

Edu-List

Educating Beginners on Linked Lists

Carmen Simpson

Supervisor: James Tam

Undergraduate Department of Computer Science
University of Calgary

1. Abstract

This paper deals with the difficulty of learning linked lists in such a way that the person being taught feels that they fully understand the finer points of such a data structure.

According to the constructivist approach to education, people learn best when they can get right in there and experiment with the material to be learned [1] because it allows the student to take what they already know, combine it with what they learn from the program, and create a more powerful and complete understanding of this data structure [5]. These principles were applied in an interactive environment that allows users to do just that; experiment, make mistakes, and in the process, come to a firm understanding of how linked lists actually work.

While there are other programs that have touched upon the concept of linked lists, this constructivist based project would ultimately place more of the control and the power to manipulate and investigate in the hands of the user, with the belief that this would prove to be a more useful tool for those wishing to learn all about Linked Lists. [1],[6],[2].

As well as basing the decision of Edu-List on principles from the education discipline, the design process will also be guided by fundamental principles in Human-Computer Interaction such as the principles of User Centered Design, [10] as well as low-fidelity prototyping techniques [7], [8] to bolster the soundness of the concepts before the actual computer implementation of the Edu-List program.

2. Introduction

Newcomers to the world of data structures and programming algorithms can find

themselves faced with an overwhelming mountain of terminology, concepts, and both basic and complex data structures and algorithms [1]. Even in the structured environment of schooling institutes, time allows little for an in-depth teaching of even the simplest of concepts before moving on to something even more involved and complex. Having the ability to be able to pull from other resources, specifically ones that are both interactive and make the learning process simple and even a little fun, can have a clear and definite advantage for the student just starting to learn about data structures. [2]. Take for example, the concept and implementation of linked lists. While the concept is fundamental to many programming languages, for a beginner, linked lists can be confusing and hard to crystallize in one's mind. It is not until one has had a chance to work with the data structure and see how the code he or she creates directly relates to the robustness and correctness of the linked list, can the concept take hold. [9]. As stated by Ben-Ari, a student who does not take the surest path to success immediately is just building up their understanding of the issue they are trying to learn about. In this way, mistakes can be just as important in the learning of the subject [1].

There are two main camps of thought, in the field of education, as to the way in which to teach others new concepts. One is the behaviorist approach, which is basically, that a teacher tells the student what to do, the student does it, and the student's behavior is observed and graded [2]. The other is constructivism. This is where what the student already knows is taken into account, and the student is allowed to actively come to the correct answers through their own interactions and experimentation. [1],[2]. Although both approaches are valid approaches, it is this more constructivist approach that will be taken for this project. The reason for this approach as opposed to the

behaviorist approach is discussed more later on and is hinted at in the previous work section.

So the goal is to create an interactive stand-alone program that allows the user to learn all the ins and outs of singly linked lists. The program would also help the user produce or create the pseudo code needed to create a complete linked list, without giving the user an easy out and simply producing the code for them.

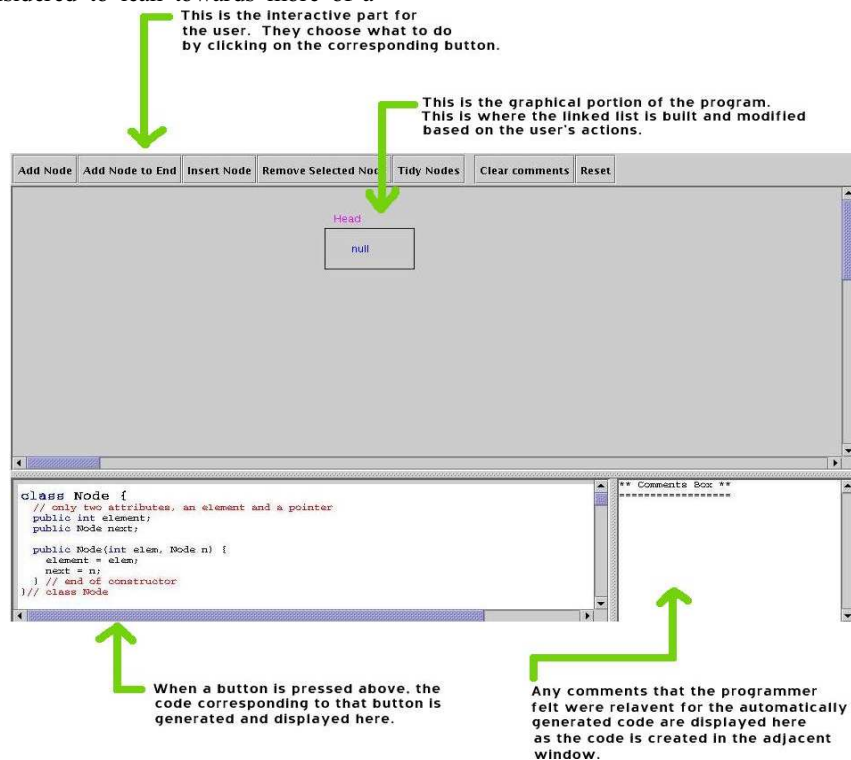
3.Previous Work

A program was created by Michael Wilson, a research student in the Dept of Computing Science and Maths at the University of Stirling in 2001. Figure 1 is a look at his program [11].

Michael Wilson's program offers actual code at the bottom of the screen, with the user simply clicking on a single button to add, insert or delete nodes from the list. The above program could be considered to lean towards more of a

interaction from the student, but there is not much experimenting going on. The user clicks on buttons at the top, and the node is perfectly added, inserted or deleted. There is no attempt required to figure out the code on the part of the user, or to stumble over any of the concepts, and in this way, the interaction is minimal.

In addition, using two concurrently running classes of students in their first year of computer science at the University of Calgary, a study was conducted to determine the strengths of the behaviorist approach in contrast to a constructivist approach. One class was primarily behaviorist in their teaching while the other class was primarily running classes of students in their first year of computer science at the University of Calgary, a study was conducted to determine the strengths of the behaviorist approach in contrast to a constructivist approach. One class was primarily behaviorist in their teaching while the other class was primarily constructivist. It was found that the students that were involved in the constructivist approach had better success in



behaviorist approach, since there is some subsequent computer classes

Figure 1: The Linked List Program created by Michael Wilson [11].

than those involved in the behaviorist approach [2].

Another similar study done at Grand Valley State University by Mark J. Van Gorp and Scott Grissom between objectivist -- which is when a student listens to the instructor, then practices what was taught by responding to the assignments by the teacher -- and constructivist teaching, they suggested that projects that were constructivist based might hold the attention of students longer while making the concepts being taught easier to understand. [6].

4.The Reason for an Interactive Constructivist Program

While a behaviorist approach can be useful in its own right, it is the hope that a more constructivist approach to learning linked lists would give users a more “in-control” approach to learning, as apposed to a teacher controlled approach. As described by Greening and Kay [5], constructivism is students taking their experiences of actively participating in the learning of a desired concept or topic, and combining it with what they already know and making a new knowledge base in their mind.

As well, in the study done by Becker and Parker [2], where one class was behaviorist and the other class was constructivist in their methods, it was noted that “Students from the constructivist class (class C) performed better in successor courses, on the average, than did students from the behaviorist class (class B).”(p. 5). Not only that, but quantitatively, Becker and Parker report that “the class C students achieved an average grade 0.7 higher than did the class B in the successor programming class; grades are compared on a 4-point scale, so this would represent an 18% improvement.”(p.5). For students just starting out, results like those means more success, and more success can mean more feelings of confidence and self-assuredness.

The proposed program would allow the user to work on all aspects of a singly linked list, while allowing the user to make mistakes and in the process learn from those mistakes.

By building their own mental model, the user can learn exactly what they need to do to implement a linked list, as well as being able to make mistakes and to see the consequences of those mistakes on the simulation. In this way, it

helps them to understand why all the different parts of the code are necessary to create a fully functioning linked list. According to Ben-Ari, learning isn't simply about giving a student a number of steps to follow, and when the student has completed those steps, that that then means that the student has learned and understands what they are doing [1]. The student needs to be able to explore, and by exploring, learn and understand and connect that new found knowledge with what they already know, whether that be in the shortest or the longest number of steps to the desired goal [3].

5.The Solution

5.1.The Original Solution

The idea was to use an appropriate metaphor to describe the functionality and the dynamics of a Linked List. A metaphor could help with a rather abstract concept by giving it a more personable face to it, something that would be entertaining and memorable.

The initial idea for the metaphor was that of an alien race called Triangularians. These aliens (Triangularians) would represent the nodes. Each alien's contents of their stomach would have been where the data was held (the contents of the node). The alien at the head of the line represented by the alien holding the scepter would represent the head of the linked list. Each alien in the line holds hands with the alien on either side, in the line. These held hands would represent the pointers to the next node. The alien at the end of the line would not be holding any one's hand, which would mean that there is nothing in that alien's hand, representing the null pointer at the end of a linked list. The aliens would look something like figure 2 in Appendix A.

This metaphor would ultimately be created and shown through an animation, which would be displayed on the screen at all times. Within the program, the user would be able to choose whether he/she would like to add to, insert into, or delete a node from the linked list being formed. Once a choice between these three would be made the prototype would have the set up of Figure 3, in Appendix A.

By clicking on “building blocks” of various phrases and symbols located on the screen in the button section, as pointed out in

figure 3, the user would be able to build phrases of pseudo code. The program would then analyze this code and the results of their pseudo code would be shown to them by manipulating the line of aliens.

The metaphor to be used was going to be tested, before actual implementation, by using Human Computer Interaction methods such as low-fidelity prototyping techniques [7],[8]. A Task Centered approach [4] as well as a User-Centered Design [10] were used in the implementation of the metaphor and the actual program.

5.2. Working with the Original Metaphor

The images of the original low fidelity prototype can be seen in Appendix A with a brief description included with the various screens.

A beginner programmer agreed to try the prototype out. In the process, it became clear that the level of abstraction needed to make the metaphor, (combined with the pseudo code,) actually understandable, the weaker the connection between the pseudo code and actual code became. It was also discovered with the original Triangularian based metaphor prototype, that some of the questions were trying to cover too much pseudo code, or too much of the linked list algorithm at one time, leading to too much ambiguity. The questions were not directed or specific enough, and so the desired results of a couple lines of pseudo code generated by the user, was at best difficult for the user, because the user didn't know how much to create or where exactly to end.

As the attempt progressed to make the questions more specific and less generic, it became increasingly clear that to succeed in making the metaphor work, more and more abstraction was needed in the pseudo code that the user was making, and thusly the phrases on the buttons in the button area. As the abstraction of the buttons and the pseudo code grew, any connection between the pseudo code and actual code that the user would then want to create based on their created pseudo code would be extremely weak. This of course was not the desired outcome, since part of the purpose of the

user making the pseudo code would be to then use what they had created to produce real code. As the Triangularian simulation evolved, that possibility became less and less viable.

Eventually the Triangularian metaphor was abandoned in favor of searching for a more direct and simpler concept to work with in the intended program.

5.3.The New Solution

The images of the new solution low fidelity prototype can be seen in Appendix B with a brief description included with the various screens.

The hope was that the new solution would lead to less ambiguity, and also no need to jump from a different metaphor to the actual node concept of the linked list. By just using the concepts of nodes and references, the phrases on the buttons would be much more straightforward, with very little abstraction. It would as well possibly be a more direct relationship between the pseudo code and how it relates to actual code.

With the new Node based prototype, questions were used that were far more specific and would hopefully not allow for so much play within the answer that could be given by the user, which would help lead to the correct answer.

6.The Design

6.1.Current Version

A simplified description of the layout can be seen in Appendix C.

So then once the jump was made from paper and sticky notes to the actual screen, the general concepts remained the same, but the layout changed quite a bit. Buttons were added for moving around the program, which would have to be there on every screen in the actual program, but were not absolutely needed to get the general concepts down in the low fidelity prototypes talked about

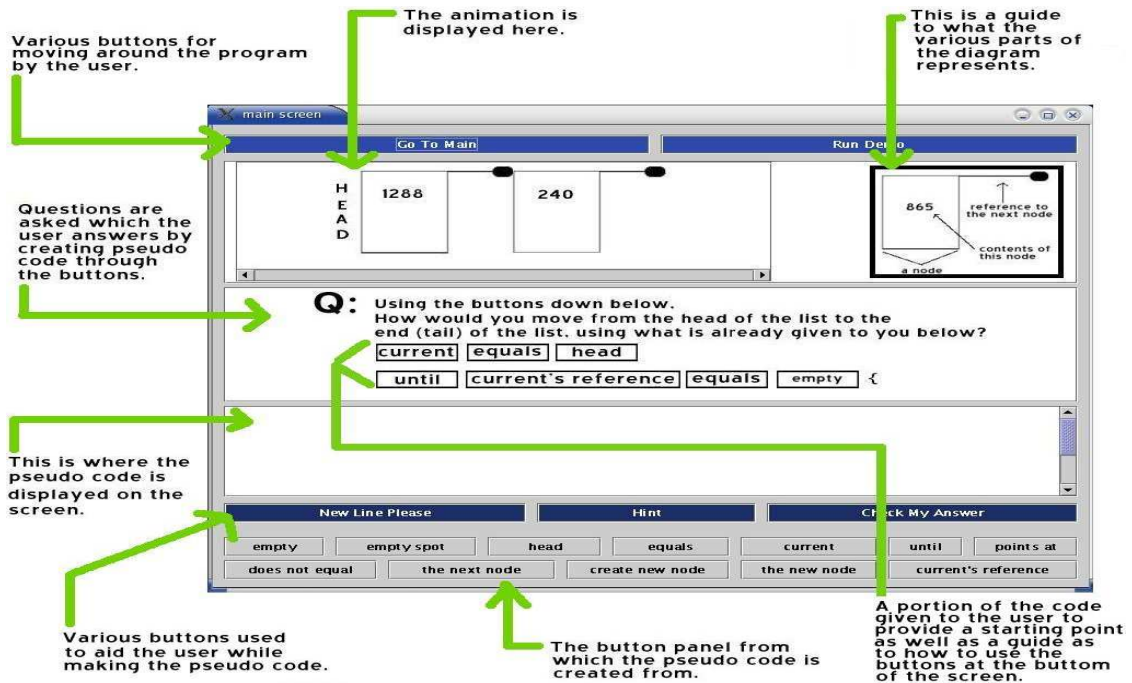


Figure 2: Screen One in “Add A Node”

and laid out in Appendix A and B. Figures 2 and 3 show the design as it stands right now.

The first figure is what the first screen in “adding a node” would look like, and the second one shows what the pseudo code would look like in conjunction with the question.

For reasons of time and for the sake of simplicity, it became clear early on that to account for every possible combination of buttons, and to supply every possible button that a user might want to use, would not be possible or practical. In order to achieve a working program in the period of time that was available for this project, the logic behind the program had to be a lot less complex than that. Along those lines, the program has a specified set of pseudo code for each question, along with a specified number of lines that that pseudo code should be

in. If the pseudo code that the user creates matches both of those parameters, then they are successful and move on to the next question. If the pseudo code or number of lines does not match, messages pop-up to try and help guide the user to a “correct” answer.

7.The Working Program

7.1. The logic and look of the program

Some examples of the various messages that pop up when a user has not created the “correct” pseudo code, can be seen in Appendix D. As well, screen shots showing the various screens for adding, inserting and deleting a node can be seen in Appendix E.

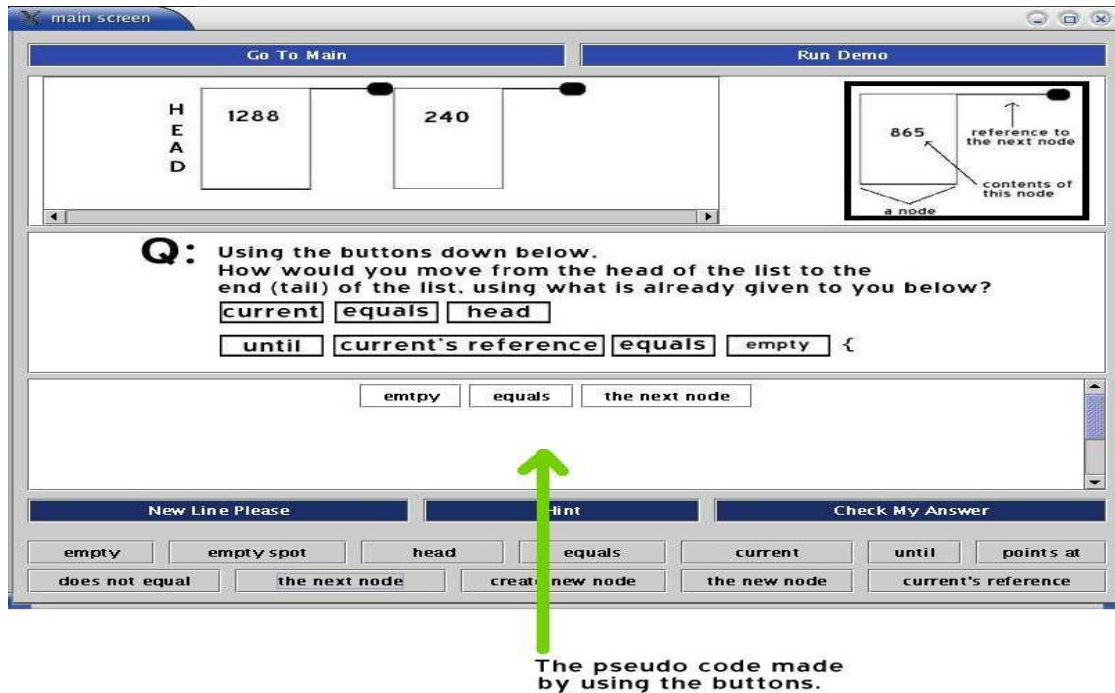


Figure 3: Making Pseudo Code on the “Add A Node” Screen.

8. Usability Study

8.1. The Participants

There were five participants in total. Some were graduate students, who are currently working as teacher advisers for first year computer science students. These graduate students were asked to be a part of the study because it was the hope that they would be able to shed some light on what their students would need or would be looking for in a program such as Edu-List.

The rest were undergraduate computer science students. Unfortunately, due to scheduling conflicts, where students taking the first course in computer science wouldn't be taught linked lists until the last week of the semester, and an inability to get any students taking the second course in computer science to volunteer as participants, there were not any participants who were from the intended target group.

For clarification, the target group for this project is considered to be first year students

who were just learning linked lists for the first time and who would be considered beginner programmers.

8.2. The Method

Each participant had the premise behind the program explained to them with a little bit of a background about the idea for the program, along with an attempt to get them to think about the program as if they were a first year student again. The participant was asked to sit down and try to figure out the program by trying to understand the questions asked in the program, and by trying to make the pseudo code for the answers. Afterward, they were given a questionnaire to fill out in regards to their impressions about the program.

8.3. Observations from the Study

To say that the results were varied would be an understatement.

The only question that the participants even somewhat agreed on was number 8, (see

appendix F for the actual questionnaire,) which asked if the participant thought the program would help beginner programmers learn about linked lists. The mark on that ranged from 3 to 2 on a scale of 1 to 5, where one was strongly disagree and three was moderately agree.

Some participants thought the Edu-list program was easy to use. They thought that the graphics were the best part of the program, and that the setup of the main screen was well done. They also liked the reference picture in the top right corner.

When asked if there was anything confusing about the program, this gave back mixed responses as well. Some thought the buttons were confusing, and others thought they were easy to understand. Some thought the questions were confusing, while others thought the questions were straightforward. Some participants thought that starter pseudo code given in the question that looked like buttons was confusing, and finally some participants didn't think that there was anything confusing about the program. While the majority thought the program was easy to use, one participant gave it a 4 out of 5 for difficulty.

9. Interpreting the Results of the Study

One of the reasons for the spread of the results can be placed in how much experience the participants actually had, and what language they were most comfortable programming in. And as was demonstrated early on, the skills and knowledge of the participants themselves directly affected the way they interacted with the program, and this ended up making the evaluation of the Edu-List program difficult at best. Indeed, this can be seen in the widespread and unique reactions and comments from each participant.

The issue with the buttons could be explained by the fact that the participants that had the most experience, were trying to think of the questions, or the making of the pseudo code in terms of a particular programming language, which was a comment that came up a number of times during the evaluating of the program. Some wanted the phrases on the buttons to be more specific to a particular coding language like C or JAVA. Edu-List had purposely tried to stay

away from a specific language, which for an experienced programmer, seemed to be difficult to require him or herself to such abstraction.

As well, despite trying to get the heavily experienced participants to think about the program through the eyes of a would be beginner programmer, this proved to be as challenging as putting training wheels back on the bike of an Olympic bicyclist. From the results, it appeared that the more experience a participant had, the more the program got in the way for that participant. For instance, repeatedly, the participants with the most experience tried over and over again to place all the pseudo code needed for the whole process of adding, deleting or inserting a node, on the first screen, beneath the first question. The program was designed to ask a number of questions, two or three per section of functionality, where the first question would have to be answered correctly before proceeding to the next question. Some participants seemed more intent on showing that they could make all the pseudo code, rather than answering the question and successfully moving to the next question. It seemed that they new the body of linked lists enough that they didn't need the questions to guide or direct them. In fact, it seemed they could see the code for a linked list, in the programming language that they preferred, already in their mind's eye, and it was in that way that they tried to put the pseudo code together -- the question at hand pretty much forgotten. The program of course was coming from the opposite direction, which was where the difficulty was for these more mature programmers. Where the purpose of the program was to have a user answer the questions, not knowing exactly what the resulting code would look like, until all questions had been answered and the results shown in the linked list at the top of the screen, these experienced participants could not unlearn what they already knew for the study. Which is fair enough. The program wasn't created for their level of expertise, and in the end, these participants were still able to give some ideas and give some valuable impressions. And it turned out that they were a good backdrop of which the study was then able to contrast them against programmers who didn't have as much experience or knowledge.

It should also be noted that there was

indeed an issue with having such a small group involved in the usability study. Results from twenty or thirty students, especially if they could have been in the target group, would have made the weaknesses and the strengths of the project stand out a lot more clearly than they do with only five participants.

10.Future Work

Here, at the end of the project, there are some possible areas that, if time allowed, the project could be taken in and enhanced upon.

Firstly, the logic behind the program could be implemented as a full fledged parser and compiler, which could take any combination of pseudo code created by the user and translate it effectively to give appropriate responses back in return.

Secondly, a user could be allowed to choose the language that they were impartial to, so if a user chose C, the terminology and phrases on the buttons could be more reflective of that programming language. And the buttons would change depending on the coding language chosen by the user.

Thirdly, a tutorial could be incorporated into the program before the user ever got to the making of pseudo code. A tutorial would let the user first become familiar with the wording that the program is offering them to work with. Through the tutorial, they would become comfortable with the meanings behind the words and phrases being used within the scope of the program. The tutorial could describe what the various phrases mean, show some examples using those phrases, and so on, to smoothly adjust the user into the program. This would be for users that might have quite a bit more experience and might therefore need to be trained in to the boundaries of the program to have them work effectively with the program. It might be worthwhile to note though that this would be to make the program more accessible to more experienced programmers, but since these are not the group that the program was intended for, in the first place, this idea may be unnecessary or outside the scope of such as program as Edu-List.

11.Conclusion

The project Edu-List was meant to be an educational tool primarily used for learning about linked lists, by implementing a constructivist approach, where the user would take the base of knowledge that they already had about linked lists, add to it by using the program, and by interacting with the program, students would build a new mental model of the data structure in their mind.

Edu-List features an interactive environment using an animated image, to be displayed at the top of the screen, to clarify the different concepts, which the data structure of linked lists is built from.

The premise behind the program was for the user to create the pseudo code for a linked list by him or her self, which would then allow the user to take their newly acquired knowledge and create the actual code that they would need to implement an actual linked list in their own programs. By employing a constructivist approach, one that would take the knowledge that they already had, add to it and develop it, and in the process create a new more complete mental image in their mind, the idea was that the student would then learn the topic more succinctly and completely, better preparing them for the next step on the way to becoming a full fledged computer scientist.

However the design of Edu-List was not only guided by principles from the Education discipline but also by fundamental principles in Human-Computer Interaction, such as the principles of User Centered Design [10], as well as low-fidelity prototyping techniques [7],[8], which were used to bolster the soundness of the concepts before the actual computer implementation of the Edu-List program ever occurred.

A usability study was preformed, where the participants included both undergraduate and graduate students. The graduate students were included because they were teacher advisors for the first year computer science classes, and the hope was that they would have unique insight into the difficulties of teaching a complicated topic like linked lists to beginner programmers.

The results that followed from this collection of participants ended up showing an interesting division. There were participants that really liked the program. They liked the graphics, they liked the set up of the main screen, and they thought the program was easy to use.

On the other hand, there were participants who really struggled with the program. They found Edu-List constricting and confusing.

One of the reasons for the difference between the responses was the level of experience that the participant brought with them to the study. It was found to be similar to a triangle, with the wide base at the top and the sharp edge at the bottom. The more experienced participants had a hard time fitting all their wide base of knowledge into the ten buttons that were given to them in the program to work with, even

when they knew they were supposed to be looking at the program as if they were beginner programmers. In that way, the program really seemed to get in their way.

For the more junior participants, the number of buttons was a more manageable size for them and the terminology or phrasing on the buttons they found to be easy to make pseudo code with. So then, for some participants, they were able to work effectively with the program.

12. Appendix A

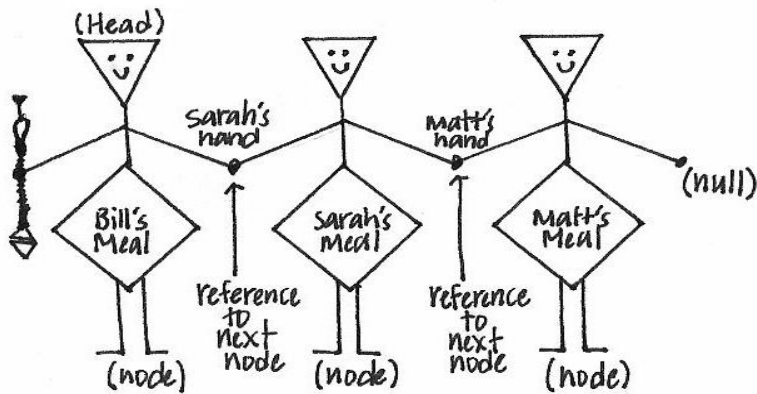


Figure 4: The Triangularian Metaphor

Figure 4 shows the aliens that were going to be used in the original metaphor for the program.

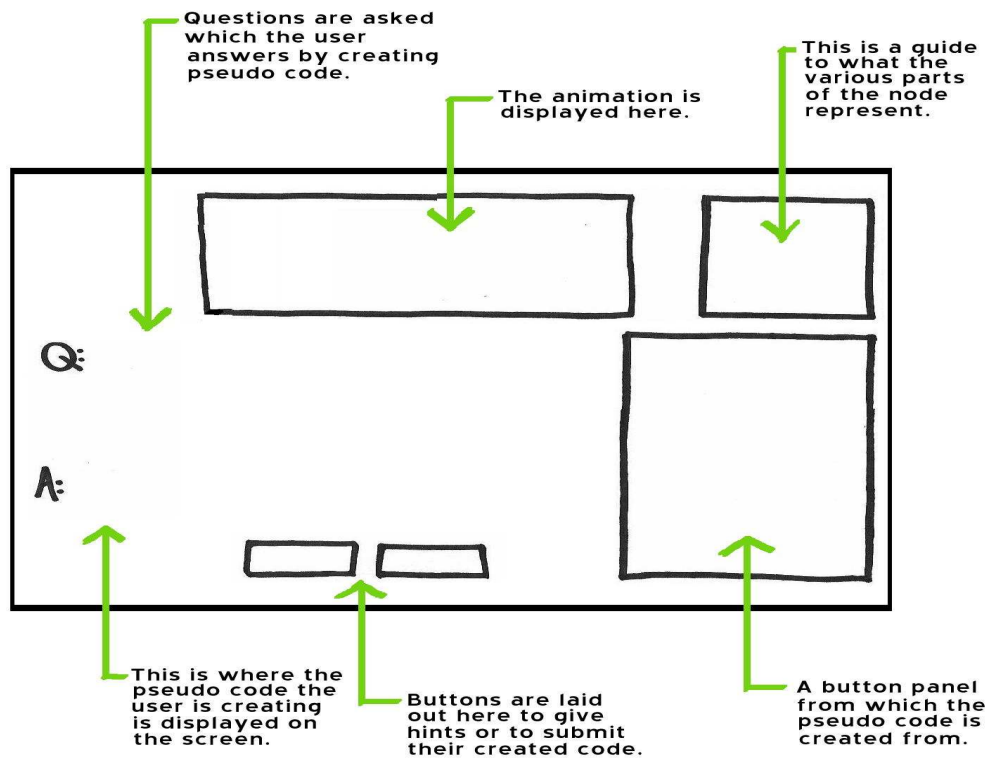


Figure 5: The General Layout of the Low Fidelity Prototypes

Figure 5 shows just the general layout that the low fidelity prototypes took in the initial stages of the project.

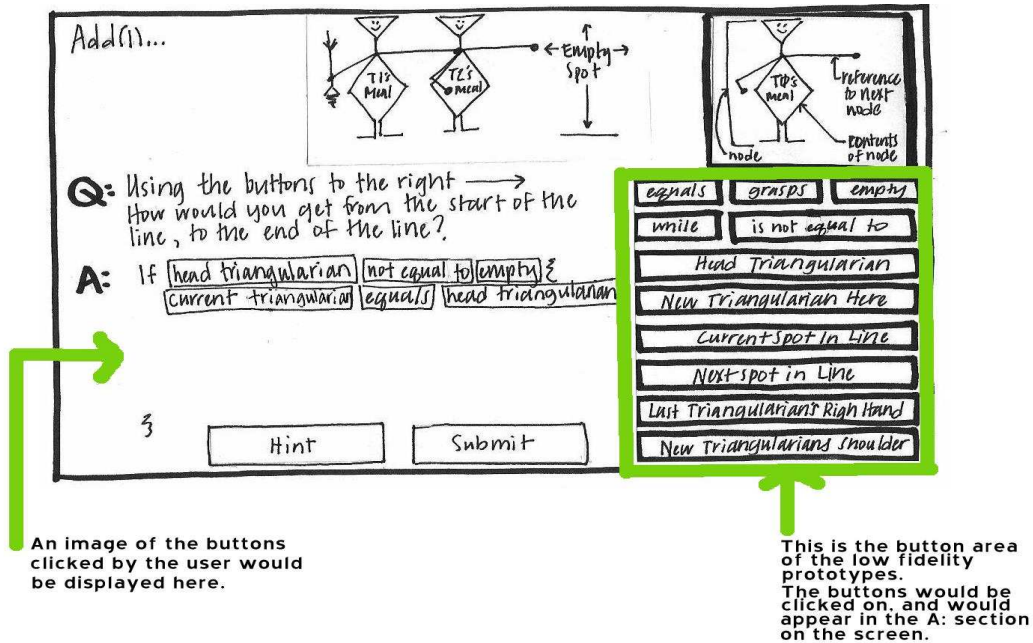


Figure 6: The button and answer areas to be used to create pseudo code by the user

The images below (figures 7-11) are the low fidelity prototypes used to create a Triangularian (alien) based simulation of the “add” functionality of the intended program.

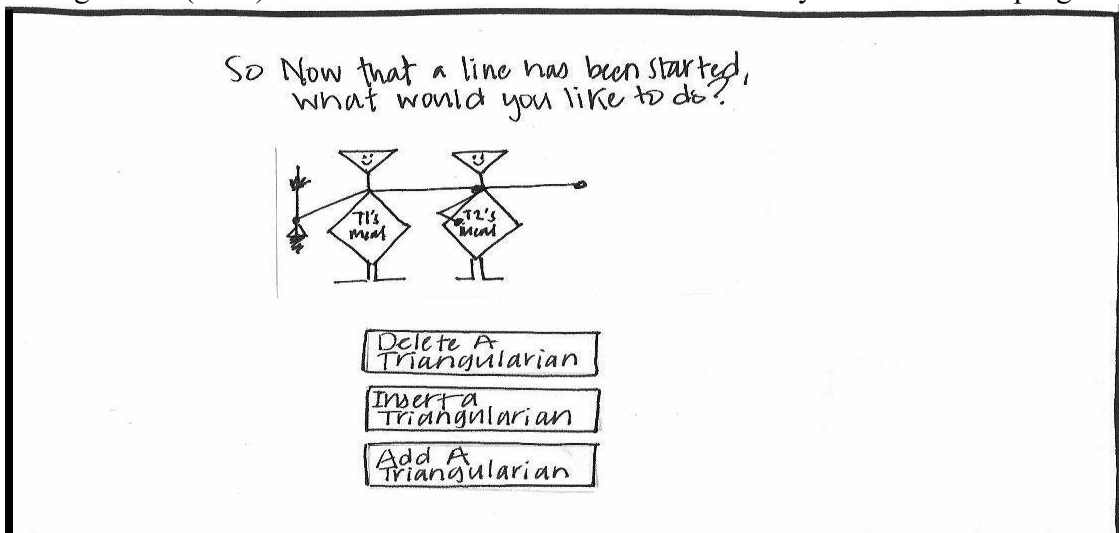


Figure 7: The Decision Screen

The program would start off with an introductory screen that provides a set of three

options for the user: delete, insert and add (Figure 7).

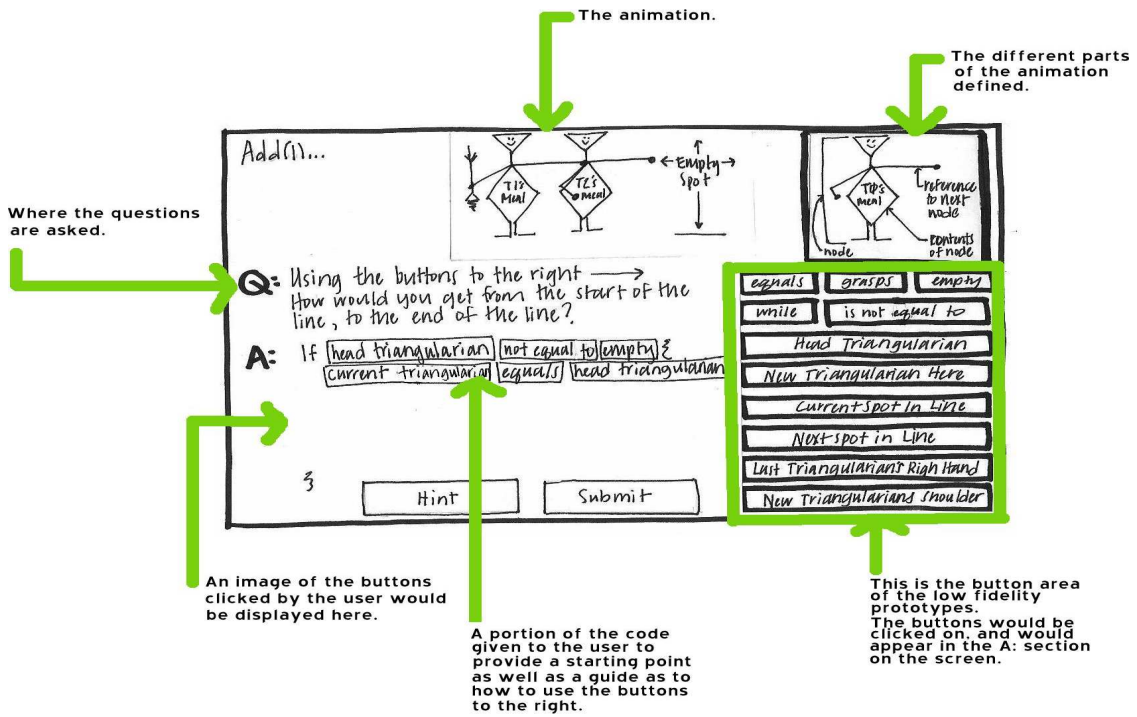


Figure 8: The First of the Triangularian Add Screens with Comments

Figure 8 shows a breakdown of the various sections on the main screen.

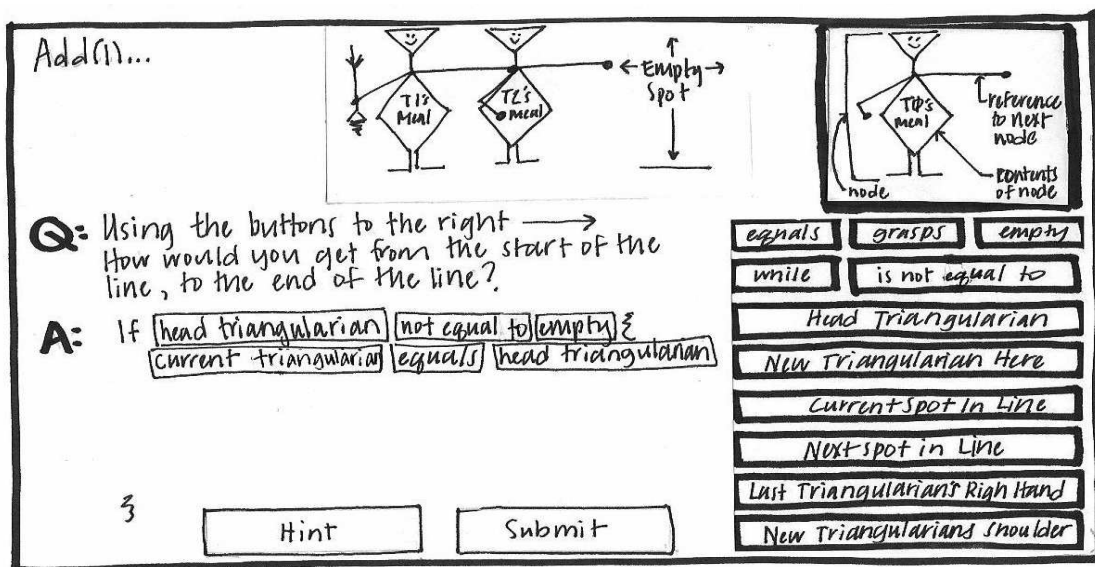
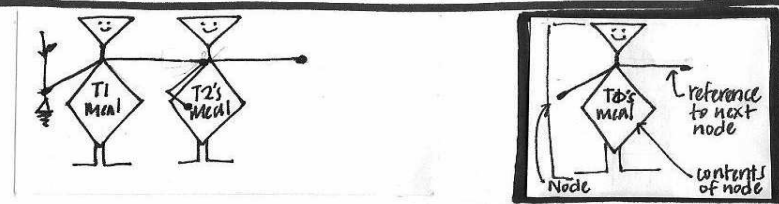


Figure 9: The First of the Triangularian Add Screens without Comments

Figure 9 shows the first question and the starter pseudo code that would be given to the user when attempting to create pseudo code for adding a node into the list.

Add (2)...



Q: What buttons would you use to Add a Triangularian to the end of the line?

A: while current spot in line not equal to empty {
current spot in line equals next spot in line
}

3

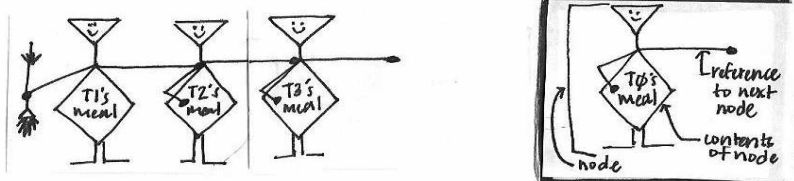
3

Hint Submit

Figure 10: The Second of the Triangularian Add Screens

Figure 10 is the second question and the same starter code that was given in the first question. The idea of the starter code is to give the user an idea of what their pseudo code should look like, and to give them a place to start.

Add (3)...



And the results of your code is....

If head triangularian not equal to empty {
current spot in line equals head triangularian
while current spot in line not equal to empty {
current spot in line equals next spot in line
}
}
current spot in line equals add a new triangularian here
last triangularian's right hand grasps new triangularian's shoulder
}

3

Figure 11: The Third and Last of the Triangularian Add Screens

13. Appendix B

The images below, (figures 12-16), are the low fidelity prototypes used to create just a simple Node based simulation of the “add” functionality of the intended program.

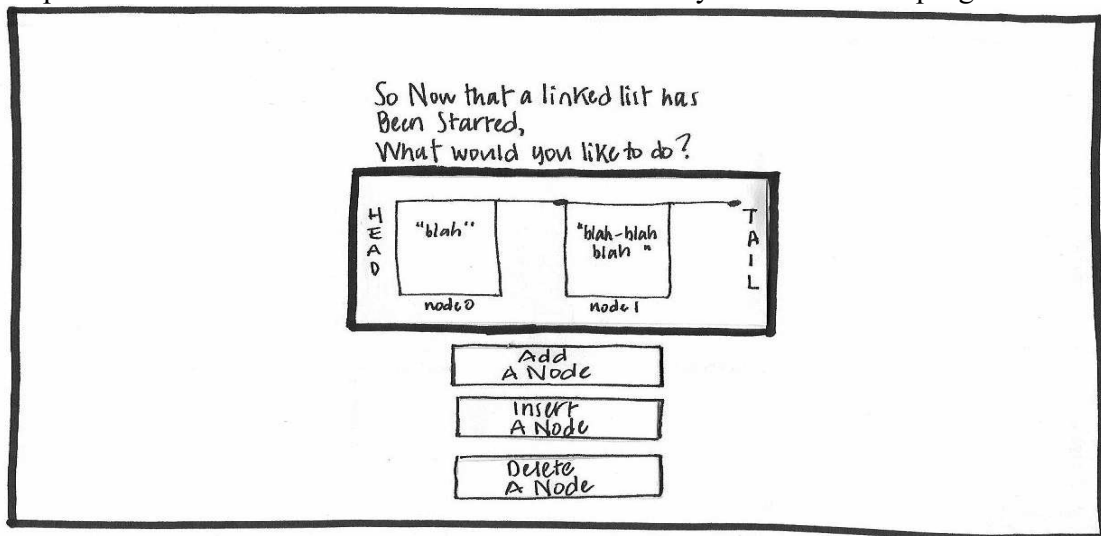


Figure 12: The Decision Screen

As in the original metaphor, the program would start off with an introductory screen that provides a set of three options for the user: delete, insert and add (Figure 12).

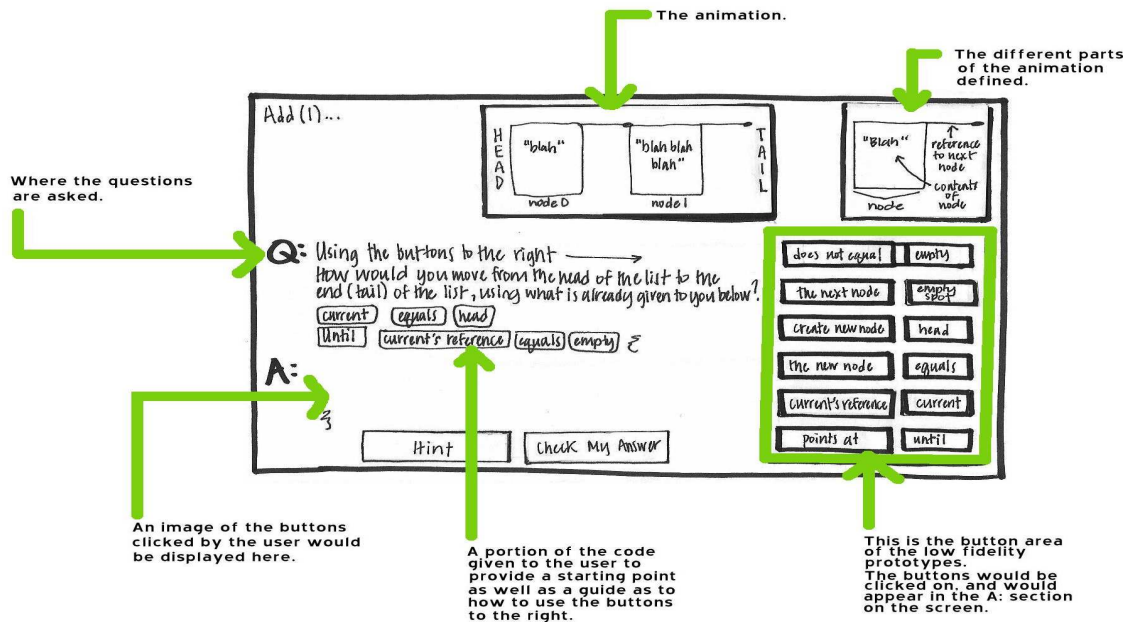
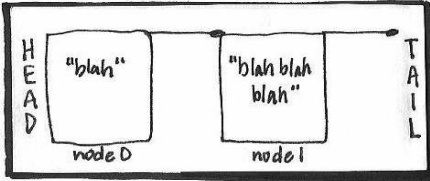
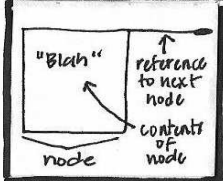


Figure 13: The First of the Node Add Screens with Comments

Figure 13 shows the breakdown of the various sections of the intended main screen.

Add (1)...

Q: Using the buttons to the right →
How would you move from the head of the list to the end (tail) of the list, using what is already given to you below?

current equals head
until current's reference equals empty

A:

3

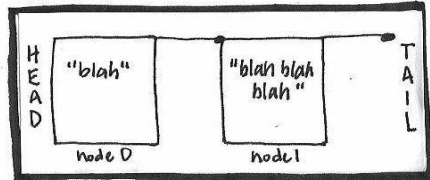
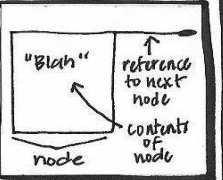
Hint Check My Answer

does not equal	empty
the next node	empty spot
create new node	head
the new node	equals
current's reference	current
points at	until

Figure 14: The First of the Node Add Screens without Comments

Figure 14 shows an early version of the question and starter code to be given for the first “Add a Node” functionality.

Add (2)...

Q: Now you are at the end (tail) of the line, which means that you are currently at node 1. How would you add a new node?

current equals head
until current's reference equals empty/null
current equals the next node

A:

3

current equals

Hint Check My Answer

does not equal	empty
the next node	empty spot
create new node	head
the new node	equals
current's reference	current
points at	until

Figure 15: The Second of the Node Add Screens

Figure 15 is very similar to figure 14, but has the second intended question for “Add a Node” functionality, and again has the starter code there as well.

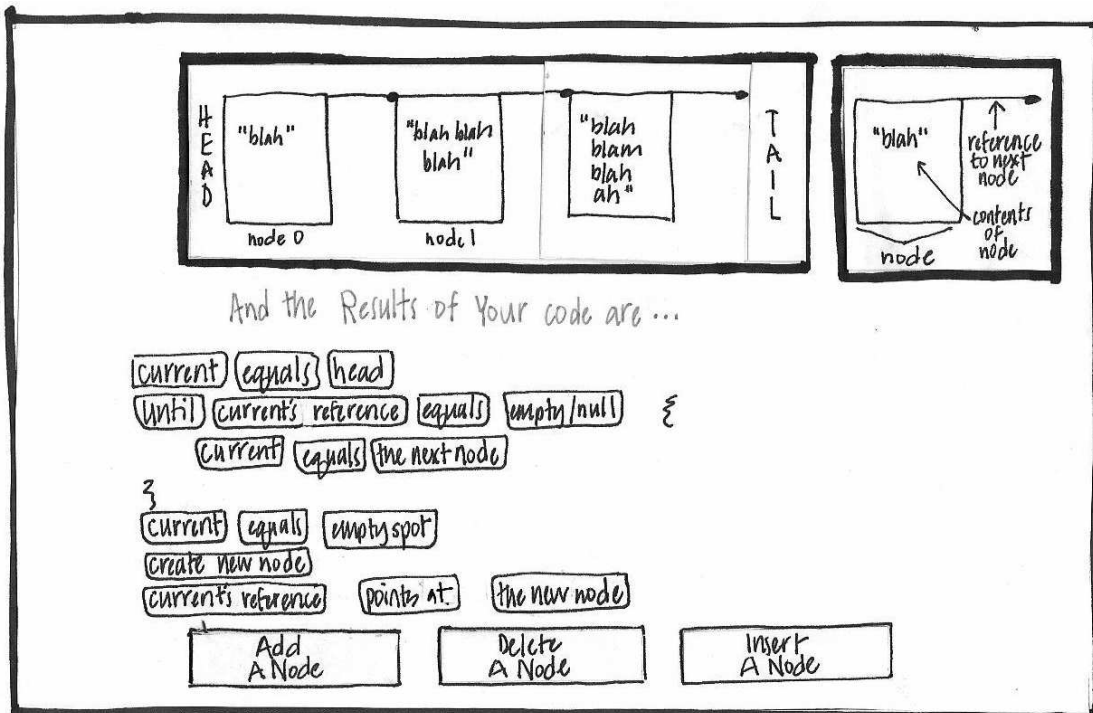


Figure 16: The Third and Last Node Add Screen

Figure 16 is what the final screen was intended to look like, complete with an updated view of the linked list.

14. Appendix C

Working with paper, pencils, pens and sticky notes, an initial layout was created to start working on a low fidelity prototype of the actual program. The basic design was again as follows (figure 17):

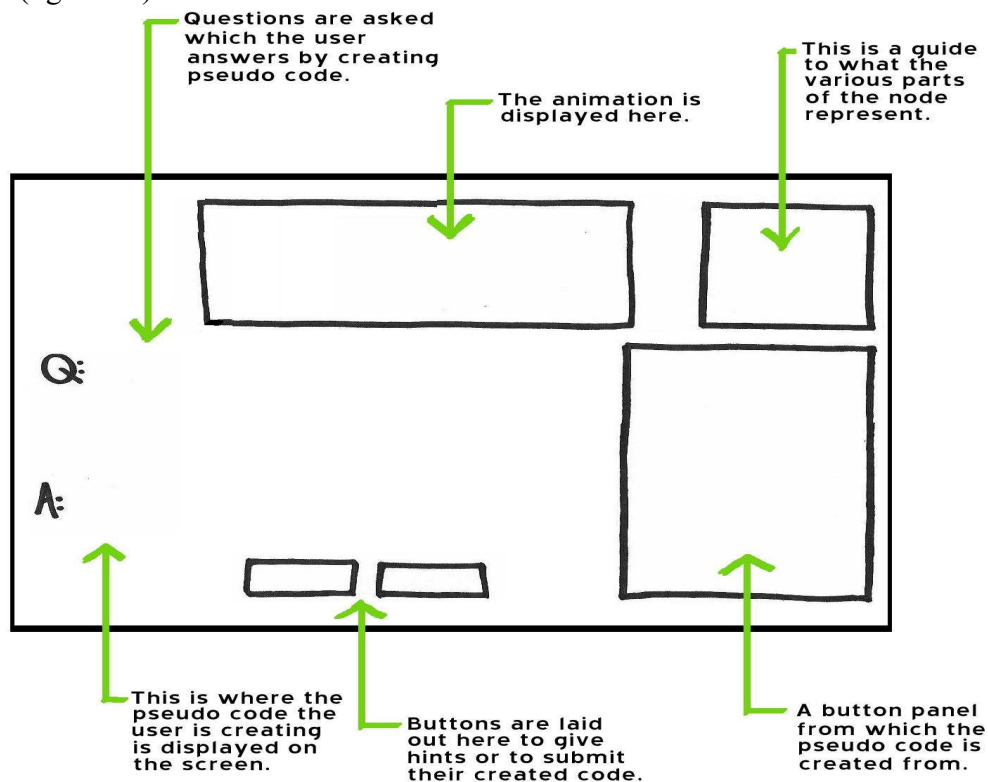


Figure 17: The general design of the low fidelity prototypes

As the user would work through the questions pertaining to either adding, inserting, or deleting a node, the code that they would generate would be built up between the question and answer section, so that at all times they could see how they were progressing and how much they had already accomplished.

15. Appendix D

Figures 18 – 22 show some of the possible messages that might be displayed while the user is making pseudo code.

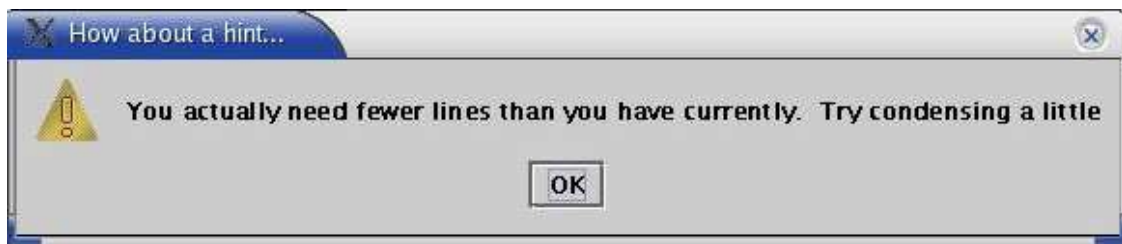


Figure 18: Message that would appear if the user has created too many lines of code



Figure 19: Message that would appear if the user has an incorrect beginning



Figure 20: Message that would appear if the beginning is correct, but there is something incorrect further in the line.

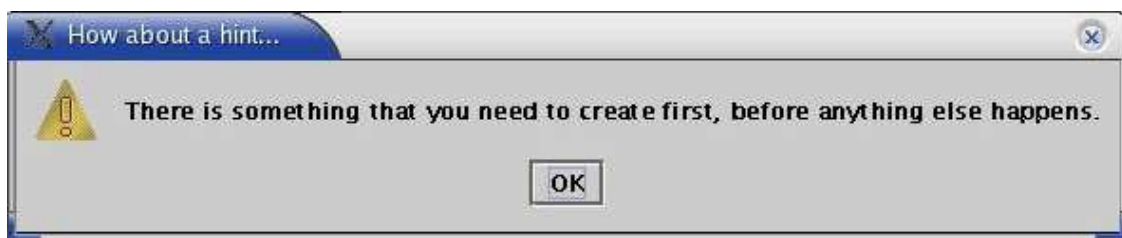


Figure 21: Message that appears if they are supposed to be creating a node for the current question, but they have not done so.

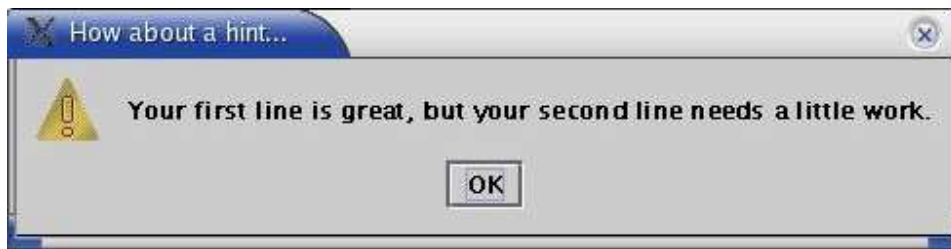


Figure 22: Message that would appear if their first line is correct, but their second line is not.

There are others, of course, but this gives you a good idea of what the messages entail.

16. Appendix E

The following figures (23-35) show the design and the appearance of the add, insert and delete functionality of the program.

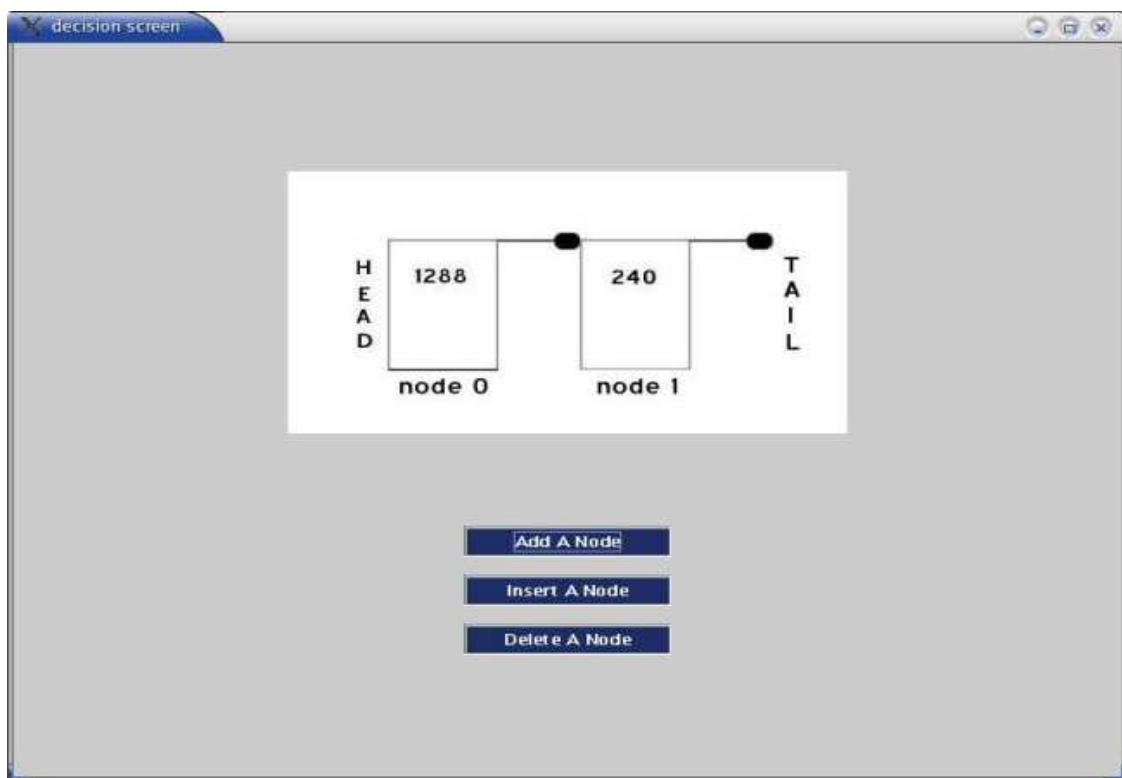


Figure 23: The first screen that user sees upon start-up.

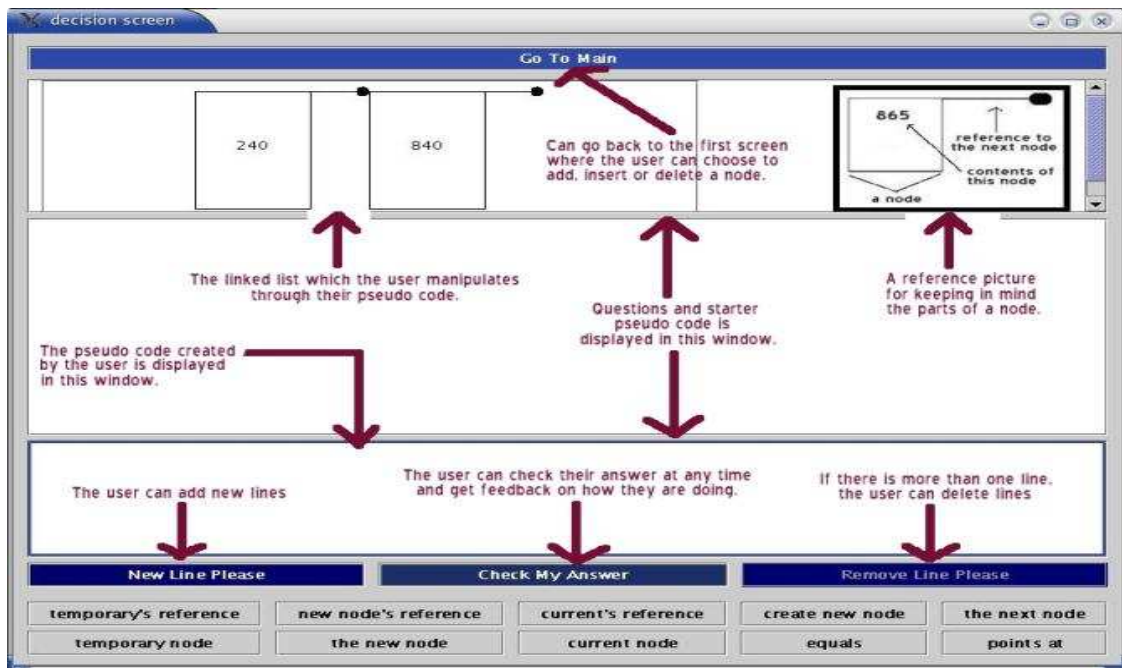


Figure 24: Defining what everything is, on the main screen.

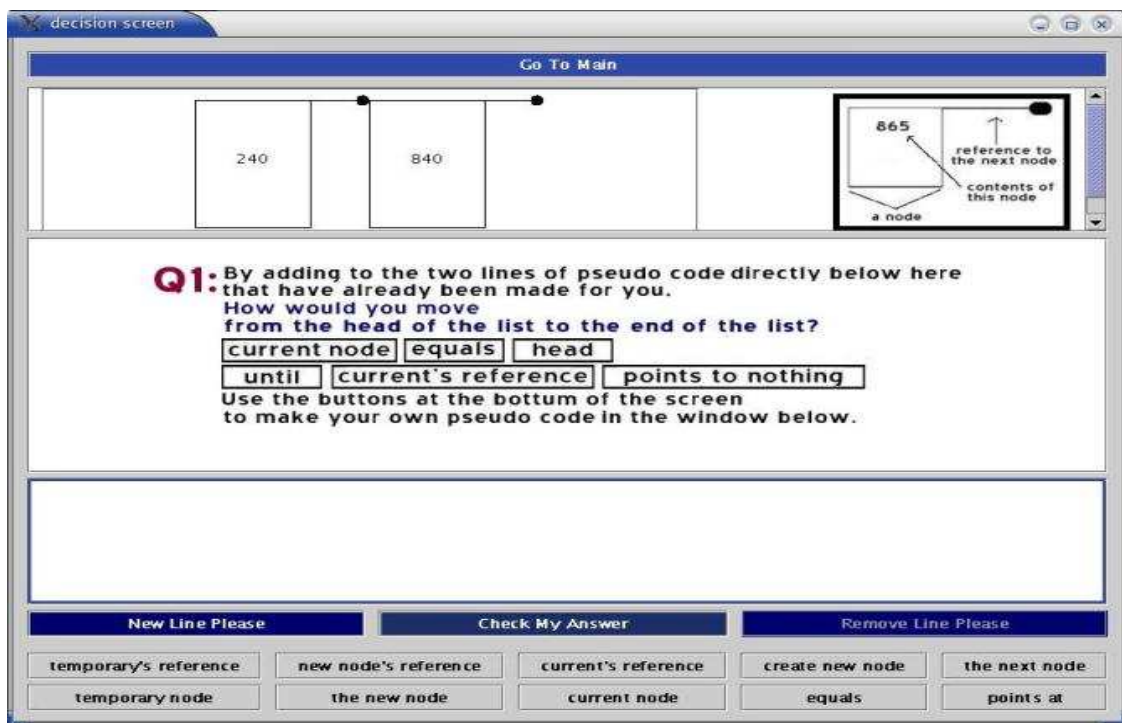


Figure 25: Question One in Add.

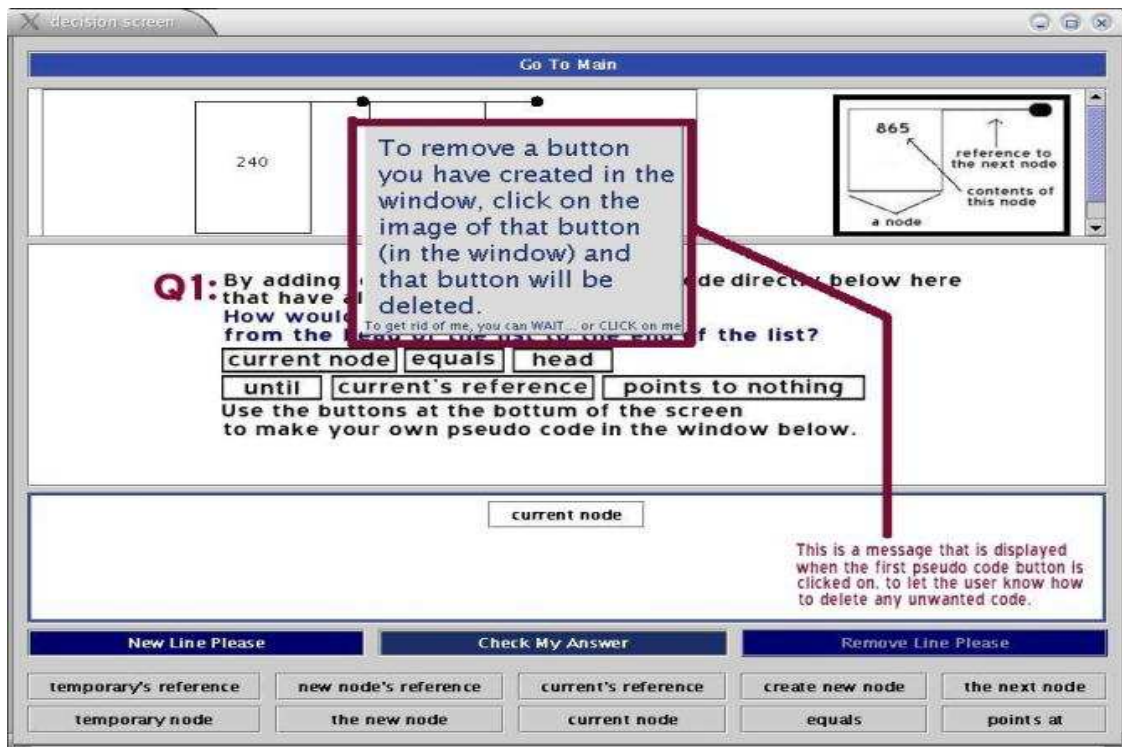


Figure 26: The pop up message describing how to delete pseudo code made by the user.

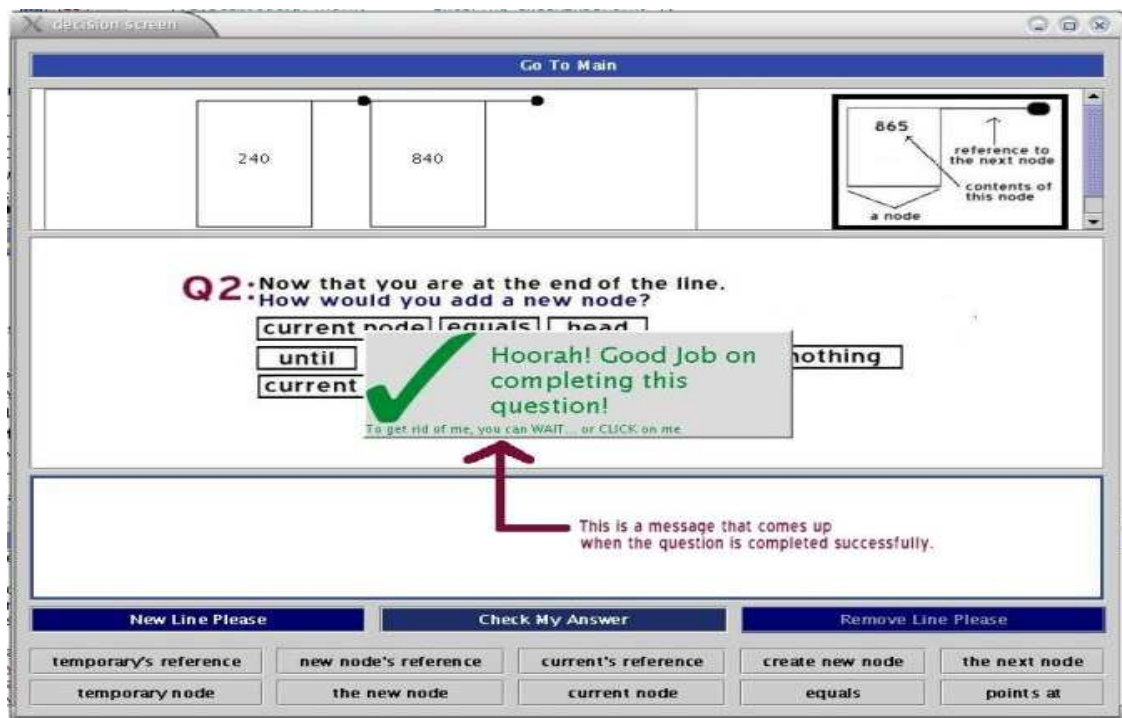


Figure 27: The pop up message for successfully completed questions.

decision screen

Go To Main

240 840

865
reference to the next node
contents of this node
a node

Q2. Now that you are at the end of the line.
How would you add a new node?

current node equals head
until current's reference points to nothing
current node equals the next node

New Line Please Check My Answer Remove Line Please

temporary's reference new node's reference current's reference create new node the next node
temporary node the new node current node equals points at

Figure 28: Question Two in Add with the expected pseudo code.

decision screen

Go To Main

240 840 527

865
reference to the next node
contents of this node
a node

Where there was only two to start with, there is now three, and a new node has been added.

Watch the linked list above
for the results of your code.

current node equals head
until current's reference points to nothing
current node equals the next node
create new node
current's reference points at the new node

Add a Node Insert a Node Delete a Node

temporary's reference new node's reference current's reference create new node the next node
temporary node the new node current node equals points at

Figure 29: The results of answering the add questions correctly.

decision screen

Go To Main

240 840 527

865
reference to the next node
contents of this node
a node

Q1: By adding to the two lines of pseudo code directly below here that have already been made for you. Using temporary node and current node, how would you move from the head of the list to the spot where you wish to add a node into the list?

temporary node equals head
current node equals head
until current node points at the node at desired spot

Use the buttons at the bottom of the screen to make your own pseudo code in the window below.

temporary node equals current node

current node equals the next node

New Line Please Check My Answer Remove Line Please

temporary's reference new node's reference current's reference create new node the next node
temporary node the new node current node equals points at

Figure 30: Question One in Insert with the expected pseudo code.

decision screen

Go To Main

240 840 527

865
reference to the next node
contents of this node
a node

Q2: Now that you are at the desired spot in the list. How would you add a new node in?

temporary node equals head
current node equals head
until current node points at the node at desired spot
temporary node equals current node
current node equals the next node

create new node

temporary's reference points at the new node

new node's reference points at current node

New Line Please Check My Answer Remove Line Please

temporary's reference new node's reference current's reference create new node the next node
temporary node the new node current node equals points at

Figure 31: Question Two in Insert with the expected pseudo code.

Go To Main

240 359 840 527

865
reference to the next node
contents of this node
a node

Where there had been three nodes, a new node has been inserted.

Watch the linked list above for the results of your code.

temporary node equals head
current node equals head
until current node points at the node at desired spot
temporary node equals current node
current node equals the next node
create new node
new node's reference points at current node
temporary's reference points at the new node

Add a Node Insert a Node Delete a Node

temporary's reference new node's reference current's reference create new node the next node
temporary node the new node current node equals points at

Figure 32: The results of answering the Insert questions correctly.

Go To Main

240 359 840 527

865
reference to the next node
contents of this node
a node

Q1. By adding to the two lines of pseudo code directly below here that have already been made for you, How would you move from the head of the list to the desired spot where you wish to delete a node from the list?

temporary node equals head
current node equals head
until current node points at the node at desired spot
Use the buttons at the bottom of the screen to make your own pseudo code in the window below.

temporary node equals current node
current node equals the next node

New Line Please Check My Answer Remove Line Please

temporary's reference new node's reference current's reference create new node the next node
temporary node the new node current node equals points at

Figure 33: Question One in Delete with the expected pseudo code.

decision screen

Go To Main

240 359 840 527

865
reference to the next node
contents of this node
a node

Q2: Now that you are at the desired spot in the list.
How would you remove a node from the list?

current node equals head
until current node points at the node at desired spot
temporary node equals current node
current node equals the next node

current node equals the next node

temporary's reference points at current node

New Line Please Check My Answer Remove Line Please

temporary's reference new node's reference current's reference create new node the next node
temporary node the new node current node equals points at

Figure 34: Question Two in Delete with the expected pseudo code.

decision screen

Go To Main

240 359 527

865
reference to the next node
contents of this node
a node

Where there had been four nodes, now there are only three, and a node has been deleted.

Watch the linked list above for the results of your code.

temporary node equals head
current node equals head
until current node points at the node at desired spot
temporary node equals current node
current node equals the next node
current node equals the next node
temporary's reference points at current node

Add a Node Insert a Node Delete a Node

temporary's reference new node's reference current's reference create new node the next node
temporary node the new node current node equals points at

Figure 35: The results of answering the Delete questions correctly.

17. Appendix F

Edu-Link Questionnaire:

General:

1. How did you learn about linked lists? (Check as many as apply)
 - In a lecture
 - Through the course of an assignment
 - I never have gotten the hang of linked lists
 - I drew it out on paper
 - I wrote code for it and ran my code
 - I got someone else to explain it to me
 - I did research on the internet
 - Other:

2. What would you say was/is the hardest thing to understand about linked lists?

About the program:

3. Did you find the wording on the buttons to be: (check as many as apply)
 - Adequate
 - Inadequate
 - Clear
 - Confusing
 - Easy to make pseudo code with
 - Hard to make pseudo code with
 - Took some getting used to
 - Took no time at all to be comfortable with
 - Didn't have all the words/phrases that I would have liked.
 - Had too many words/phrases to choose from
 - Other:

4. In the program, what area, if any, needed improvement?

5. In the program, what if anything, did you like best about the program?

6. Did you find any part of the program confusing

- No Yes, because

7. Did you find the program: Extremely Hard
 Easy to use Moderate to Use
 1 2 3 4 5
 Reason for Your Answer:

8. Do you think that the program would help beginner programmers learn about
 linked lists?
 Strongly Agree Moderately Agree Strongly Disagree
 1 2 3 4 5
 Reason for Your Answer:

9. Do you think it would be easy to take the pseudo code the user creates in the program and turn it into real code?

Strongly Agree Moderately Agree Strongly Disagree
1 2 3 4 5

Reason for Your Answer:

10. Would this program help to teach beginner programmers about linked lists?

Strongly Agree Moderately Agree Strongly Disagree
1 2 3 4 5

Reason for Your Answer:

18. References

- [1] Ben-Ari, M. (1998). *Constructivism in Computer Science Education*. Proceedings of the Technical Symposium on Computer Science Education. SIGSCE '98 (Atlanta, GA).
- [2] Becker, K and Parker, J.R.. (2003). *Measuring Effectiveness of Constructivist and Behaviorist Assignments in CS102*, Department of Computer Science, University of Calgary.
- [3] Bransford, J and Brown, A and Cocking, R. (1999). *How People Learn: Brain, Mind, Experience, and School*. Washington: National Academy Press.
- [4] Greenberg, S. (2002) *Working Through Task-Centered System Design*. in Diaper, D and Stanton, N. (Eds) *The Handbook of Task Analysis for Human-Computer Interaction*. Lawrence Erlbaum Associates.
- [5] Greening, T and Kay, J. (2001) *Editorial*, *Computer Science Education* 11:3, 167.
- [6] Grissom, S and Van Gorp, M. (2001) *An Empirical Evaluation of Using Constructive Classroom Activities to Teach Introductory Programming*, *Computer Science Education* 11:3, 247-261.
- [7] Nielson, J. (1993). *Chapter 4.8: Prototyping*. *Usability Engineering*. Academic Press (Boston).
- [8] Rettig, M. (1994). *Prototyping for Tiny Fingers*. *Communications of the ACM* 37:4. 21-28.
- [9] Roger, P. (1996) *Teach the Way Most People Learn – The Upside -Down Way*. *Community College Week* 9:8, 4-5.
- [10] Soderston and Rauch (1996). *The Case for User-Centered Design*. Proceedings of the Society of Technical Communication.
- [11] Wilson, M. *Animated Data Structures: the Linked List*. (2001)
<http://www.cs.stir.ac.uk/~mew/dissertation/simulation.htm>.