MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CENG 491
SENIOR DESIGN PROJECT
INITIAL DESIGN REPORT
FALL 2007

Group Name: Hellim

Group Members:

Kutay YILDIRICI	e143393@metu.edu.tr	1433937
Tayfun ÇAKICIER	e154349@metu.edu.tr	1543495
Halit Emre SAYILIR	e143385@metu.edu.tr	1433853
Anil KOYUNCU	e143380@metu.edu.tr	1433804

Group Mail: hellim-ltd@googlegroups.com

Project Title: CStar - Optimizing C Compiler

Table of Contents:

1.Introduction:

1.1.Detailed Problem Definition:

The optimization is a very important topic for compilers. Performance is increasingly being important nowadays in the market. An optimized and tuned program can run much more fast and better.

Our first aim is to implement optimizations in order to reduce run time of programs. Also a well developed implementation must not increase the compilation time too much. In order to make good implementations we must develop good and efficient algorithms for optimizations. Also our implementations must not change the functionality of the programs.

We will think in two dimensions for optimizations; time or space. Optimizations like Dead Code Elimination, Constant Folding may increase compile time but will decrease the size of the program. Optimizations like Basic Block Ordering, Strength Reduction will increase the size of program, but can dramatically decrease the run time of the program.

Our second aim is to implement as much as optimizations we can. The increased variety of simple implementations is better than having less number of complex optimizations. Optimizations will be done over intermediate representation of the framework.

In our project we will use the framework QuickC developed by CStar. QuickC is a design by contract framework. Most of the important functions (like creating expression trees, managing optimizations, etc…) are ready for us to use for implementing optimizations. For this project we will implement and add our implementations to the framework.

With the features of the framework, we can call any optimization, any times and any moment during the compilation. For example we can do the Constant Folding at the beginning of the compilation, after we can call the Common Subexpression Elimination optimization, then we can do the Constant Folding optimization again.

Optimizations are made through the “Anatrop” (analysis-transformation-optimization) and “AnatropManager” classes in the framework.

All optimizations have their options that can be set with the “Option” class of the framework. For example we can set properties for optimizations to recursively call themselves again if it had made a change on the program. We can set the number of maximum calls for an optimization.

Also the optimizations can be done in the specified scope like Basic Blocks, Program, Statements, etc…

Rather than optimizations we will also develop an Optimization Manager for the framework. The Optimization Manager will be implemented as an Anatrop in the framework. We will implement the manager in two modes; Interactive Mode and Normal Mode. In Normal mode, the manager will read the names and usages of the optimizations from an external file. In Interactive Mode, we will get the names, scopes and options of the optimizations from the user via standard input. Then we will set the options for optimizations and execute them.

We will also develop a Test Case Generator for the framework. It will get target and option files from user. It will read the options from these files and set these in the framework. Then it will dump generated C Code. At the end these will be sent to output.

1.2.Design Constraints and Limitations:

1.2.1.Constaints:

Experience: Although we have participated in many software projects and homeworks , our current project is harder than them because of new concepts. Our group is getting familiar to framework everyday , but some detailed usages must be examined correctly. Also it can be difficult to handle some unexpected problems about this new concept.

Time: The project must be finished by June and also we should provide at least two of optimizations at the end of this semester. We should use our time efficiently to not fall behind the schedule. So we should follow our works due to our Gantt Chart.It can be seen at table 1.a

Performance: It is important that our optimizations must work efficiently. So we must design and build our optimizations to be work with high performance. Then we should make some testing to confirm. Drystone and wetstone benchmark tools will be useful.

1.2.2.Limitations:

User interface: As known , the compiler has not got a complicated user interface. Everything is handled by some commands which are entered in Linux terminal. Also test case generator is handled by some extra commands and the optimization manager is handled by an external file.

Platform: Our project is set on Linux machines. Also we should make our implementations by framework. There are important classes and functions in framework that should be used or inherited in order to integrate our optimizations to framework and use them.

Language: C++ programming language should be used as all framework , necessary class and functions were written in C++.

2.Architectural Design:

2.1.Class Diagrams:

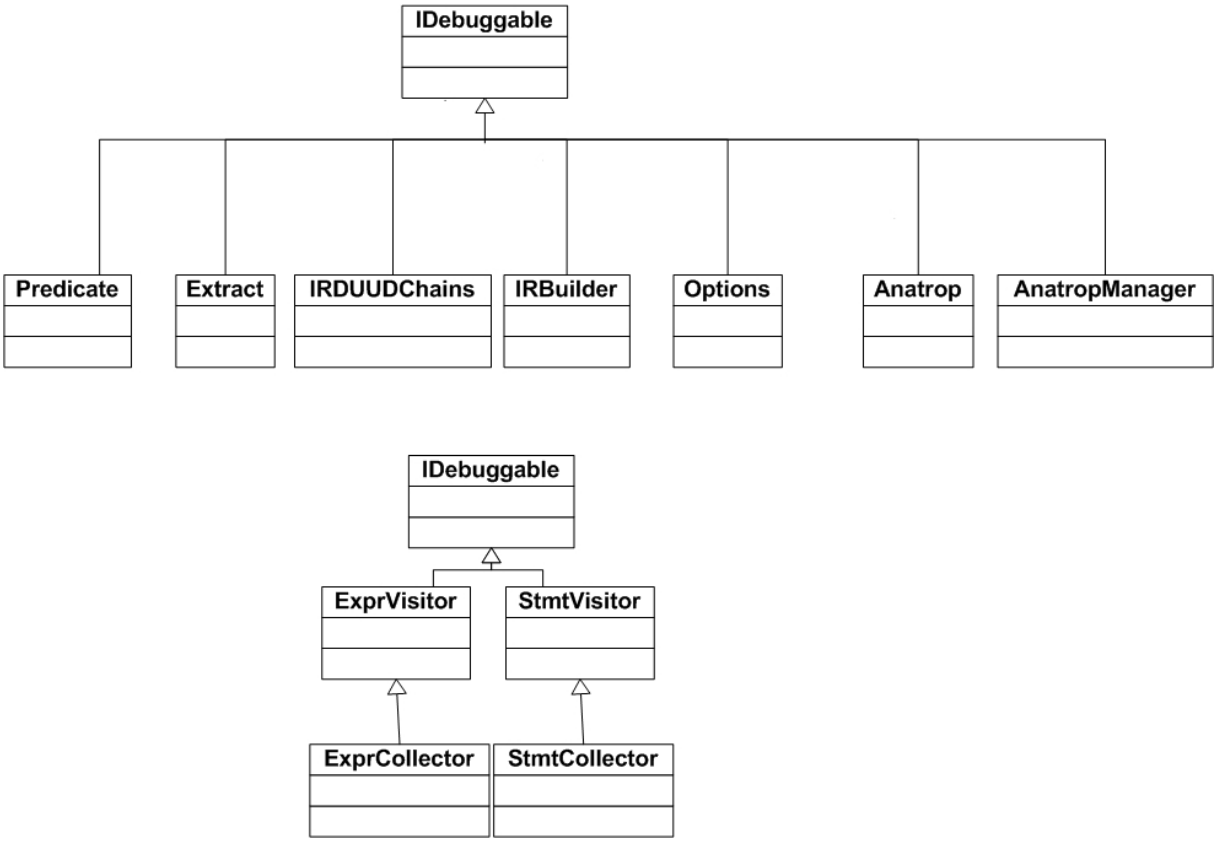

Diagram 1.a) **some classes from framework that we think important**

Diagram 1.b) **some important methods of the given classes**

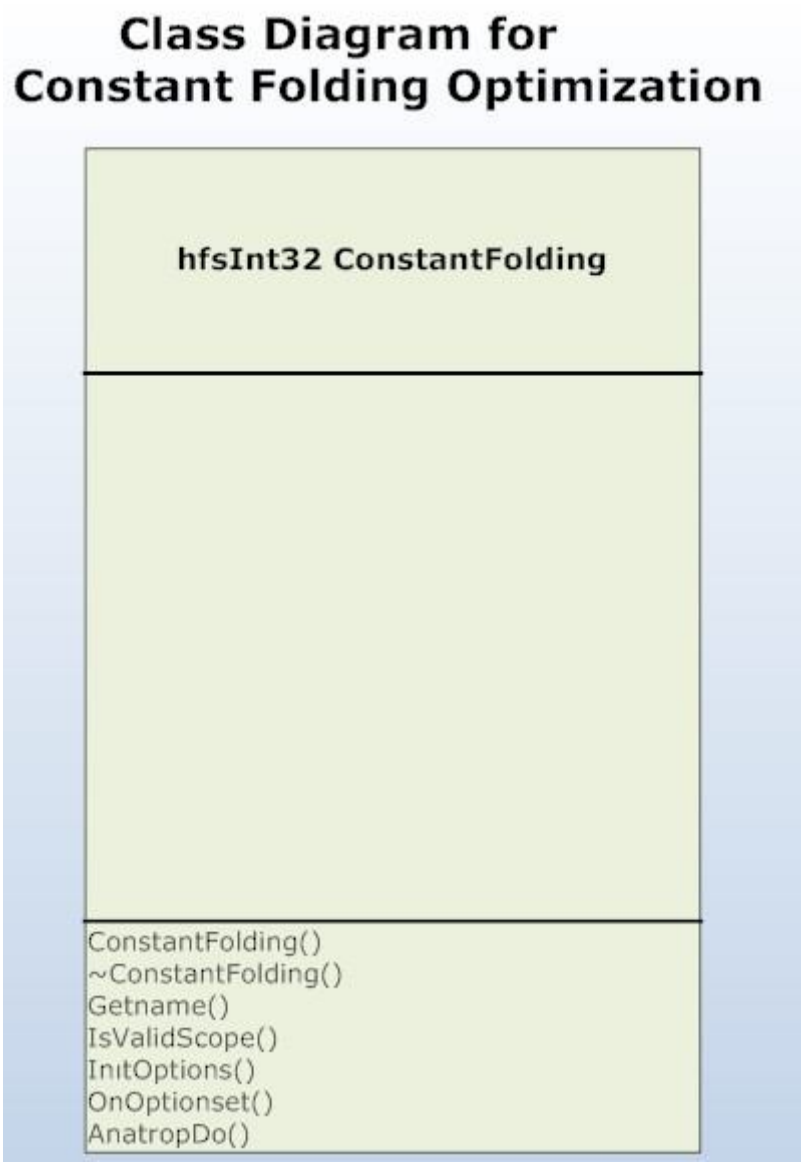

Diagram 1.c) class diagram for constant folding optimization

2.2.Sequence Diagrams:

An example sequence diagram for peephole optimization:

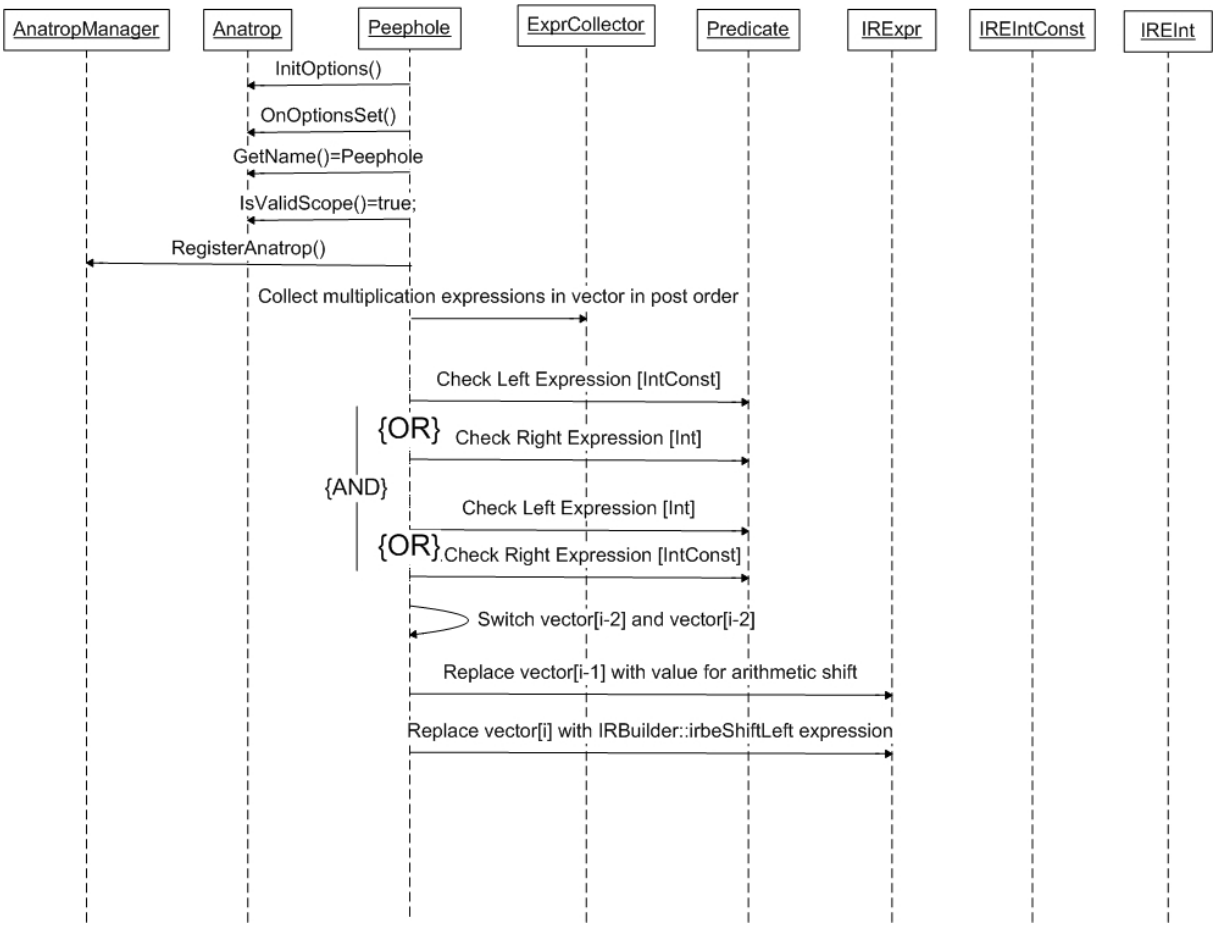

Diagram 2.a) sequence diagram for peephole optimization

Replace multiplication statements like a*2 or 8*b with left arithmetic shifts and integer constants to decrease processor cycles.

An example sequence diagram for constant folding optimization:

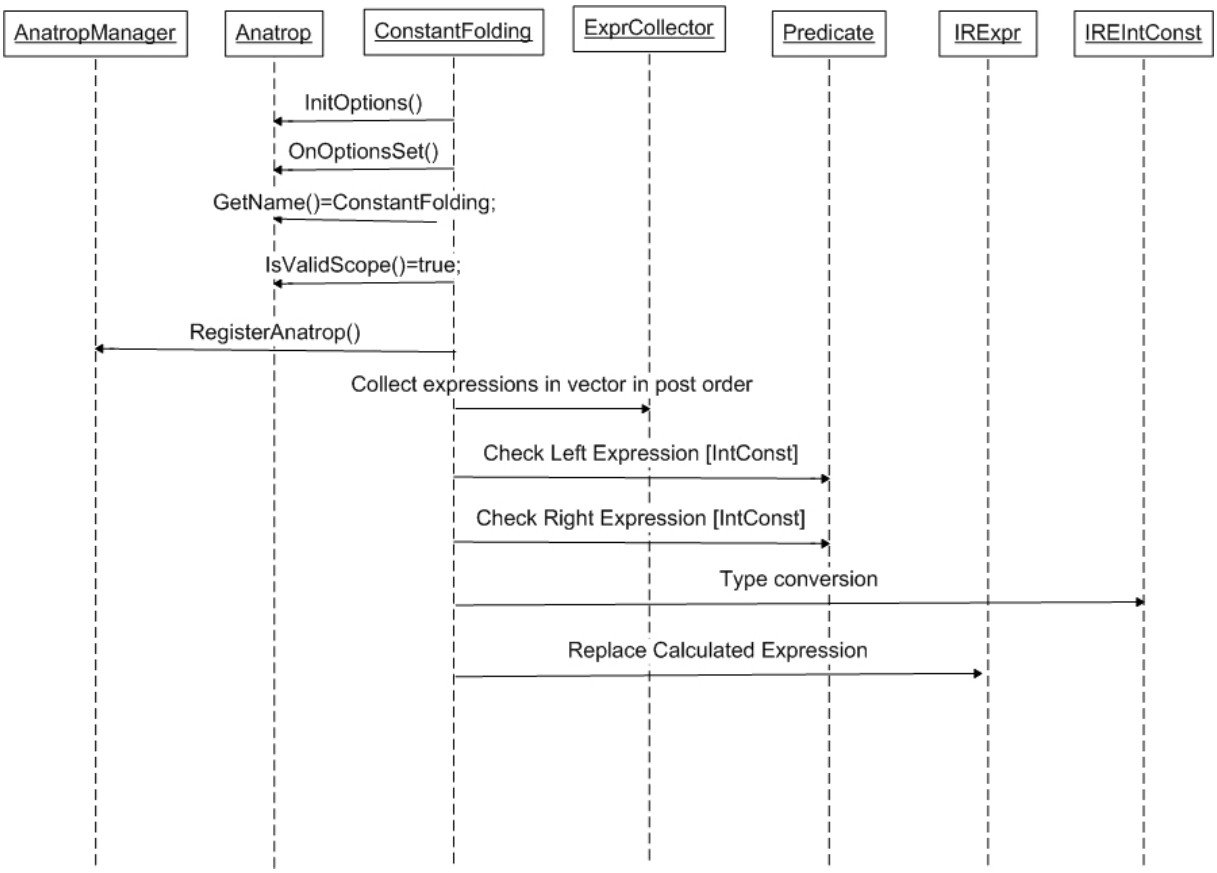

Diagram 2.b) sequence diagram for constant folding optimization

2.3.Use Case Diagrams:

2.3.1.Use Case Diagram of The Test Case Generator:

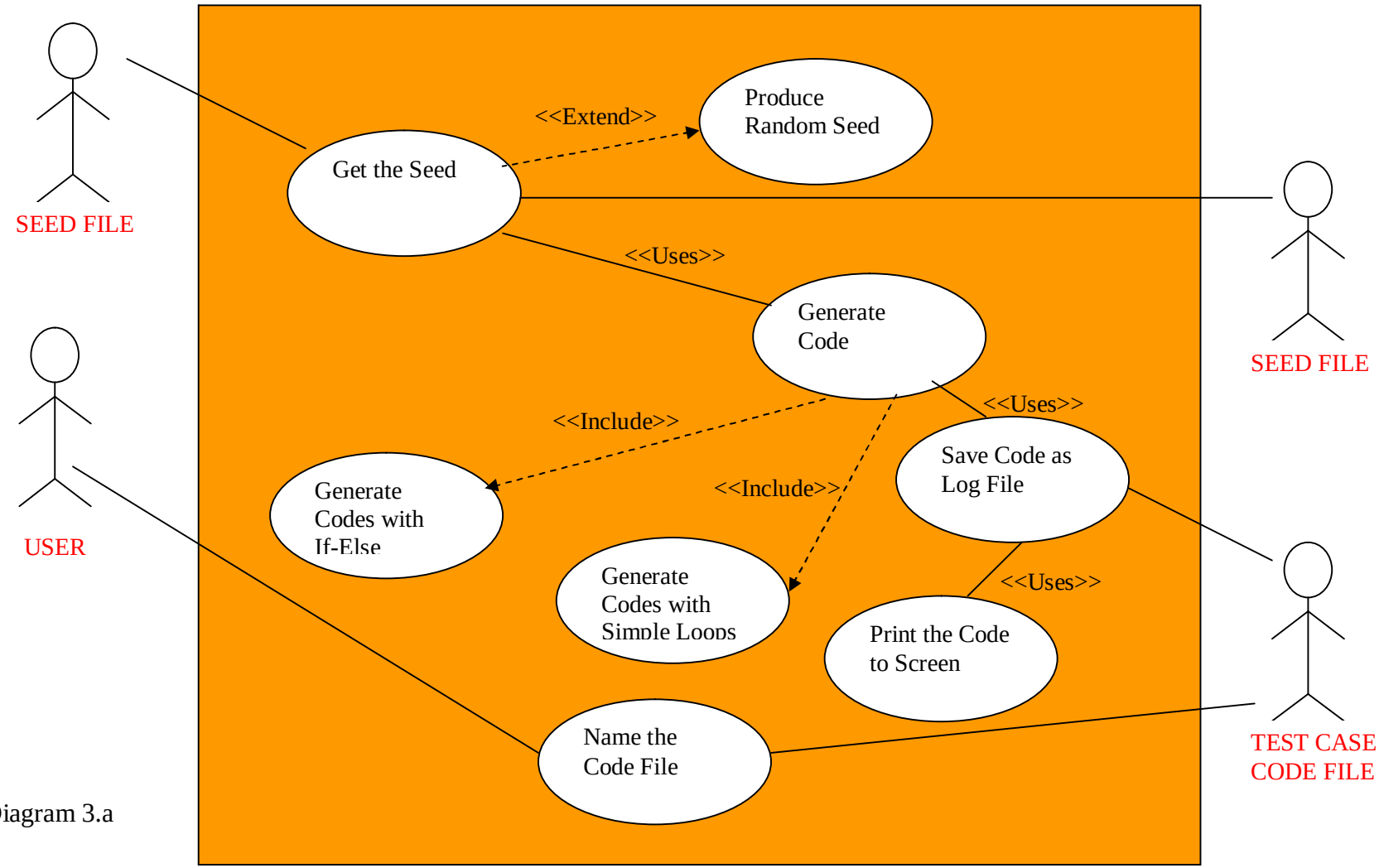

Diagram 3.a

2.3.2.Use Case Diagram of The Optimization Manager:

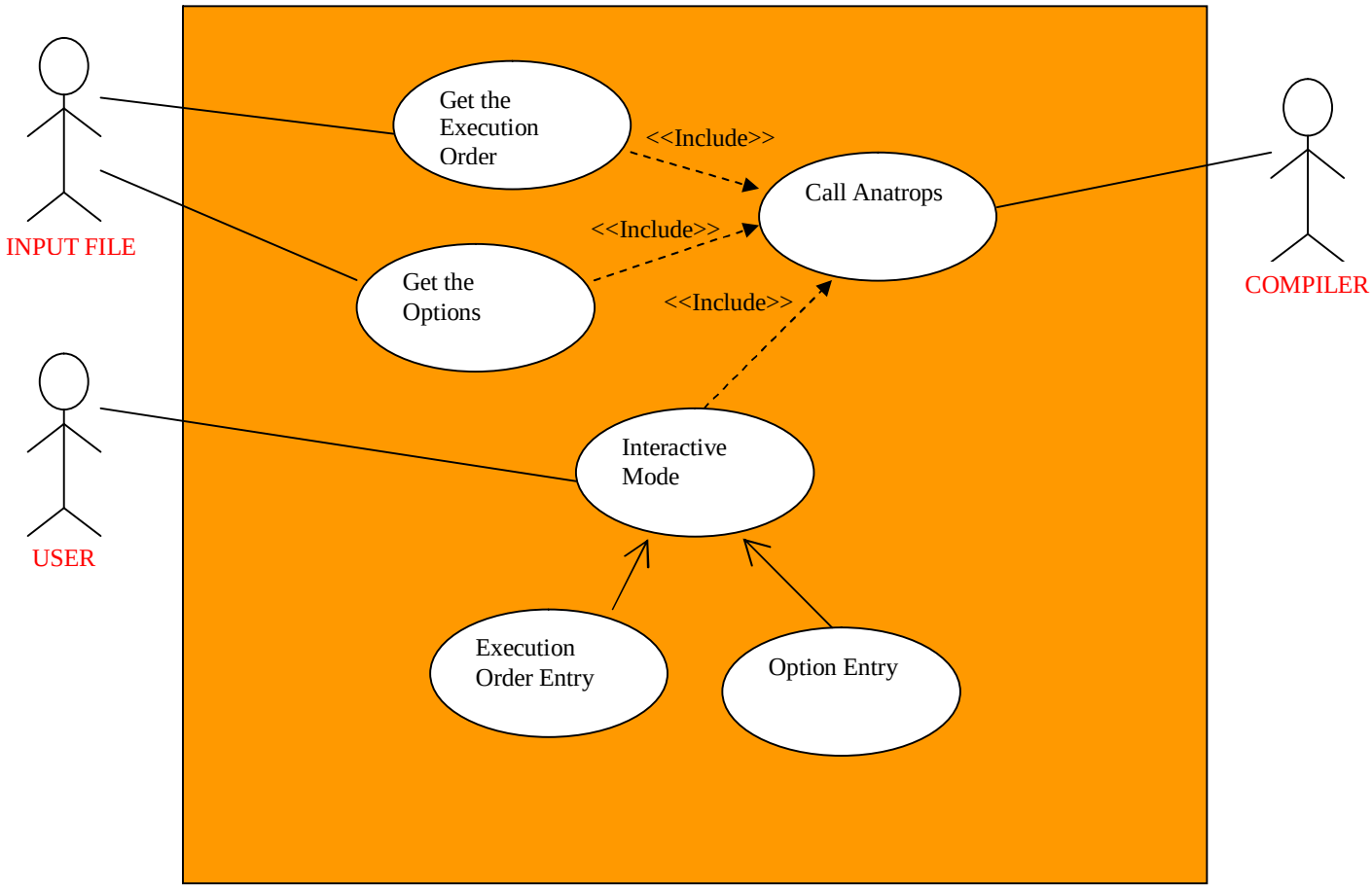

Diagram 3.b

2.3.3.TEST CASE GENERATOR EXPLANATORY TABLES:

Use Case	**Get the Seed**
Purpose	**Taking the Seed from the File or Itself**
Type	**Primary, Real**
Actors	**Seed File**
Description	**Program takes the seed from file or itself**
Cross-References	**Use Case: Produce Random Seed, Generate Code**

Use Case	**Produce Random Seed**
Purpose	**Producing random seeds for the program**
Type	**Primary, Real**
Actors	**None**
Description	**When there is no need for seed file to produce seed it produces seed**
Cross-References	**Use Case: Get the Seed**

Use Case	**Generate Code**
Purpose	**Generating the test case codes**
Type	**Primary, Real**
Actors	**None**
Description	**After getting the seed, it randomly generates statements like if-else and simple loops**
Cross-References	**Use Case: Get the Seed, Generate Code with If-Else, Generate Code with Simple Loops, Save Code to a Log File**

Use Case	**Generate Code with If-Else**
Purpose	**Generating If-Else statements**
Type	**Primary, Real**
Actors	**None**
Description	**According to taken seed it generates If-Else statements**
Cross-References	**Use Case: Generate Code**

Use Case	**Generate Code with Simple Loops**
Purpose	**Generating Simple Loops**
Type	**Primary, Real**
Actors	**None**
Description	**According to taken seed generates simple Loops like for, while, etc.**
Cross-References	**Use Case: Generate Code**

Use Case	**Name the Code File**
Purpose	**Naming the code file as user wish**
Type	**Primary, Real**
Actors	**User, Test Case Code File**
Description	**Specifies the name of the produced code file**
Cross-References	**None**

Use Case	**Save Code to a Log File**
Purpose	**Saving the code to a log file**
Type	**Primary, Real**
Actors	**Test Case Code File**
Description	**Saves the code to a log file**
Cross-References	**Use Case: Print the Code to Screen**

Use Case	**Print the Code to Screen**
Purpose	**Printing the code to screen**
Type	**Primary, Real**
Actors	**None**
Description	**Prints the generated code to screen**
Cross-References	**Use Case: Save Code to a Log File**

2.3.4.OPTIMIZATION MANAGER EXPLANATORY TABLES:

Use Case	**Get the Execution Order**
Purpose	**Specifying the Execution Order**
Type	**Primary, Real**
Actors	**Input File**
Description	**Reads the execution order of the optimizations from an external file**
Cross-References	**Use Case: Call Anatrops**

Use Case	**Get the Options**
Purpose	**Specifying the Options of anatrops**
Type	**Primary, Real**
Actors	**Input File**
Description	**Reads the options of the anatrops from an external file**
Cross-References	**Use Case: Call Anatrops**

Use Case	**Interactive Mode**
Purpose	**Entering order and options by hand**
Type	**Primary, Real**
Actors	**User**
Description	**There is command for changing the mode to interactive. User can enter his own execution order and options in this mode**
Cross-References	**Use Case: Execution Order Entry, Option Entry**

Use Case	**Execution Order Entry**
Purpose	**Entering the execution order of optimizations by user**
Type	**Primary, Real**
Actors	**None**
Description	**User enters the execution orders as he wishes**
Cross-References	**Use Case: Interactive Mode**

Use Case	**Option Entry**
Purpose	**Entering the options of anatrops by user**
Type	**Primary, Real**
Actors	**None**
Description	**User enters the options of anatrops as he wishes**
Cross-References	**Use Case: Interactive Mode**

Use Case	**Call Anatrops**
Purpose	**Calling the anatrops by specified order and options**
Type	**Primary, Real**
Actors	**Compiler**
Description	**It calls the anatrops and run it by specified order and options**
Cross-References	**Use Case: Get the Execution Order, Get the Options, Interactive Mode**

2.3.5.SCENARIOS:

a) Test Case Generator:

Scenario 1: Random seed is produced by the program and code is generated with default name after that saves it to a log file and prints it to screen

Scenario 2: A previous seed is taken from the seed file and code is generated with default name after that saves it to a log file and prints it to screen

Scenario 3: User gives his specific name to the code file

b) Optimization Manager:

Scenario 1: Optimization manager takes the execution order and options from an external input file and calls anatrops

Scenario 2: Optimization manager works at interactive mode user can enter execution order and options by hand and calls anatrops

3.Dataflows:

3.1.Test Case Generator:

Test case generator is an anatrop, so it will be implemented as an anatrop. It has some requirements to be implemented to fulfill its functionality. It will work only for the program scope. it will also be executed by a C file.

We should clear the module of the IR Program. It will be achieved by calling the reset() method of the IRProgram which removes all the modules from the program. We will use the IRBuilder class for implementation of the generator. The options set will be read and its functionalities will be added to the IR.The number of functions, statements and expressions will obey the rules which will be taken from the user with input.

If, for, while, do, assignments, expressions will be generated according to the input taken from user. The input will be specifications for the generation rules; like ranges and probabilities of statements, expressions; number of functions.

In the generator, we will implement the types as rich as possible, like integers, floats, strings, reals, bools, etc.. We will also define expressions defined in the scope of the framework like binary expressions (addition, multiplication, xor, arithmetic shifts, etc..), constants(integer, string...), locals, globals, etc...

We will implement a kind of Bayesian Networks that Mr. Karpat stated. With the use of Bayesian Network, we will implement different probabilities for statements.
For example let's consider the assignment statement; int a = 12.
Here we will look up in the Bayesian Network and create a type of integer according to the probability. Then in the given range we will assign a constant integer with its probability.

We will also define what can be the contents of the context.
At each compilation a single file is created. We will write the created random seed into a file. The seed will also be taken with user input.

In order to be able to write better looking generated code, we will extend the intermediate representation to a higher level or a HIR implemented by the framework in the following days will be used. After the generation of IR, we will dump this IR into C code. This generated C code will be saved to a log file and also it will be printed to the screen.

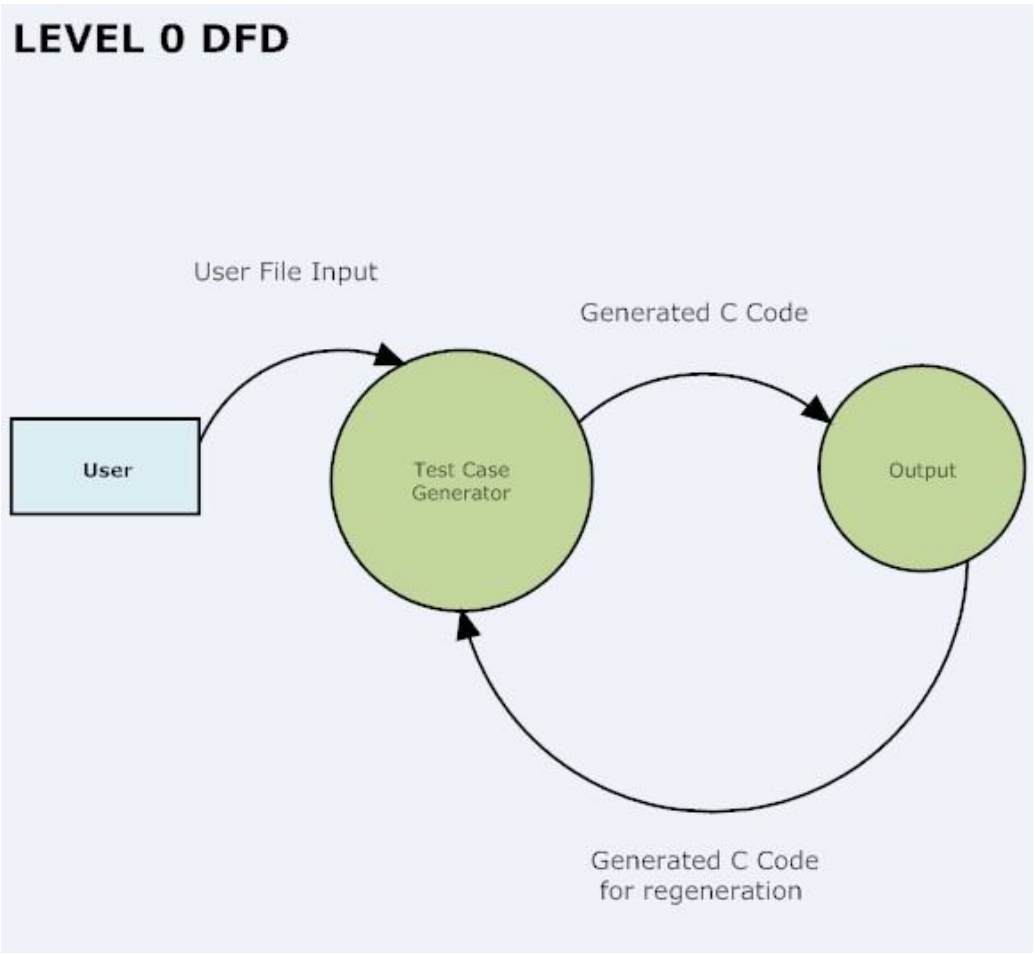

Diagram 4.a

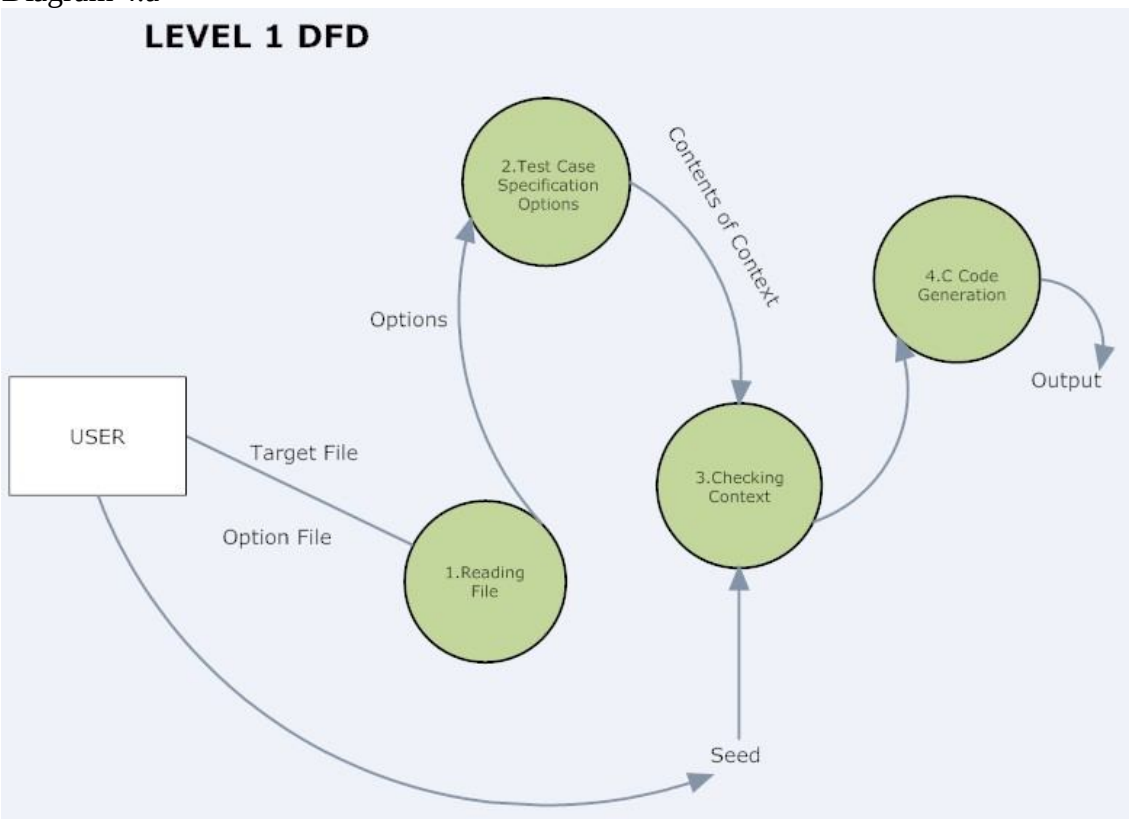

Diagram 4.b

Level 2 DFD

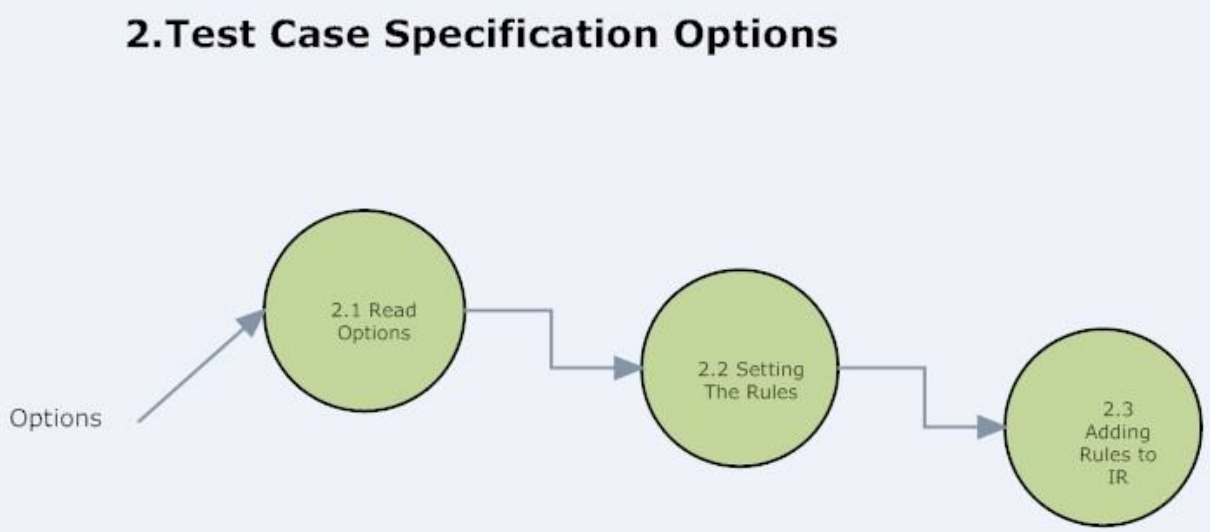

Diagram 4.c

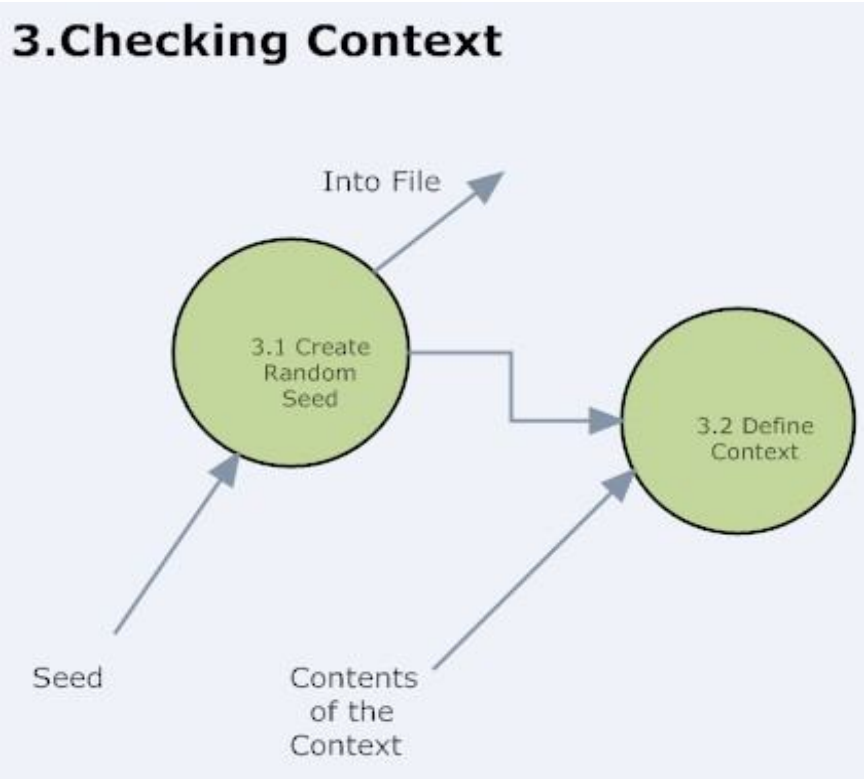

Diagram 4.d

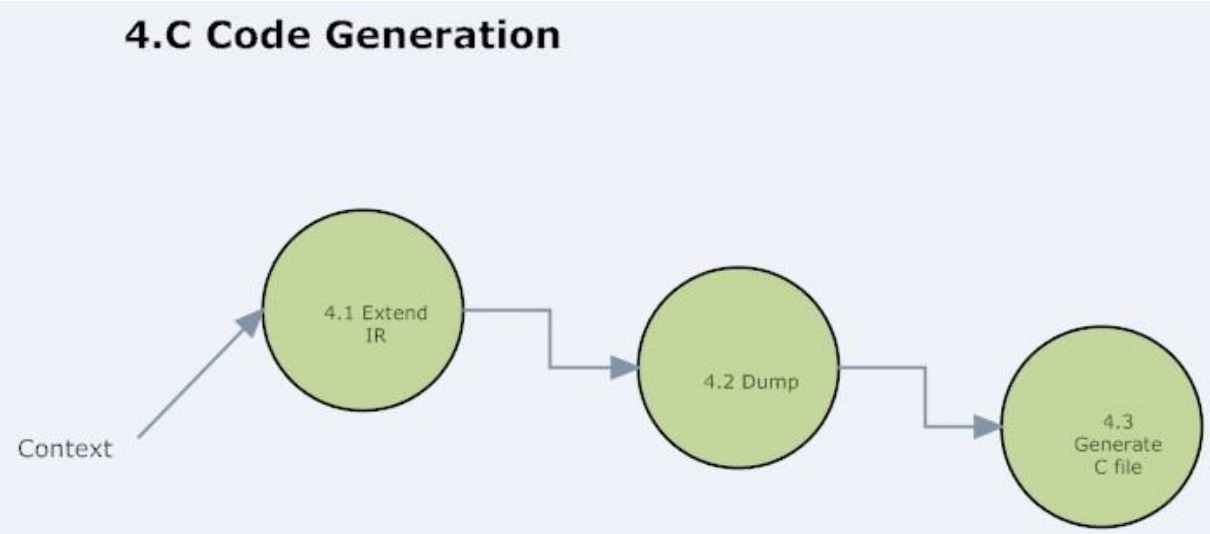

Diagram 4.e

3.2.Optimization Manager:

Actually , the manager is an anatrop in the framework. Basicly , at compilation time an external file will be read or interactively user enters optimizations, scopes and options via standard input and so that the optimizations , which can be in some order or can be integrated to each other, will be chosen. There are two ways to do this. Firstly , user must write in a file with a specific way which optimizations can be an object .Secondly , user enters data with standard input and specify it . Then choosen optimizations will be added to be applied and then they will be dumped and executed.

The user can call the interactive mode via a code like:
ATOOptimizationManager.EnterInteractif();

Or with a command line input like :
-mode:Manager:"IActive”

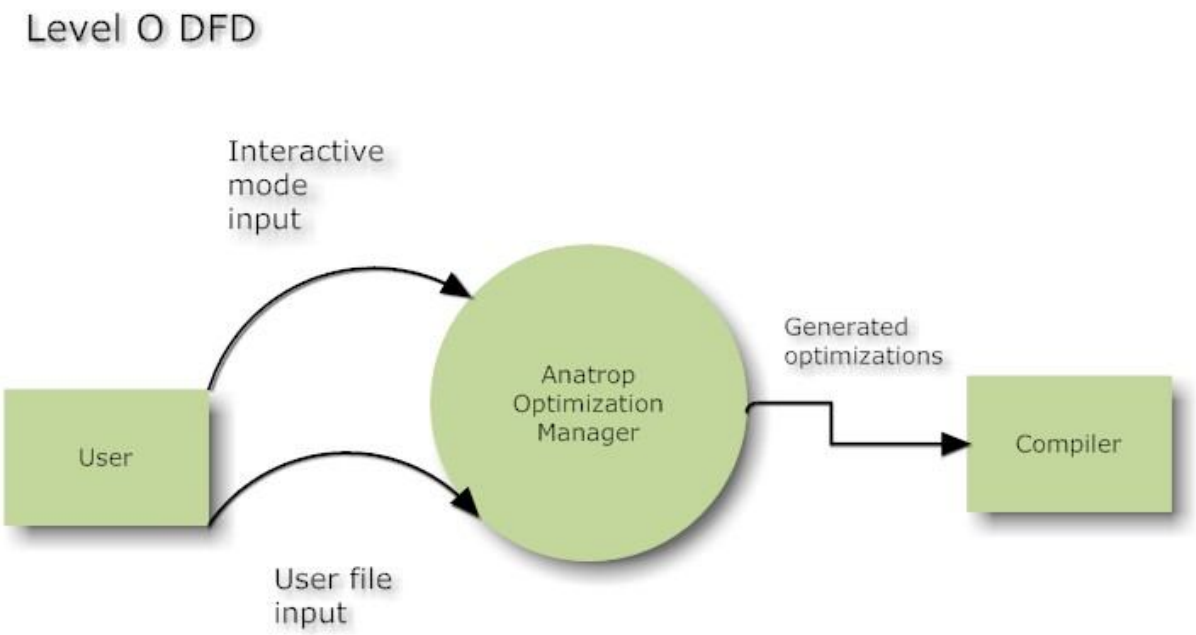
Level O DFD
Interactive mode input
User
Anatrop Optimization Manager
Generated optimizations
Compiler
User file input

Diagram 4.f

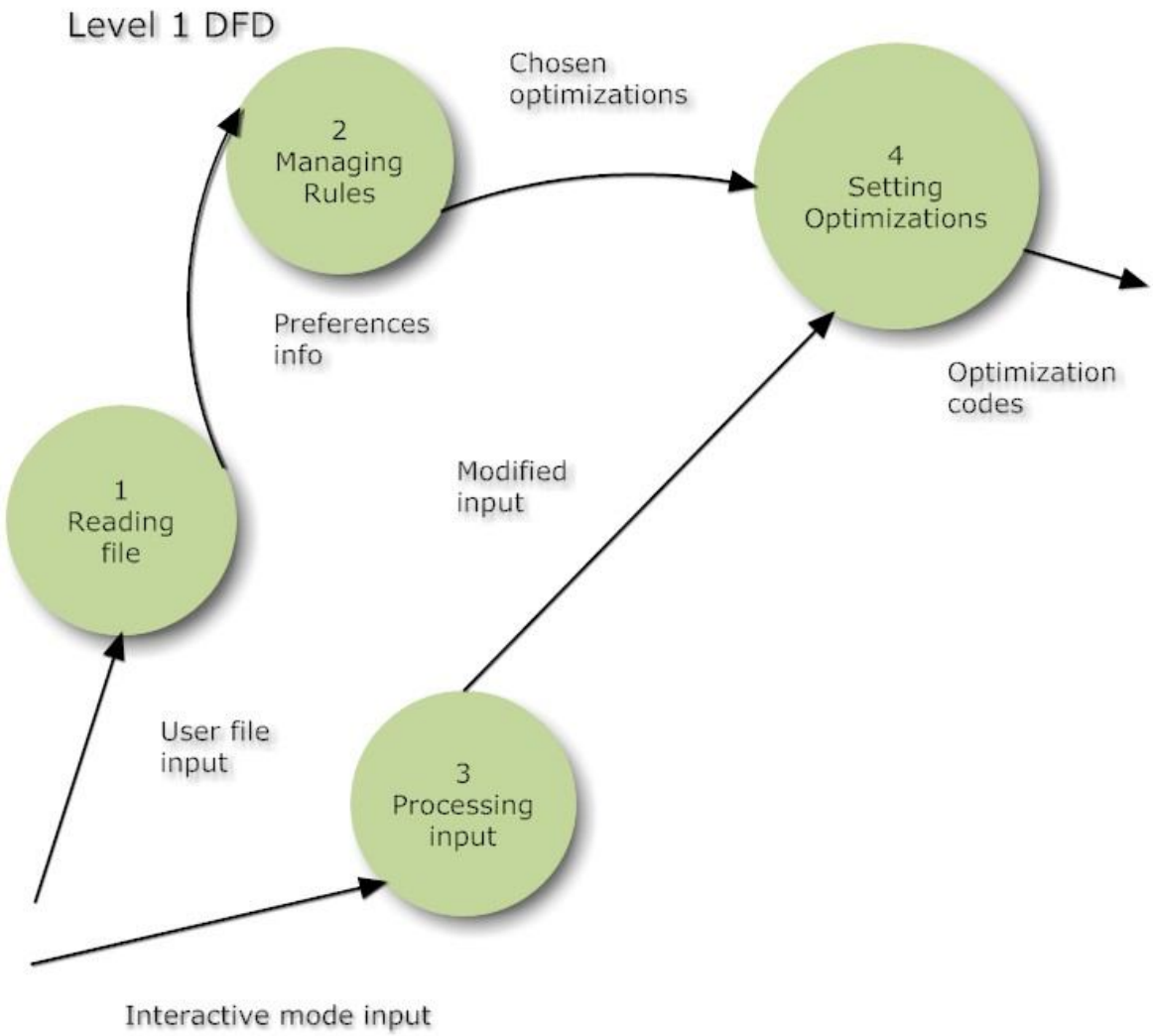

Diagram 4.g

Level 2 DFD

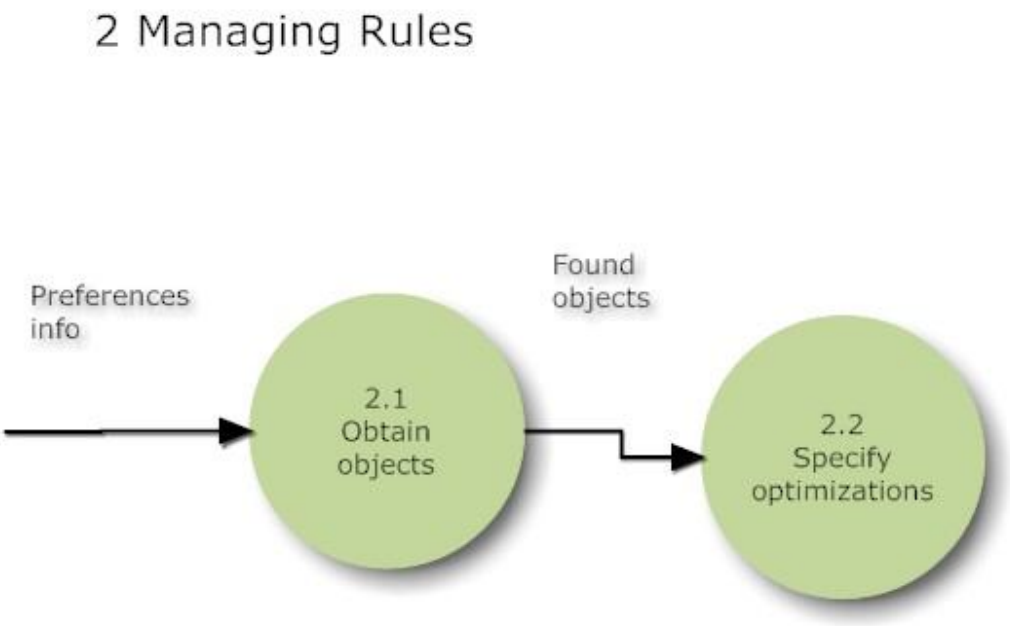

Diagram 4.h

3 Processing Input

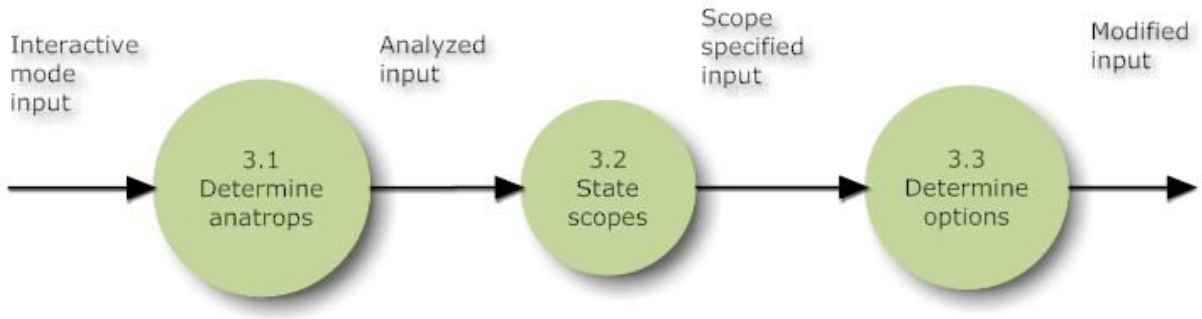

Diagram 4.i

4. Setting Optimizations

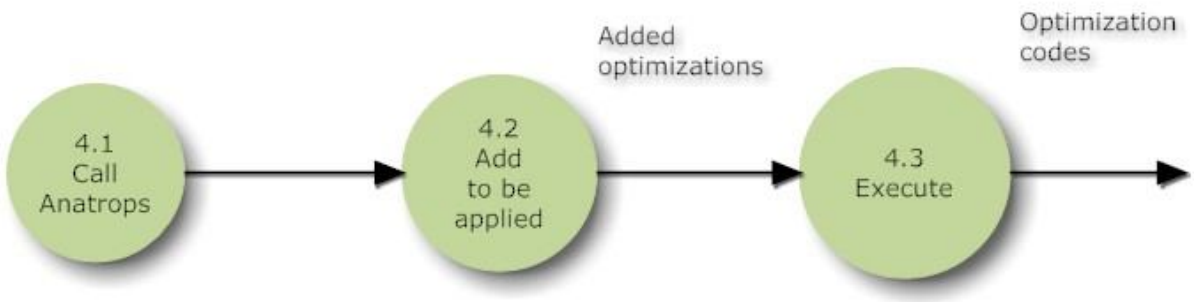

Diagram 4.j

4.SYNTAX SPECIFICATION:

4.1 PROJECT LANGUAGE:

Our Project is based on a framework which is coded with C++ programming language. That is why we have to use C++ as our project programming language. We will use framework while coding the optimizations but the other parts of the project will be handled by us. As a result of this , we will use the same language, as in the optimizations, with the other parts of the project. C++ will be our project language.

4.2 ANATROP CLASSES:

Every software project has its unique syntax definitions and guidelines to follow up but mostly software experts do not pay attention to this. Hellim project group will really try to obey the guideline and the syntax definitions while writing codes. It is obvious that a person, it does not matter he is a computer engineer or software expert, can have problems while checking out the codes if do not have a guideline to follow up; maybe only the author of the codes can understand it. We do not want such kind of things. As a result of our weekly meetings we decided to have standard methods for naming anatrops. It must be begin with a prefix like "*ato_*" and then it will continue with its original name which explains its functionality. Name must be unique. As an example we can show this;
For the dead code elimination optimization*: ato_deadcode.cpp* and *ato_deadcode.hpp*

4.3 COMMENTS:

Increasing the understandability of the codes we write comments after some lines. We will use C++ comments to reduce the complexity.
It will be like this;
// this function does that
// this call does that

5.Gantt Chart:

ID	Task Name	Resource Names	% Complete	Start	Finish	Duration
1	Requirements Specification		100%	9/3/2007	10/16/2007	32d
2	Project Proposal		100%	9/3/2007	10/5/2007	25d
3	Learning the languages and tools		100%	9/3/2007	11/15/2007	54d
4	Requirements Analysis Report		100%	10/12/2007	11/2/2007	16d
5	Initial Design Report		100%	11/12/2007	12/3/2007	16d
6	Final Design Report		0%	12/24/2007	1/11/2008	15d
7	Constant folding	Halit	60%	11/26/2007	12/14/2007	15d
8	dead code elimination	Halit	0%	12/17/2007	1/4/2008	15d
9	local/global copy propagation	Halit	0%	1/7/2008	1/25/2008	15d
10	local/global forward substitution	Kutay	0%	12/10/2007	1/18/2008	30d
11	Basic Block Ordering	Kutay	0%	1/18/2008	2/7/2008	15d
12	strength reduction	Tayfun	0%	12/11/2007	12/31/2007	15d
13	dead object elimination	Tayfun	0%	1/2/2008	1/22/2008	15d
14	jump optimizations	Tayfun	0%	1/23/2008	2/12/2008	15d
15	if simplifications	Tayfun	0%	2/12/2008	3/3/2008	15d
16	tail merging	Tayfun	0%	3/3/2008	3/21/2008	15d
17	local/global common subexpression elimination	Anil	0%	12/10/2007	1/18/2008	30d
18	unreachable code elimination	Anil	0%	1/18/2008	2/7/2008	15d
19						
20	Partial redundancy elimination	Anil	0%	2/11/2008	3/14/2008	25d
21	Procedure cloning and specialization	Kutay	0%	2/11/2008	3/14/2008	25d
22	Tail recursion	Halit	0%	2/1/2008	3/6/2008	25d
23						
24	Total Excepted Time for Optimizations		10%	11/30/2007	3/6/2008	70d
25	Test Case Generator	Anil+Halit	8%	12/3/2007	2/22/2008	60d
26	Optimization Manager	Kutay+Tayfun	0%	1/7/2008	3/7/2008	45d

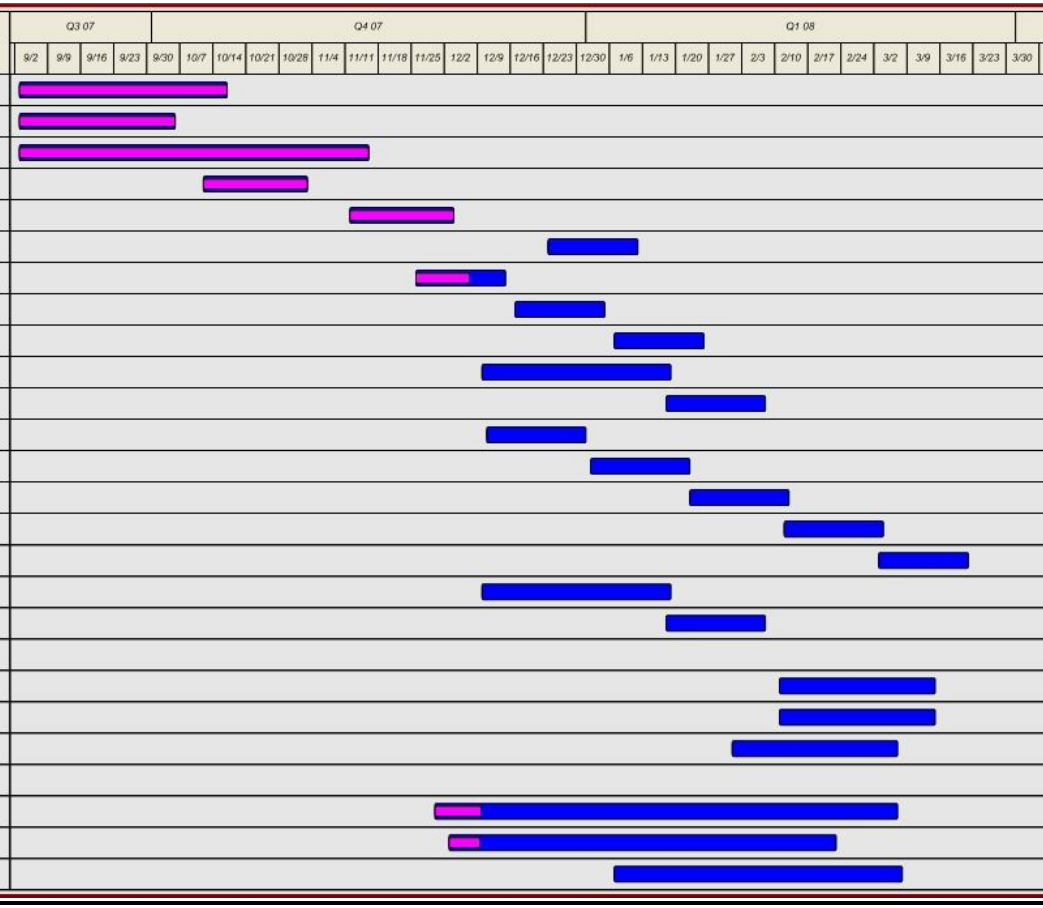

Table 1.a

6.Conclusion:

6.1.What has been done so far?

Since we started to this project , we have improved our skills and knowledge on the project. Firstly , we have studied and stated the aim of the project. Then we determined main topics of the project as we showed in our proposal. After determining the main topics , we have started analyzing the framework. We have examined working principle of the framework , classes and functions. Then we have searched information about optimizations and appropriate algorithms for them. Then we stated the requirements , what is needed ,scheduling and work weight. After examining the requirements , we have started to specify how to design our project . We have worked which classes and functions must be used and we stated how data and functions must work.

6.2.Future Work:

We have reached some important goals. Now we will start to develop our project. As it can be seen in our gantt chart(table 1.a) , we have divided our work. We will finalize our design aims. Then we will go into detail in programming level.