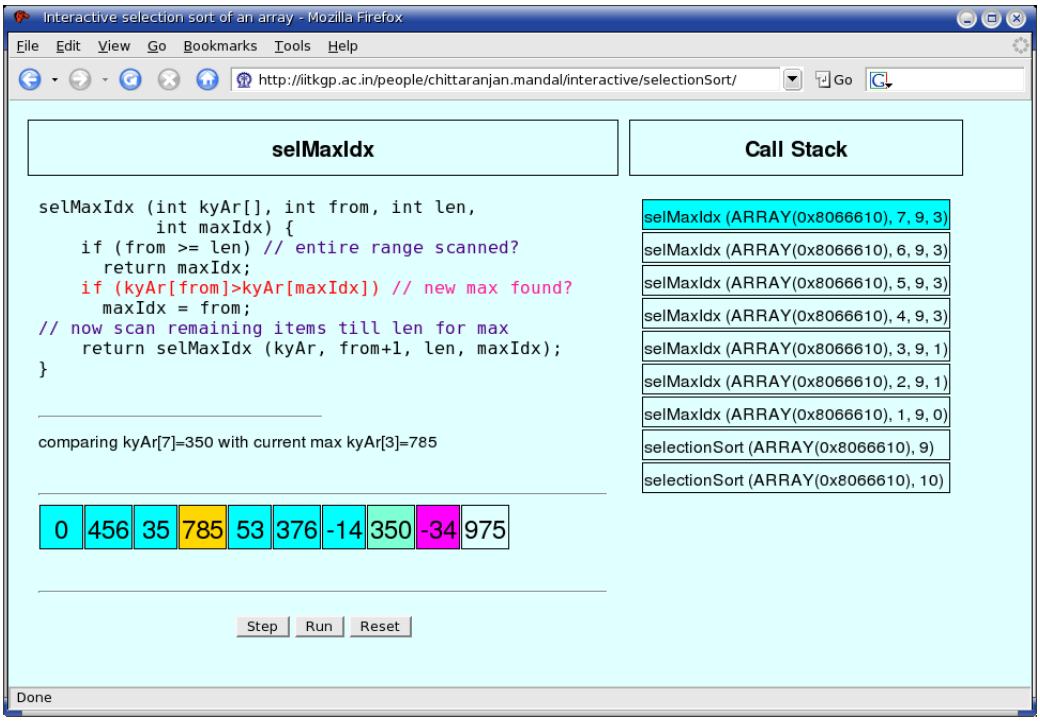

Figure 6: Screen shot - 2 of selection sort