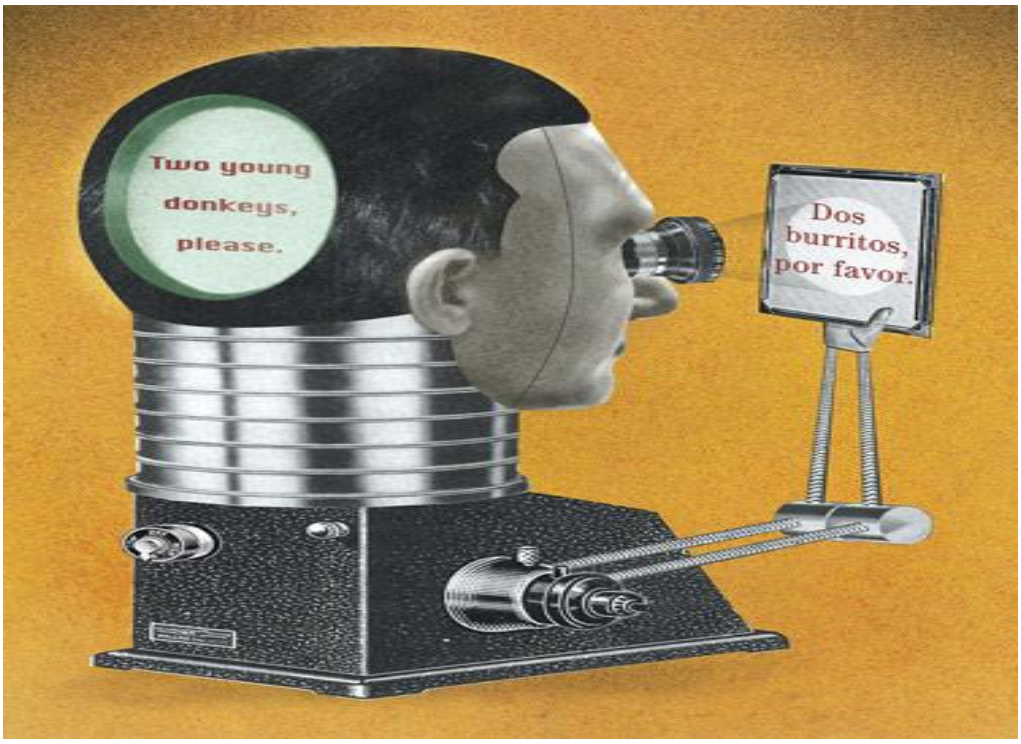
Institiúid Teicneolaíochta Cheatharlach
INSTITUTE of TECHNOLOGY CARLOW
At the Heart of South Leinster

Occam & C++ Translator



Student Name: Shaoguang Miao

Student ID: C00131017

Supervisor: Joseph Kehoe

Table of Contents

I. Introduction

Occam to C++ Translator is software which translates an Occam source file into C++ source code file. There are two separate processes for doing it. First of all, Lexer part will separate the Occam source file into tokens, and then make the tokens from each line into a linked list. Pass the linked list to Parser after Lexer finish recognise each line. Parser part will translate the linked list to C++ source code according to the C++ grammars. I wrote a library for the C++. This library is just for assignment part, because the assignment is really different between Occam and C++. The translated file can not run without including this library.

The whole project was written in C++ in Visual Studio 2008 environment. In this project, I used stack, linked list and file data structures to implement each functions. For the type of the tokens, I used *static const int* type as global variables. They will be introduced in the first part of this document.

There are 7 modules for this project: Graphic User Interface, Open Occam File, Save Occam File, Export C++ File, Translation, File Operations and File. What is the duty and how it works will be talked in the module description part.

Testing is always a big part of the project. For this project, test is more important. Because the Occam source file is written by people, there are numbers of possibility for representing one meaning. Even it is a context free language; it still need more test to prove is good enough to handle everything. There are some simple examples which used to test this program showed in the testing part.

The rest part is talking about what kind of experiences I get from this project and what I have done in the project.

II. Data Structures

The most widely used data type in this project is ***struct.*** It is the core part of linked list and stack. In the whole project, whatever in the Lexer part or Parser part, linked lists ware always used for handling tokens. Because the Occam uses indentation to mark the scope of each keywords but C++ uses curly bracket, make sure the scope of each key word is the first thing. In this part, I used mounts of stack to record the start position of each key word, like *if_stack, while_stack, seq_stack, par_stack*, and *replicator* stack etc. I will show the detail of stack node and token node following. Actually, it is exactly as same as the data type introduced in design document.

First of all, the token structure showed below:

```
struct token{
        int type;//Used to recored the type of the word.
        char *name;//Used to record the value of the  word.
        token *next;//Point to next node.
        token *pre;//Point to prvious node.
};
```

After that, the stack node showed below:

```
struct stack{
        /*Using position to identify the position of indentation and number to indentify the number of
        variables in this scope. */
        int position_and_number;
        stack* next;
};
```

Sometimes I also have to record the string which should be translated and I do want the type is the element of it. Thus, I create a new struct for doing that kind of thing to save memory space. It also showed below:

```
struct parser_linked_list{
        char *name; //Record the name of the node.
        parser_linked_list *next;
};
```

How to define the data type of the tokens is easy, but which one is best is the difficult problem. In this project I used *static const int* type to define global variables to be the data type of the tokens. It is also the same with the description in the design document. The detail showed below:

```
static const int UNKNOWN=0;
static const int ASSIGN=1;
static const int QUESTION=2;
static const int EXCLAMATION=3;
static const int SKIP=4;
static const int STOP=5;
static const int SEQ=6;
static const int IF=7;
static const int WHILE=8;
static const int PAR=9;
static const int ALT=10;
static const int LR_BRACKET=11;
static const int RR_BRACKET=12;
static const int LC_BRACKET=13;
static const int RC_BRACKET=14;
static const int FOR= 15;
static const int FROM = 16;
static const int NUMBER = 17;
static const int OPERATOR = 18;
static const int CHAN = 19;
static const int BOOL = 20;
static const int BYTE = 21;
static const int INT = 22;
static const int INT16 = 23;
static const int INT32 =  24;
static const int REAL32 = 25;
static const int REAL64 = 26;
static const int QUOTATION = 27;
static const int INDENTATION = 28;
static const int LA_BRACKET = 29;
static const int RA_BRACKET = 30;
static const int EQUAL = 31;
static const int END_NODE = 32;
static const int DECLARATION = 33;
static const int PUTINT = 34;
```

It also showed the key data type and key word of the Occam which I have handled most of them in this project.

Further more, I used three template file for storing the information which the software get from the Occam source file. After the Lexer made the Occam source code into a linked list consisted by tokens, Lexer also should write this linked list into a file. This file is useless for the final result, just for debugging the software and let me know the token was defined correctly or not. The other file is use to store the normal expression beside a SEQ followed by PAR. It is a really special expression, because the Parser will build a function for it. This expression will be translated into final C++ file immediately. After finish the translating, the Parser will copy the entire thing from the normal file into final C++ file to build an exactly matched C++ program.

Because the assignment part in the Occam is really different from the C++, I developed a library by my own. In the C++ main function will call the function to do the assignment part. The basic constructor shown below:

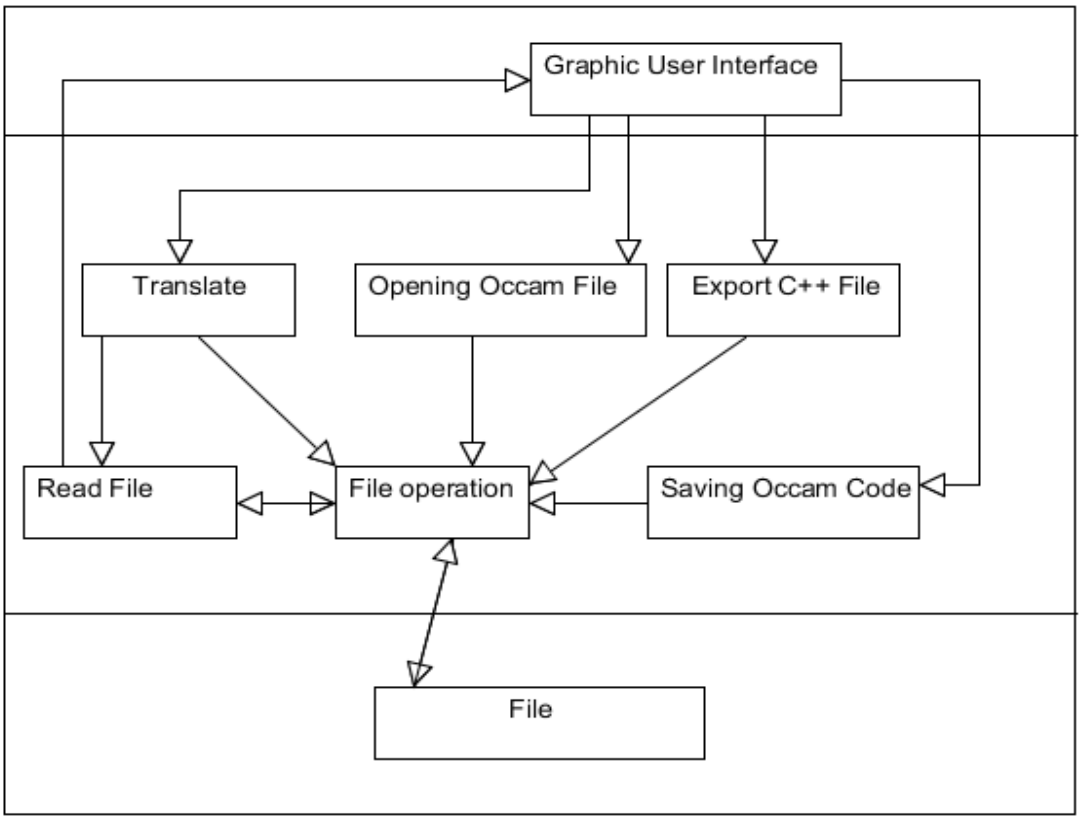
```
static void assignment_INT(int *para1 ,int *para2 ,int *para3 ,int *para4 ,int *para5 ,
                           int value1,int value2,int value3,int value4,int value5){
*para1 = value1;
*para2 = value2;
*para3 = value3;
*para4 = value4;
*para5 = value5;
}
```

Because the boost library will support upto 10 parameters for each thread, the max number of the parameters will be 10. Get the address of the varibale which is left of the assignment and value of variable which is right of the assignment. Then do the thing dscripted above.

III. Module Descriptions

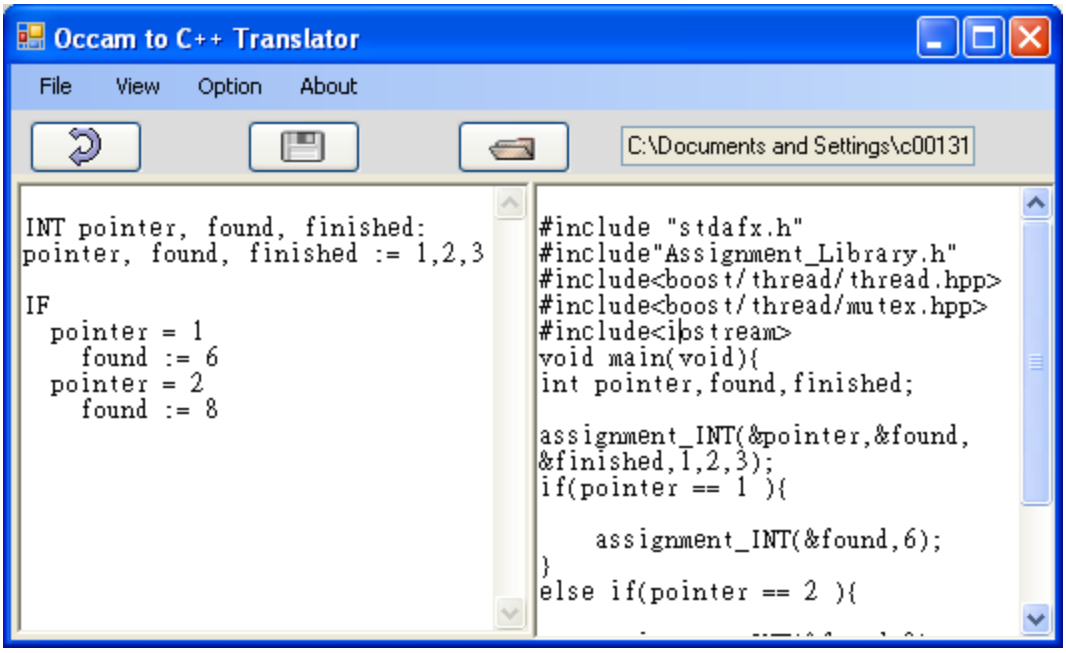
1. Module Diagram

As what I showed in the design document, the Module Diagram shown below:



2. Graphic User Interface:

User will control the software through out of Graphic User Interface. It is also the only thing the user can know about the software, thus I try my best to make it looking better. The main window showed below (Fig.1). Actually, it is followed by standard rules to make it easy to use for user.



3. Open Occam File:

After the user clicked the open file button, this module will be used. An open file dialog will be showed on the top of main window; user could select the file which he wants to open. After the use submits the file, the content of the file selected will be shown in the left text box area. This must be the first step of the whole project. Translate and save button can not be used without opening any file before.

4. Save Occam File:

After the user modified the content of the Occam source file, the user could use this module to save a file into directory. The user could save the current file or save it as the other new file which will be created by the software.

5. Export C++ File:

This is exactly like save as option, the different is it will export a C++ file. This module can not be used unless the user has translated Occam file to C++ file.

6. Translation

This is the core module of this project. There are two parts in this module. First of all, the Lexer will recognise every string from the Occam source and put it in a linked list and pass the linked list to Parser which is the other part of this module. Parser will translated the linked list and write the result into a template file. After all the translating, the Lexer will make the entire of all template files together as a new C++ files.

7. File Operations

This file used the I/O stream to read from and write to the files. Actually, I did not create an individual class for this module, but it is a special module of the project. Whatever it is GUI, Lexer or Parser; this part will be used in it. All of the operations to the files are included in this module.

8. File

This module is the memory module. This project used three template file for store the useful information. All of the template files should belong to this module.

IV. Testing Results

Testing is always a big part for any project, so is mine. This project is separated by lots of small pieces of functions, and I made a testing for each piece of functions. I will give the test detail following:

1. Simple assignment Test:

```
INT32 apple, apricot:

apple := 2

apricot := 3

apple := apple + 1
```

C++ Result:

```
#include "stdafx.h"
#include"Assignment_Library.h"
#include<boost/thread/thread.hpp>
#include<boost/thread/mutex.hpp>
#include<iostream>
void main(void){
int apple,apricot;
assignment_INT(&apple,2);
assignment_INT(&apricot,3);
int newapple_1= apple+1;
assignment_INT(&apple,newapple_1);
}
```

2. Swap Test:

```
INT32 apple, apricot:
apple := 2
apricot := 3
apple,apricot := apricot, apple
```

C++ Result:

```
#include "stdafx.h"
#include"Assignment_Library.h"
#include<boost/thread/thread.hpp>
```

```
#include<boost/thread/mutex.hpp>

#include<iostream>

void main(void){

int apple,apricot;

assignment_INT(&apple,2);


assignment_INT(&apricot,3);

assignment_INT(&apple,&apricot,apricot,apple);

}
```

3. IF statement Test:

```
INT pointer, found, finished:

pointer, found, finished := 1,2,3


IF

  pointer = 1

    found := 6

  pointer = 2

found := 7
```

C++ Result:

```
#include "stdafx.h"

#include"Assignment_Library.h"
```

```
#include<boost/thread/thread.hpp>

#include<boost/thread/mutex.hpp>

#include<iostream>

void main(void){

int pointer,found,finished;


assignment_INT(&pointer,&found,&finished,1,2,3);

if(pointer == 1 ){

   assignment_INT(&found,6);

}

else if(pointer == 2 ){

   assignment_INT(&found,7);

}

}
```

4. Nested IF statement Test:

```
INT pointer, found, finished:

pointer, found, finished := 1,2,3

IF

  pointer = 1

    IF

      finished = 3

        found := 6

  pointer = 2

found := 7
```

C++ Result:

```
#include "stdafx.h"

#include"Assignment_Library.h"

#include<boost/thread/thread.hpp>

#include<boost/thread/mutex.hpp>

#include<iostream>

void main(void){

int pointer,found,finished;


assignment_INT(&pointer,&found,&finished,1,2,3);

if(pointer == 1 ){

   if(finished == 3 ){

      assignment_INT(&found,6);

   }

}

else if(pointer == 2 ){

   assignment_INT(&found,7);

}


}
```

5. Replicator Test:

```
INT pointer, found, finished:

pointer, found, finished := 1,2,3


SEQ i = 0 FOR 10

  pointer := i + pointer

  finished := i * finished
```

C++ Result:

```
#include "stdafx.h"

#include"Assignment_Library.h"

#include<boost/thread/thread.hpp>

#include<boost/thread/mutex.hpp>

#include<iostream>

void main(void){


int pointer,found,finished;

assignment_INT(&pointer,&found,&finished,1,2,3);

for( int i=0; i<10; i++){


  int newi_pointer= i+pointer;

  assignment_INT(&pointer,newi_pointer);

  int newi_finished= i*finished;
```

```
  assignment_INT(&finished,newi_finished);
}
}
```

6. WHILE Test:

```
INT pointer, found, finished:
pointer, found, finished := 1,2,3

WHILE pointer <> 10
 pointer := pointer+1
 found := found + 1
```

C++ Result:

```
#include "stdafx.h"
#include"Assignment_Library.h"
#include<boost/thread/thread.hpp>
#include<boost/thread/mutex.hpp>
#include<iostream>
void main(void){
int pointer,found,finished;
assignment_INT(&pointer,&found,&finished,1,2,3);
while (pointer != 10 ){
 int newpointer_1= pointer+1;
assignment_INT(&pointer,newpointer_1);
```

```
 int newfound_1= found+1;

assignment_INT(&found,newfound_1);

}

}
```

7. WHILE_IF Test

```
INT32 apple, apricot:

apple, apricot := 6, 7

WHILE apricot <> 10

  IF

    apricot <> 10

      IF

        apple = 6

          apple := apple+7

 apricot := apricot + 1
```

C++ Result:

```
#include "stdafx.h"

#include"Assignment_Library.h"

#include<boost/thread/thread.hpp>

#include<boost/thread/mutex.hpp>

#include<iostream>

void main(void){

int apple,apricot;
```

```
assignment_INT(&apple,&apricot,6,7);
while (apricot != 10 ){
  if(apricot != 10 ){
      if(apple == 6 ){
          int newapple_7= apple+7;
          assignment_INT(&apple,newapple_7);
      }
  }
 int newapricot_1= apricot+1;
assignment_INT(&apricot,newapricot_1);
}
}
```

8. Nested Loop Test:

```
INT32 apple, apricot:
apple, apricot := 6, 2
WHILE apricot < 3
  SEQ i = 0 FOR 10
     apple := apple +1
  apricot := apricot + 1
```

C++ Result:

```
#include "stdafx.h"
#include"Assignment_Library.h"
#include<boost/thread/thread.hpp>
```

```
#include<boost/thread/mutex.hpp>

#include<iostream>

void main(void){

int apple,apricot;

assignment_INT(&apple,&apricot,6,2);

while (apricot < 3 ){

  for( int i=0; i<10; i++){

      int newapple_1= apple+1;

      assignment_INT(&apple,newapple_1);

  }

  int newapricot_1= apricot+1;

  assignment_INT(&apricot,newapricot_1);

} }
```

9. PAR Test:

```
INT32 apple, apricot:

apple := 1

apricot := 2

PAR i = 0 FOR 10

    apple := apple + 1

apricot := apricot + 1
```

C++ Result:

```
#include "stdafx.h"

#include"Assignment_Library.h"
```

```
#include<boost/thread/thread.hpp>

#include<boost/thread/mutex.hpp>

#include<iostream>

void main(void){

int apple,apricot;

assignment_INT(&apple,1);

assignment_INT(&apricot,2);

for( int i=0; i<10; i++){

   int newapple_1= apple+1;

   boost::thread assign_thrd_11(boost::bind(&assignment_INT,&apple,newapple_1));

   assign_thrd_11.join();


    int newapricot_1= apricot+1;

   boost::thread assign_thrd_12(boost::bind(&assignment_INT,&apricot,newapricot_1));

   assign_thrd_12.join();

}

}
```

V. The differences from earlier design document

There are just little bit different between my final version and the design document, because I have done the design of whole structure of project. The different is how to implement the algorithm.

For GUI part, at the beginning, I want to add search function which supporting the user look for some special key words in the document. However, the time is limited, I did do that. The same reason, I did not do the options part either.

VI. What I have learned

1. Technical Aspect

When I was doing the research of the project, I get deep understanding about what is parallel programming, how the parallel programming works and the advantages and disadvantages of some popular parallel libraries for C++. In this project I use BOOST to implement the parallel part. I get a little idea about how the complier works at that time.

When I was doing the project, I get a deep understand about what a good complier should include, how a good complier works and how to design a good compiler. During this time, I knew how important the software engineering is. The clear the module is made, the easy the project is modified. UML is a good tool to design document. It makes the project structure clearly to understand and make myself understand the project deeply.

Last year, after I finish the project, I understand the important of the project design. I had same feeling about the project design this year. When I started doing the project, I just finish a part of project. I finish that part very successful, but not the other part. I rewrite the assignment part for three times for looking for a better way. If I design it first, I believe it will be quicker to finish it.

2. Personal Aspect

Test in this project is a quiet important part. After finish a little part of it, I have to make sure it is right. I think I spend more ½ times to debug and test in the whole project. I became a more patient people during this project. It let me know that a good project is not only made by intelligence but also patient. Fortunately, I am interested in the technical area of this project. It makes me happy.

VII. What I achieved & not achieved

The *token.h* has defined all the keyword I should translated in this project, what I have done and not have done showed below:

```
static const int UNKNOWN=0; //Done
static const int ASSIGN=1; //Done
static const int QUESTION=2;  //Done
static const int EXCLAMATION=3; //Not Yet
static const int SKIP=4; //Done
static const int STOP=5; //Done
static const int SEQ=6; //Done
static const int IF=7; //Done
static const int WHILE=8; //Done
static const int PAR=9; //Done
static const int ALT=10; //Not Yet
static const int LR_BRACKET=11;  //Done
static const int RR_BRACKET=12; //Done
static const int LC_BRACKET=13; //Done
static const int RC_BRACKET=14; //Done
static const int FOR        = 15; //Done
static const int FROM = 16; //Done
static const int NUMBER = 17; //Done
static const int OPERATOR = 18; //Done
static const int CHAN = 19; //Done
static const int BOOL = 20; //Done
static const int BYTE = 21; //Done
static const int INT = 22; //Done
static const int INT16 = 23; //Done
static const int INT32 =  24; //Done
static const int REAL32 = 25; //Done
static const int REAL64 = 26; //Done
static const int QUOTATION = 27;  //Not Yet
static const int INDENTATION = 28; //Done
static const int LA_BRACKET = 29; //Done
static const int RA_BRACKET = 30; //Done
static const int EQUAL = 31; //Done
static const int END_NODE =  32//Done;
static const int DECLARATION =;  33; //Done
static const int PUTINT = 34; //Done
```

VIII. Problems & Solutions

Generate Speaking, there is no too much problems in the whole project. The most difficult part should be PAR & SEQ part. Because if a SEQ key word is in the PAR scope, the parser should create a function for the expressions which is in the SEQ scope. How to pass the value and address into this function is a big problem. Because all of the code written by peoples not the machine, there are too many possibilities in this situation. If there are values and address should be pass into the function at same time, how the parser handle this problem?

For example:

```
INT pointer, found, finished:
INT32 apple, apricot:
pointer, found, finished := 1,2,3
apple, apricot := 6, 7
WHILE apricot <> 10
 PAR i = 0 FOR 10
  IF
    pointer = 10
     SEQ
      IF
        apple = 6
        found := found +i //Here is the point of the code
  apple := pointer+1
 apricot := apricot + 1
 pointer := 1
```

Because I have to pass the address of found assign with found + i, I have to pass the address and value at same time. However it is not allowed to create a function declaration like this:

```
void squence_f_4(int *i,int *found,int apple,int found);
```

What I have done for this problem is record the entire variable appeared in the SEQ scope in a linked list. When I write expression in the function, I will check if it is in the linked list. If it is in the linked list, the parser will handle it using special way. If it is not, add it into linked list. For the function call, parser will just pass the address of the variable, because it's easy to get the value if the function has its address.

IX. What I would do if starting again

I will notice the following points if let me do it again:

Get more suggestions from the supervisor:

In this project, I get lot of suggestion from my supervisor Mr. Joseph Kehoe. He gave me a lot of good suggestions to help me to do my project. However, I did not do what he suggested that using Antlr or some the other complier tools to do this project. I used C++ to write this project, it almost drives me crazy. I have to write every little detail by my self, so that I spend too much time on the project. If let do it again, I will take his suggestions and arrange my time, maybe using complier tools as a module of the project is good idea.

Algorithm Design:

As I mentioned before, I believe it is much better than coding part until now. If the programmer would like spend one hour in the project design part, he will spend less than 10 hours in the coding part. The design part is not only the part we are doing before us writing the design document, but also for the design of each function. If I would like write the algorithm for some import function, I will save lot of time. Thus I will pay more attention on the algorithm design part.

X. Conclusion

I learned a lot from this project whatever in technical skill or software engineering. I get deep understand about what is software development, it is not only the coding, and the more import thing is design and test. If I would like spend the shortest time to finish developing software, I will try my best to design a good structure first.

I was very interested in the structure of the complier, which is also the reason why I want to do this project. I understood how to make a good compiler and what are the features of a good compiler through this project.

Parallel computing will be more and more popular in the feature. I also get the basic idea how to make a parallel program during this long time. From the researching, I understand what the normal parallel computing library does and how to use them to make parallel programs. If I got time, I may look at the code of the library to understand how it works.

Finally, I have to thanks my supervisor Mr. Joseph Kehoe, he give me a lot of help during this academic year.