User Interfaces Development in openDIEL

Argens Ng

August 5, 2016

Abstract

This paper serves the purpose of reporting the progress of user interfaces enhancement in the work flow engine openDIEL. While openDIEL has the potential to become a powerful work flow engine, its interface towards users has been staggering in development which magnifies the urge of this project. In this paper, I will describe the progress of a module making python script known as modMaker, as well as the development tool timer.

1 Introduction

OpenDIEL stands for open Distributive Interoperable Executive Library. It is a lightweight software framework which aims at combining different interoperable computational componenets to simulate system-wide scientific application. It uses Message Passing Interface (MPI) to facilitate the cooperation between loosely coupled modules and outputs a single executable.

To use openDIEL, user needs to provide:

1. Modules
2. Configuration File (using libconfig)
3. Driver (driver.c)

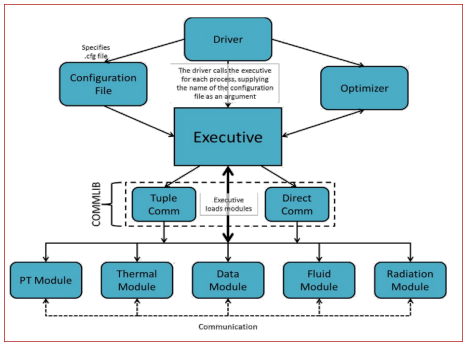

Figure 1: The structure of openDIEL.

Our ultimate goal is of course to automate the generation of all of the above three files, or sets of files. In this project, we will focus on the generation of the module first. This will be done by a python script called modMaker.

2 ModMaker

2.1 What is modMaker?

At the current stage, modMaker is a package of 2 python scripts – modMaker.py and worker.py. Together they can transform a C-file or a directory of C-files (and other files from C family) into a module (or modules). This is done by a series of pattern matching of strings as well as the addition of static supporting files.

2.2 What is a module?

For user defined code or simulation models to run in openDIEL, it has to be in the format of an openDIEL module. There are a few requirements. For example, it has to be rid of the main program. It also has to be rid of MPI_COMM_WORLD, MPI_Init and MPI_Finalize to facilitate the cooperation between different modules. This complicated formatting would be done by modMaker in the following manner.

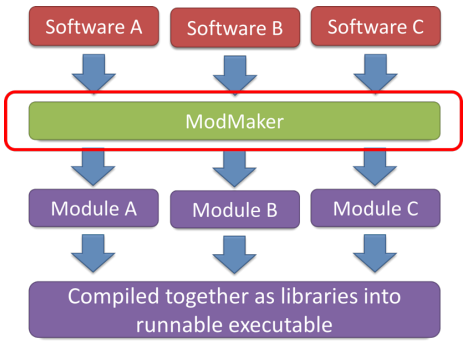

Figure 2: Illustration of modules.

2.3 How to transform a module?

To transform a module, we have to first understand the syntactic structure of the language. As the C-family is our first target, we would naturally focus on it first.

I started off by focusing on the target "int main".. As "int main" is now located in driver.c, we would need to replace the main program of individual modules with a function header. To do that, we have to accurately identify "int main" and modify it. (To be more precise, we would also need to consider "void main" but we would focus on "int main" only for the purpose of this paper)

Thinking as a human, I quickly realize that as long as "int" and "main" are two separate and individual strings, it would be unique and correct as the target we would like to change. This actually holds true for all other targets that we wish to change and hence the problem becomes "identifying individual strings correctly separated". From this insight, the following flow diagram is constructed.

Figure 3: ModMaking workflow.

2.3.1 Locating identifying feature

Two approaches were considered for locating identifying feature. The first one is "matching by character" and the other is "matching by string". To illustrate the difference, take a look at Fig.4 below.

Text	/	*	T	h	i	s		i	s
Pattern 1	i	n	t		m	a	i	n	
Pattern 2	M	P	I	_	I	n	i	t	
No match x 2	/	*	T	h	i	s		i	
No match x 2		*	T	h	i	s		i	s

Text	/	*	T	h	i	s		i	s
Pattern 1	i	n	t		m	a	i	n	
No match	/	*	T	h	i	s		i	
No match		*	T	h	i	s		i	s
Pattern 2	M	P	I	_	I	n	i	t	
No match	/	*	T	h	i	s		i	
No match		*	T	h	i	s		i	s

Figure 4: Matching-By-Character vs Matching-By-String

The upper diagram illustrates the process of matching-by-character. By reading in and comparing at each character, we can determine if it is a match to any of the patterns that we are trying to match. So since the first character is a '/', which is neither a match to pattern 1 nor pattern 2, we will continues to match the next character with the heads of both string (we will not proceed within the patterns).

The bottom diagram illustrates the process of matching-by-string. By reading in a line at a time, we can use python function string.find(string) to see if the line contains the target patterns, looping by pattern. Again, we found out that there is no match for both patterns at both position 1 and 2 in 4 comparisons.

While the benefit is not significant in this case. Matching-by-character can lead to a performance boost if the patterns have similar sub-string head to start with. For example, when matching patter "MPI_Init" and "MPI_Finalize", we can determine that both patterns have first 4 characters matched or not with 4 comparisons instead of 8, after understanding that they starts with the same 4

characters.

However, it was soon realized that this require extra effort in identifying common sub-string start. As long as the built-in string.find(string) can terminate comparisons prematurely upon finding unmatched characters, the performance boost would be insignificant, especially so when compared with the time waiting user input. Hence matching-by-string was used.

2.3.2 Finding Replacement Candidate

First we need to understand the difference between "locating identifying feature" and "finding replacement candidate". Identifying feature refers to pattern such as "int[space]main" and "int[tab]main" (Notice that they are no different from the compiler's perspective). However, we would like to replace much more than identifying feature itself. For example, we might want to replace "int main(int argc, char** argv)" or simply "int main ()". This is when syntactic freedom of C proves to become a barrier.

Luckily, C is a language depending heavily on separators, in contrast to Python, which is a indentation based language, and Fortran, which has its own set of strict formatting rules, C can have the whole code in one line and minimal separation. Its extensive use of special characters is both a threat and opportunity for our module transformation. In this case, it is the solution to the above problem.

```
1) int main(int argc,char**argv){\n
2) MPI_Init(&argc, &argv);
3) rc=MPI_Init(&argc,&argv);
4) int rc=MPI_Init(&argc,&argv);
5) int main(intargc,char**argv) {\n
```

Figure 5: Finding replacement candidate

It soon became apparent that the closing parenthesis ")" marks the end of our replacement candidate. To tackle this problem more systematically, we divide the statements that we need to convert in C into 3 categories.

1. Function Title
2. Statement
3. Variable

Function title refers to cases like "int main". They are likely ended with a closing parenthesis and then followed by a open bracket "". Statement such as "MPI_Init()" can be function call or assignment of variables. Luckily in C, they are usually ended with a semi-colon ";". Lastly, variables are usually not enclosed in separators and they are identifiable by themselves alone. Examples are "MPI_COMM_WORLD".

By locating the separator in front of an "Identifying Feature" and the one after, we can now locate with high accuracy the "Replacement Candidate" for our modules. After checking the spaces within and making sure each token is in fact a word by itself, we can pass on the results for the user to verify.

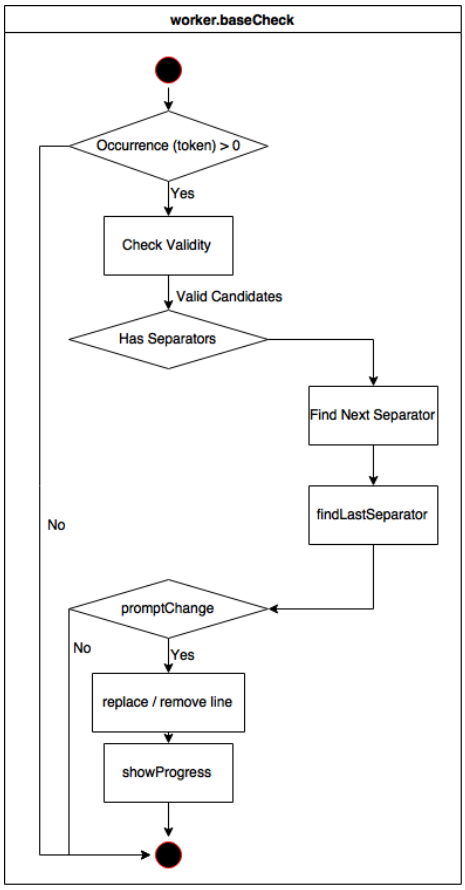

Figure 6: Modular approach to various checking method

2.3.3 User Participation

Notice that we have been using the word "Replacement Candidate". This is because we believe that there might be missing factors even after serious investigation, and we do not wish to alter the program without user consent. This can cause serious problem that is hard to debug.

Hence all the replacement that we would like to change would appear on screen in a highlighted manner, with replacement suggestion listed out to see if the user see fit.

This soon raises another problem, which is the huge number of prompts generated. While the modMaker script was created with test cases of 2 files and around 40 lines in total, it was soon discovered that hundreds of files and tens of thousands of codes are common in real life scenario. Hence 3 tactics were employed to combat this problem.

The first one is the combination of similar prompts. In our case, MPI_COMM_WORLD is the most common token and hence generates the most prompts. Hence we decided to count, combine and ask for confirmation all together as illustrated in Fig.8.

The second tactic is to use extension matching to prevent going into un-

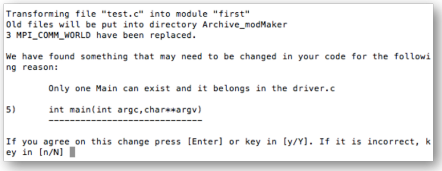

Figure 7: Prompting for user confirmation

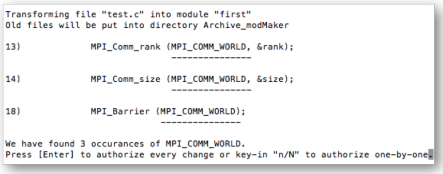

Figure 8: Combination of similar prompts

wanted files. This takes in multiple inputs and hence for example, ".c" and ".cpp" could both be accepted in one transformation.

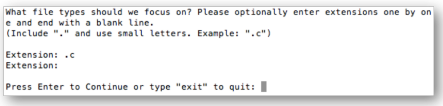

Figure 9: Getting the list of allowed file extension

This last tactic was to screen out files without any replacement candidate. While this seems obvious in later stages, it was not thought of in earlier stages and a lot of times was wasted for confirming file searches with user input.

As seen in the last figure, a large proportion of files did not require the modMaker to go through nor user to confirm. They are hence simply skipped. A large sum of time is saved after these 3 tactics were implemented.

2.3.4 Testing

Testing is a huge part in modMaker, simply because changing a program at source code level is dangerous and risky. Hence we provide sufficient testing tools for the user to make sure their module runs normally.

After changing a program into a module. It compiles into a library and acts like a function. Hence, a "tester.c" is provided for a main program entry point for the new module. The user can then compile it as a program again and see if it runs correctly.

An artificial "IEL.h" was also provided for the user to test the module out of openDIEL scenario. In the developement stage, we can separate the issues of module transformation from any problem in openDLEL. Even in later stages when openDIEL becomes stable, it is still always better to keep the testing environment as clean and simple as possible.

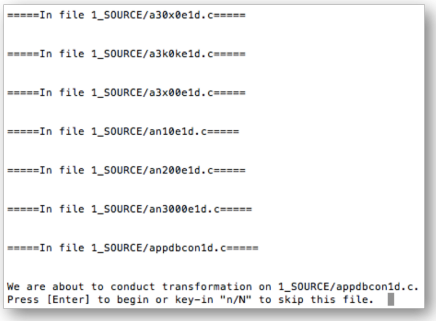

Figure 10: Skipping files without any chances of replacement

2.4 Timer

As a tool targeting at high-performance computing, openDIEL is supposed to save the user time. It is supposed to automate process combination with minimal overhead and thus by using timer, we can monitor this process and mark any progress.

We will now divide the use of timer from two perspective – that of openDIEL user and that of developer.

2.4.1 For OpenDIEL User

As said in the very beginning of this paper, users have to provide a configuration file to utilize openDIEL. This file consists the dependency of the modules as well as groupings. This file greatly determines the order of execution of modules as well as idle times.

While automated optimization would be ideal in the long run, at the current stage, changing the configuration file would be more than sufficient for optimization at the user end.

At the end of every run, there would be a conclusive timing information as follow.

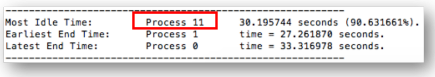

Figure 11: Showing conclusive timing information

Seeing that process 11 takes the most time, we can then go into a folder with more detailed information.

User can then tell that Function 5 waits for a lot of its dependencies and this results in a hefty 84% time wasted in process 11. The user can then think of how the configuration file can be rewritten to better optimize openDIEL and the CPU usage.

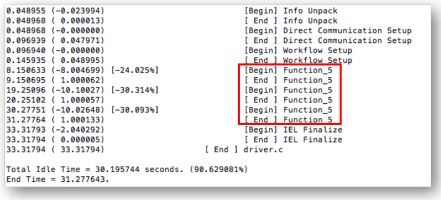

Figure 12: Showing detailed timing information for process 11

2.4.2 For OpenDIEL Developer

For openDIEL developer, timer can actually help reduce overhead in coordinating different modules. This is done by using levels defined in "timestamp" function.

Level basically refers to the hierarchy of function call. The top level (or level 1) is always driver.c. This has to be placed inside driver.c by the user after MPI_Init and before MPI_Finalize. Each call then add the level by 1 while return statements generally should be accompanied by a negative level of 1. This goes deeper and deeper and returns similar to call stack.

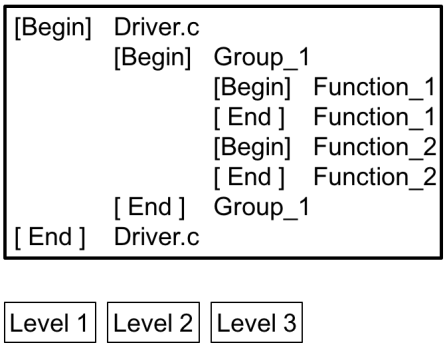

Figure 13: Showing the use of levels in timestamping

Levels are not only useful for presentation. It is also useful for extracting information for timing data without intervention of user.

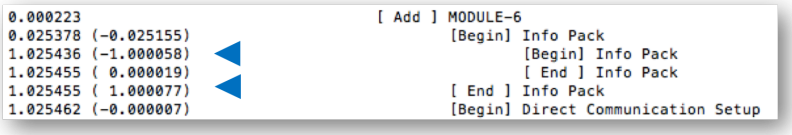

Figure 14: Showing the meaning of different times of timer files

Next to each time, a square bracket would contain a number representing time difference. If the level difference is negative, openDIEL has just returned from a function call or finished a process. The timer would then find the corresponding starting time and the total time used in this process can then be

found. This is shown as a positive number in the square bracket.

If level difference is positive of zero, however, this means that the time difference would be the time used between neighboring process of time used to prepare for this particular process. This time is most likely overhead that could be reduced by openDIEL developer. Hence the time difference is shown as negative.

2.5 Possible Development

While we have gone over several test cases and debugged several times, there is undeniably potential threats in our approach – that is the approach of string matching – to module making.

The first one is the dependency on human support. As a developer, it is quite frankly easy to develop white box test cases that can render modMaker useless. For example, we can create variables that contains the substring MPI_COMM_WORLD, or we can create wrap MPI_Init() in another function located in another file that is not gone under the radar of modMaker.

It would be even easier if we do not wish to defeat the core purpose of module making. For example, we can put the keyword "main" in every program as a comment, then the pre-detection stage of module making would become useless. We can also eliminate line break characters from the file. While this would not affect module making, users can no longer identify the key feature that require our attention because the limit of screen width would render highlighting useless.

The second one is the unpredictability of program structure. Simply put, we just cannot be sure we can handle all C-programs flawlessly using modMaker. For example, we only learned that "argc", "argv" replacement should be available in future versions of modMaker. However a quick implementation was proven ineffective. This is because we have been treating the "main file", or the file containing main program, indifferently from other files. A effective and efficient way to do that in the future would be to either identify the "main file" or asking the user to provide it, as the "main file" contains a lot of features that we would like to work on and is the place we need to add a lot of signatures.

To stress on the seriousness, imagine a program that used preprocessor directives. They are in effect another language building on top of C-files. These have even more unpredictable behavior and syntax. Imagine if the user has always been using it to replace keywords at compile time, thus using the directives together with compiler to correctly generate executable, the chance of a successful module making would then be even dimmer.

Problems like this would, in my prediction, come up for every large scale programs that we convert in the future. While we predicted at the beginning that human intervention was necessary, the amount needed at the current stage is far from satisfactory. The hope for rapidly losing this need is also dim. The creation of a fully automated modMaker would be as complex as creating a compiler or interpreter. This would not be an ideal solo or short project in the near future.

2.6 Future Work

Besides continual improvement of modMaker and timer, I believe that future development of the User Interface of openDIEL can aim at module making of

FORTRAN codes and the issue of multiple file I/O in running multiple copies of parallel modules.

2.7 Reference

1. openDIEL (http://cfdlab.utk.edu/openDIEL/opendiel.php)
2. Stack Overflow (http://stackoverflow.com)

2.8 Acknowledgment

This project is made possible only with the support of my mentor Dr. Kwai Wong, the NSF, the University of Tennessee and Oak Ridge National Laboratory. I would like to express my deepest gratitude towards all of the people, in particular Dr. Wong, who have supported and guided me throughout the project. I would not have achieved close that what I have today without them.

2.9 Appendix

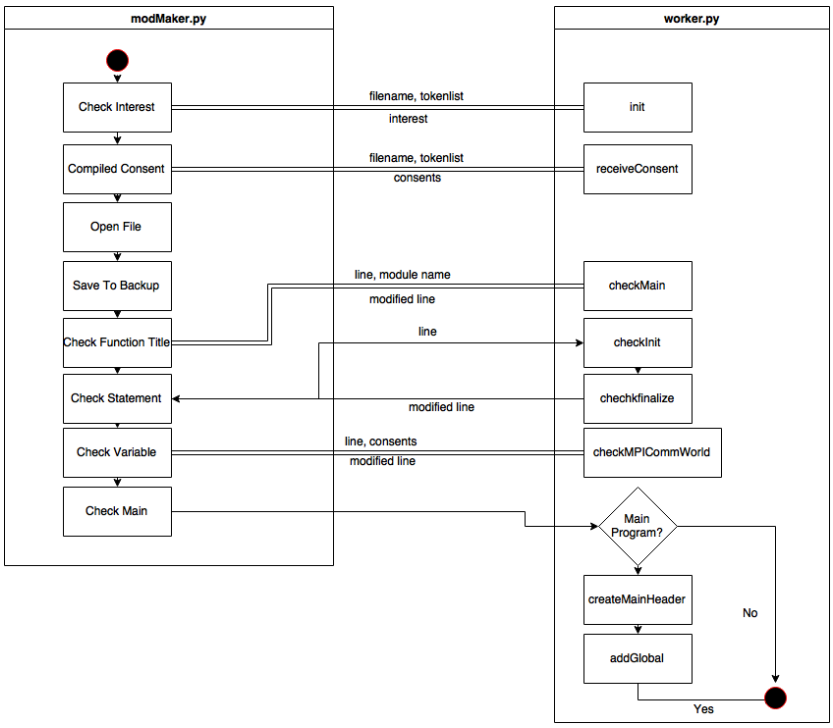

Figure 15: The flowchart of transforming a file

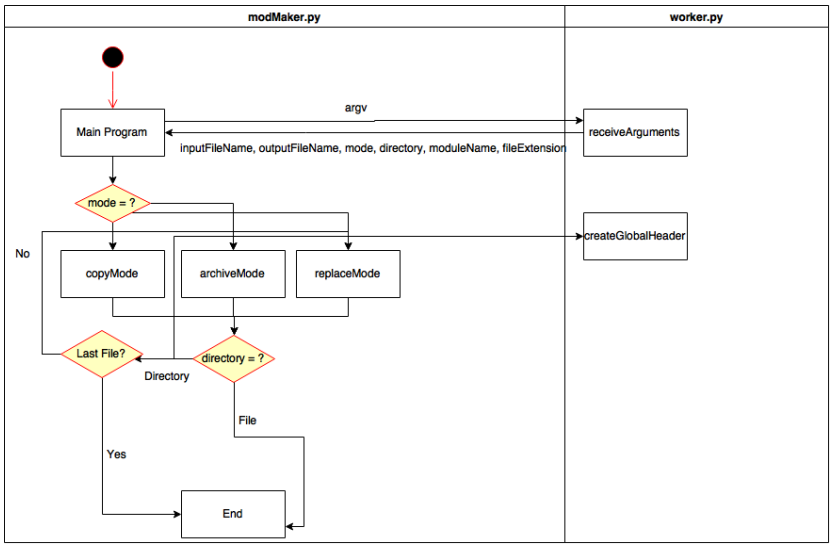

Figure 16: The flowchart of modMaker as a whole.

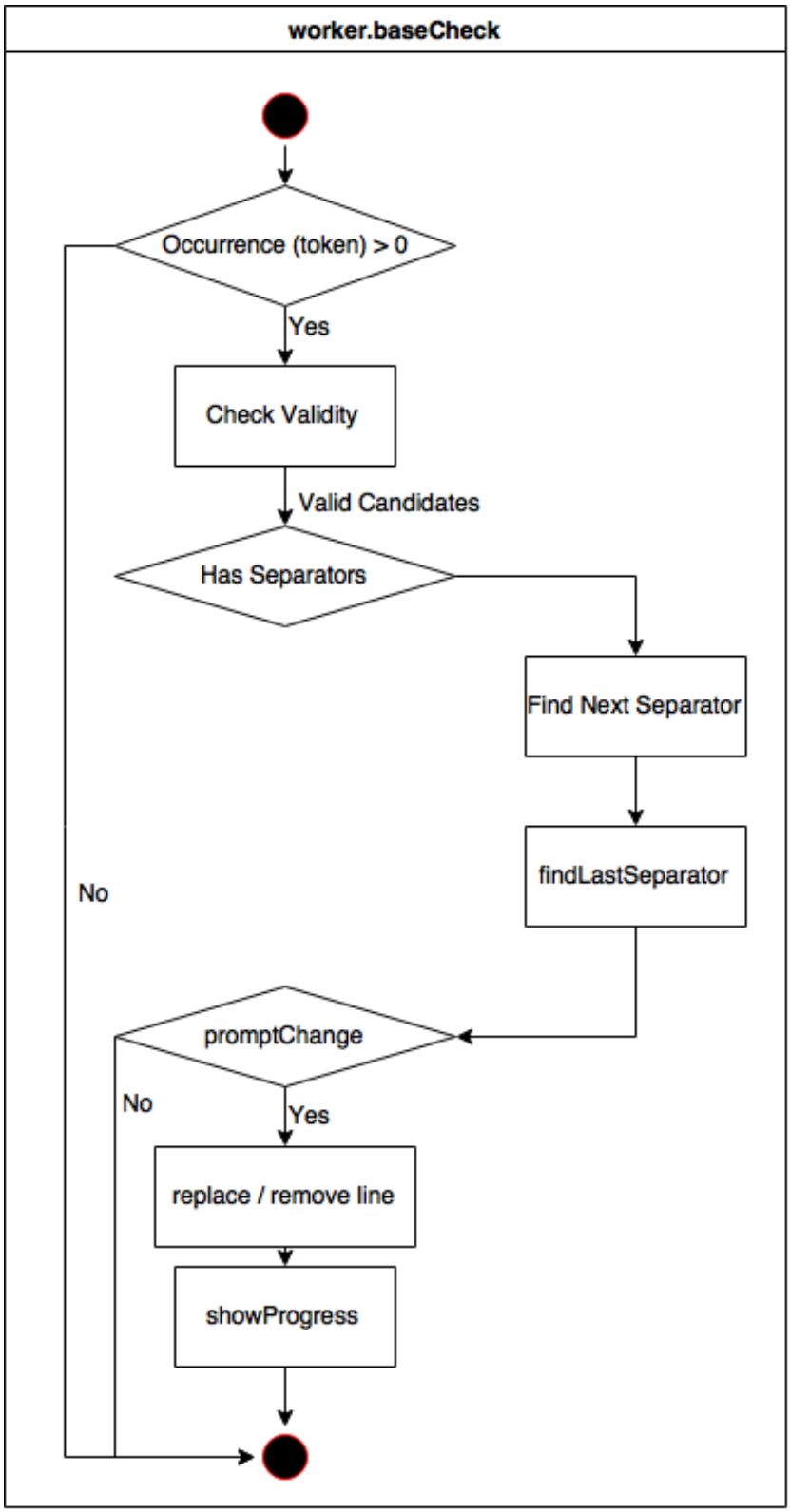

Figure 17: The flowchart of baseCheck, one of the fundamental checking programs

```
import worker
import sys
import os

tokensOfInterest = ["MPI_COMM_WORLD", "MPI_Finalize", "MPI_Init", "
    main(", " main "]
#include \"IEL.h\"\n
header = "#include \"MODULE_GLOBAL.h\"\n"

# Description:    Transform a file under the copy mode
# Usage:      copyMode (inputFileName, outputFileName, moduleName)

# Variables:    [string] inputFileName:   the file to be changed
#         [string] outputFileName:  the filename of the file to be
    output
#         [string] moduleName:    the name of module to be created

def copyMode (inputFileName, outputFileName, moduleName):

  global header, tokensOfInterest
  hasInterest = worker.init (inputFileName, tokensOfInterest)
  if not hasInterest:
    return

  (consents, cont) = worker.receiveConsent (inputFileName, ["
    MPI_COMM_WORLD"])
  if not cont:
    worker.showProgress ("%s has been left untouched." % (
    inputFileName))
    return

  try:
    inputFile = open (inputFileName, 'r')
    outputFile = open (outputFileName, 'w')
    outputFile.write (header)

    for line in inputFile:

      (foundMain, changedMain, headerCreated) = worker.getMain ()

      line = worker.checkFunctionTitle (line, moduleName)
      line = worker.checkStatement (line)
      line = worker.checkVariable (line, consents)

      if worker.foundMain and not worker.headerCreated:
        worker.createMainHeader (moduleName, outputFileName)

      if worker.foundMain and not worker.changedMain:
        line = worker.addGlobal (line)

      outputFile.write (line)
      worker.lineCounter += 1

    inputFile.close ()
    outputFile.close ()
  except:
    print "Unexpected error:", sys.exc_info()[0]
    if os.path.exists (outputFileName):
      os.remove (outputFileName)
    extention = len (inputFileName)
    print "
```

```
    " + "-" * (extention + 1)
    print "Module building in %s have been reverted while prior
    changes retained." % (inputFileName)
59  print "
    ------------------------------------------------------------
    " + "-" * (extention + 1)
    raise
61
# Description:    Transform a file under the replace mode
63# Usage:      replaceMode (inputFileName, outputFileName,
    moduleName)
65# Variables:    [string] inputFileName:   the file to be changed
#         [string] moduleName:    the name of module to be created
67
def replaceMode (filename, moduleName):
69
  global header, tokensOfInterest
71  hasInterest = worker.init (filename, tokensOfInterest)
  if not hasInterest:
73    return

75  (consents, cont) = worker.receiveConsent (filename, ["
    MPI_COMM_WORLD"])
  if not cont:
77    worker.showProgress ("%s has been left untouched." % (filename)
    )
    return
79
  inputFile = open (filename, 'r')
81  lines = inputFile.readlines ()
  inputFile.close ()
83
  try:
85    outputFile = open (filename, 'w')
    outputFile.write (header)
87
    for line in lines:
89
      (foundMain, changedMain, headerCreated) = worker.getMain ()
91
      line = worker.checkFunctionTitle (line, moduleName)
93    line = worker.checkStatement (line)
      line = worker.checkVariable (line, consents)
95
      if worker.foundMain and not worker.headerCreated:
97      worker.createMainHeader (moduleName, filename)

99    if worker.foundMain and not worker.changedMain:
        line = worker.addGlobal (line)
101
      outputFile.write (line)
103   worker.lineCounter += 1

105  outputFile.close ()

107 except:
    print "Unexpected error:", sys.exc_info()[0]
109  outputFile = open (filename, 'w')
    for line in lines:
111   outputFile.write (line)
    outputFile.close ()
```

```
113     extention = len (filename)
        print "
        ------------------------------------------------------------" +
         "-" * (extention + 1)
115     print "Changes in %s have been reverted while prior changes
        retained." % (filename)
        print "
        ------------------------------------------------------------" +
         "-" * (extention + 1)
117
# Description:    Transform a file under the archive mode
119 # Usage:       archiveMode (inputFileName, outputFileName,
      moduleName)

121 # Variables:    [string] inputFileName:   the file to be changed
#           [string] outputFileName:  the filename of the archived
      file
123 #           [string] moduleName:    the name of module to be created

125 def archiveMode (inputFileName, outputFileName, moduleName):

127   global header, tokensOfInterest
    hasInterest = worker.init (inputFileName, tokensOfInterest)
129   if not hasInterest:
      return
131
    (consents, cont) = worker.receiveConsent (inputFileName, ["
      MPI_COMM_WORLD"])
133   if not cont:
      worker.showProgress ("%s has been left untouched." % (
      inputFileName))
135     return

137   inputFile = open (inputFileName, 'r')
    lines = inputFile.readlines ()
139   inputFile.close ()

141   outputFileName = worker.sequenced (outputFileName)

143   try:
      outputFile = open (inputFileName, 'w')
145     archiveFile = open (outputFileName, 'w')

147     outputFile.write (header)

149     for line in lines:
        archiveFile.write (line)
151
        (foundMain, changedMain, headerCreated) = worker.getMain ()
153
        line = worker.checkFunctionTitle (line, moduleName)
155     line = worker.checkStatement (line)
        line = worker.checkVariable (line, consents)
157
        if worker.foundMain and not worker.headerCreated:
159       worker.createMainHeader (moduleName, inputFileName)

161     if worker.foundMain and not worker.changedMain:
          line = worker.addGlobal (line)
163
        outputFile.write (line)
165     worker.lineCounter += 1
```

```
167       outputFile.close ()
          archiveFile.close ()
169     except:
          print "Unexpected error:", sys.exc_info()[0]
171       outputFile = open (inputFileName, 'w')
          for line in lines:
173         outputFile.write (line)
          outputFile.close ()
175       if os.path.exists (outputFileName):
            os.remove (outputFileName)
177       extention = len (inputFileName)
          print "
          ----------------------------------------------------------------------" +
           "-" * (extention + 1)
179       print "Changes in %s have been reverted while prior changes
          retained." % (inputFileName)
          print "
          ----------------------------------------------------------------------" +
           "-" * (extention + 1)
181       raise

183   status = ""
      (inputFileName, outputFileName, mode, directory, moduleName,
          fileExtension) = worker.receiveArguments (sys.argv)
185   option = "1"
      while option != "" and option != "exit":
187     option = raw_input ("\nPress Enter to Continue or type \"exit\"
          to quit: ")

189   if option == "exit":
        sys.exit (0)
191
      if directory == "F":
193     if mode == "C": #-----------------------------------

195       copyMode (inputFileName, outputFileName, moduleName)

197     elif mode == "R": #---------------------------------

199       replaceMode (inputFileName, moduleName)

201     elif mode == "A": #---------------------------------

203       if not os.path.exists (outputFileName):
            try:
205           os.makedirs (outputFileName)
            except OSError:
207           worker.displayMsg ("We do not have the permission to create
           a directory OR disk space is full OR directory already exists.
           Please either:\n\n 1)\t create a directory;\n 2)\t remove some
           files; OR\n 3)\t use another mode\n")
              sys.exit (1)
209
          outputFileName = os.path.join (outputFileName, inputFileName)
211       archiveMode (inputFileName, outputFileName, moduleName)

213   elif directory == "D":
        if mode == "C":
215
          if not os.path.exists (outputFileName):
217         try:
```

```
            os.makedirs (outputFileName)
219       except OSError:
            worker.displayMsg ("We do not have the permission to create
        a directory OR disk space is full OR directory already exists.
        Please either:\n\n 1)\t create a directory;\n 2)\t remove some
        files; OR\n 3)\t use another mode\n")
221         sys.exit (1)

223     worker.createGlobalHeader (outputFileName)

225     for dirname, dirnames, filenames in os.walk(inputFileName):
            for filename in filenames:
227           if not filename.startswith ('.'):

229           iName = os.path.join (dirname, filename)
              ext = os.path.splitext (filename)[1].lower()
231
              if ext in fileExtension:
233             oName = os.path.join (outputFileName, filename)
                oName = worker.sequenced (oName)
235
                worker.showProgress ("\n======In file %s======\n" % (
    iName))
237             copyMode (iName, oName, moduleName)

239   elif mode == "R":

241     worker.createGlobalHeader (inputFileName)

243     for dirname, dirnames, filenames in os.walk(inputFileName):
            for filename in filenames:
245           if not filename.startswith ('.'):
              iName = os.path.join (dirname, filename)
247           ext = os.path.splitext (filename)[1].lower()

249           if ext in fileExtension:
                worker.showProgress ("\n======In file %s======\n" % (
    iName))
251             replaceMode (iName, moduleName)

253   elif mode == "A":

255     worker.createGlobalHeader (inputFileName)

257     if not os.path.exists (outputFileName):
          try:
259         os.makedirs (outputFileName)
          except OSError:
261         worker.displayMsg ("We do not have the permission to create
        a directory OR disk space is full OR directory already exists.
        Please either:\n\n 1)\t create a directory;\n 2)\t remove some
        files; OR\n 3)\t use another mode\n")
            sys.exit (1)
263
      for dirname, dirnames, filenames in os.walk(inputFileName):
265       for filename in filenames:
            if not filename.startswith ('.'):
267         iName = os.path.join (dirname, filename)
            ext = os.path.splitext (filename)[1].lower()
269
            if ext in fileExtension:
271           worker.showProgress ("\n======In file %s======\n" % (
```

```
    iName))
            archiveMode (iName, os.path.join (outputFileName,
    filename), moduleName)

worker.showProgress ("
    ==========================================================
    ")
worker.showProgress ("We have completed the transformation.")
worker.showProgress ("Please note that \"argc\" and \"argv\" in
    main program have to be changed into")
worker.showProgress ("\"exec_info->modules->mod_argc\" and \"
    exec_info->modules->mod_argv\" respectively")
worker.showProgress ("
    ==========================================================
    ")
```

modMaker.py

```
# This is the worker program that uses python to do work
# It can read in a c file and return a "package" used under the IEL
    model

import sys
from array import *
import os

#Global Flags

#initRemoved = False
#finRemoved = False
#mainRemoved = False

#Global Buffer

status = ""
lineCounter = 1
lines = []
foundMain = False
changedMain = False
headerCreated = False
#changed = False

################## Display Functions ##################

# Clears the screen for displaying messages
def clearScreen ():
  print("\033c")

# Shows debug messages without being cleared (not exactly by not
    clearing)
def debug (s):
  showProgress (s)

# Shows the archived "more-important" messages that would "not be
    cleared"
def displayArchive ():
  clearScreen ()
  print status

# Display a one-time message
def displayMsg (s):
```

```
    displayArchive ()
42  print s

44# Shows progress and archive the progress
def showProgress (s):
46  global status
    status = status + s + "\n"
48  displayArchive ()

50# Displays a line with its highlight beneath certain region
def displayWithHighlight (lineNumber, line, start, end):
52  print "%d)  %s" % (lineNumber, line[:len(line) - 1])
    count = line.count ("\t")
54
    tempString = "\t" * (count + 1)
56  tempString = tempString + " " * (start - count)
    tempString = tempString + "-" * (end - start + 1)
58  tempString = tempString + "\n"

60  print tempString

62################## Worker Functions ##################

64# Description:    Validate user-input path
# Usage:      pathName = validatePath (pathName, True)
66
# Variables:    [str] pathName: user-input pathname
68#         [bool]  existing: whether an existing or non-existing
      pathName is needed
#         [bool]  directory:  whether a directory or file pathName
      is needed, true refers to directory
70
# Returns     [str] a valid pathName as-per required
72
def validatePath (pathName, existing, directory):
74
  while existing and pathName == "":
76  if directory:
      pathName = raw_input ("Please specify a directory for
      transformation: ").strip()
78  else:
      pathName = raw_input ("Please specify a file for
      transformation: ").strip()
80
  if existing and directory:
82  while not os.path.exists (pathName):
      print "\"%s\" doesn't exists" % (pathName)
84  pathName = raw_input ("Please enter an existing dirname: ").
      strip()
    while not os.path.isdir (pathName):
86  print "\"%s\" is not a directory" % (pathName)
      pathName = raw_input ("Please enter an existing dirname: ").
      strip()
88
  if existing and not directory:
90  while not os.path.exists (pathName):
      print "\"%s\" doesn't exists" % (pathName)
92  pathName = raw_input ("Please enter an existing filename: ").
      strip()
    while not os.path.isfile (pathName):
94  print "\"%s\" is not a file" % (pathName)
      pathName = raw_input ("Please enter an existing filename: ").
```

```
    strip()
96
98  if not existing and directory:
      while os.path.exists (pathName):
100     print "\"%s\" exists already" % (pathName)
        pathName = raw_input ("Please enter a non-existing dirname or
        leave blank for auto-generated directory: ").strip()
102
    if not existing and not directory:
104   while os.path.exists (pathName):
        print "\"%s\" exists already" % (pathName)
106     pathName = raw_input ("Please enter a non-existing filename
      or leave blank for auto-generated filename: ").strip()
108
    return pathName
110
# Description:    Find the previous occurance of string s in line
    with respect of startLocation
112 # Usage:       void function (); int main ();
# This function can locate the first ";"
114
# Variables:    [str] line:     the line in question
116 #         [int] startLocation:  the token must be before this
    position
#         [str] s:        token in question
118 # Returns     [int] location of token. -1 if not found.

120 def findLastSeperator (line, startLocation, s):

122   location = line.find (s)
    prev = location
124
    if location == -1 or location > startLocation:
126   return -1

128   while (location < startLocation):
      prev = location
130   location = line.find (s, location + 1)
      if location == -1:
132     return prev

134   return prev


136


138 # Description:    Determines if a given character is alpha-numeric.
      (10 numbers, 26 characters of upper- and lowercases and
    underscore)
    # Usage:       if isAlphanumeric (c): ...
140
# Variables:    [char]  c:  character in question
142 # Returns     [bool]  True if c is alphanumeric. False if c is not

144 def isAlphanumeric (c):

146   valid = False
    if (ord (c) >= 48 and ord (c) <= 57):
148   valid = True
    if (ord (c) >= 65 and ord (c) <= 90):
150   valid = True
```

```
  if (ord (c) >= 97 and ord (c) <= 122):
    valid = True
  if (ord (c) == 95):
    valid = True

  return valid


# Description:    Checks if a sub-line contains only spaces (or space-equivalent, ie. tab)
# Usage:       if spaceOnly (line, startLocation, endLocation): {do sth}

# Variables:    [str] line:     the line in question
#         [int] startLocation:  start of substring
#         [int] endLocation:  end of substring

# Example:       0 1 2 3 4 5 6 7 8 9
#         w o r d     w o r d
#         Pass in (line, 4, 6)
# Returns     [bool]  True if there's only spaces. False otherwise.

def spaceOnly (line, startLocation, endLocation):

  if startLocation >= endLocation:
    return False

  valid = True

  for counter in xrange (endLocation - startLocation):
    if line [startLocation + counter] != ' ' and line [startLocation + counter] != '\t':
      valid = False

  return valid

# Description:    To prevent overwriting existing files by adding numbers at the back
# Usage:       outputFileName = sequenced (outputFileName)

# Variables:    [str] filename: the original filename
# Returns     [str] filename: the "safe", "updated" filename

def sequenced (filename):

  temp = filename
  counter = 1

  while os.path.exists (temp):
    temp = filename + "_" + str(counter)
    counter += 1

  return temp



################## User Communication ##################

# Description:    Let the user decide whether or not to make a change
# Usage:       if promptChange (line, ln, start, end, reason): {do sth}
```

```
208 # Variables:    [str] line:     the line in question
    #          [int] linenumber:   linenumber to be displayed
210 #          [int] start:      start of questionable segment
    #          [int] end:      end of questionable segment
212 #          [str] reason:     reason for change
    # Returns      [bool]  True if user accepts the change. False if not
        .
214
    def promptChange (line, lineNumber, start, end, reason):
216
      if start == end:
218     return line

220   print "We have found something that may need to be changed in
        your code for the following reason:\n"
      print "\t%s\n" % (reason)
222
      end = end - 1
224
      while (True):
226
        displayWithHighlight (lineNumber, line, start, end)
228
        option = raw_input ("If you agree on this change press [Enter]
        or key in [y/Y]. If it is incorrect, key in [n/N] ").strip()
230
        if option == "" or option == "y" or option == "Y":
232       return True
        elif option == "n" or option == "N":
234       return False
        else:
236       print "Unrecognized input. Please try again."


238
    # Description:    Receive file related inputs and options from user
240 # Usage:      (inputfile, outputfile, mode, directory, moduleName)
        = receiveArguments (argv)

242 # Variables:    [list]  argv: System arguments vector
    # Returns      [tuple] tuple of (input_filename, output_filename,
        mode, directory, module_name)
244 # Representations:  modes:  A will replace original files which
        will then be put into an archive
    #             R replaces without archiving
246 #             C duplicates before changing. Original copy is not
        affected
    #         direc:  D means that modMaker makes changess on a
        directory
248 #             F means that modMaker makes changes on a file

250 def receiveArguments (argv):
      clearScreen ()
252   mode = "C"
      directory = "F"
254   modeSpec = False
      dirSpec = False
256
      li = sorted (sys.argv[1:])
258   counter = 0

260   while (counter < len(li) and li[counter][0] == "-"):
```

```
262     if len(li[counter]) == 1:
          print "Please check on input format and spacing. Unrecognized
        Input!"
264       sys.exit (1)

266     elif li[counter][1] == "r" or li[counter][1] == "R":
          if not modeSpec:
268         mode = "R"
            modeSpec = True
270       else:
            print "Overlapped Flags!! %s, %s" % ("-" + mode, "-R")
272         sys.exit (1)

274     elif li[counter][1] == "c" or li[counter][1] == "C":
          if not modeSpec:
276         mode = "C"
            modeSpec = True
278       else:
            print "Overlapped Flags!! %s, %s" % ("-" + mode, "-C")
280         sys.exit (1)

282     elif li[counter][1] == "a" or li[counter][1] == "A":
          if not modeSpec:
284         mode = "A"
            modeSpec = True
286       else:
            print "Overlapped Flags!! %s, %s" % ("-" + mode, "-A")
288         sys.exit (1)

290     elif li[counter][1] == "d" or li[counter][1] == "D":
          if not dirSpec:
292         directory = "D"
            dirSpec = True
294       else:
            print "Overlapped Flags!! %s, %s" % ("-" + directory, "-D")
296         sys.exit (1)

298     elif li[counter][1] == "f" or li[counter][1] == "F":
          if not dirSpec:
300         directory = "F"
            dirSpec = True
302       else:
            print "Overlapped Flags!! %s, %s" % ("-" + directory, "-F")
304         sys.exit (1)

306     elif li[counter][1] != "o" and li[counter][1] != "O":
          print "Unrecognized Flag %s" % (li[counter])
308
        counter += 1
310
    counter = 1
312 inputFileName = ""
    outputFileName = ""
314 moduleFlag = False
    moduleName = ""
316
    while (counter < len (sys.argv)):
318   if moduleFlag and moduleName == "":
        moduleName = sys.argv [counter]
320
      elif sys.argv [counter][0] != "-":
```

```
322         if inputFileName == "":
              inputFileName = sys.argv [counter]
324         elif outputFileName == "":
              outputFileName = sys.argv [counter]
326
        if sys.argv [counter] == "-o" or sys.argv [counter] == "-O":
328         if not moduleFlag:
              moduleFlag = True
330         else:
              print "Overlapped Flags!! %s, %s" % (sys.argv [counter], "-
          O")
332           sys.exit (1)
334       counter += 1
336     if mode == "C": #-----------------------------------
        if directory == "F":
338       inputFileName = validatePath (inputFileName, True, False)
          outputFileName = validatePath (outputFileName, False, False)
340     else:
          inputFileName = validatePath (inputFileName, True, True)
342       outputFileName = validatePath (outputFileName, False, True)
344     elif mode == "R": #---------------------------------
        if directory == "F":
346       inputFileName = validatePath (inputFileName, True, False)
        else:
348       inputFileName = validatePath (inputFileName, True, True)
350     if outputFileName != "":
          displayMsg ("You have chosen to use the replace option.
        Changes will be made directly to the original file and %s doesn
        't matter\n" % (outputFileName))
352
          warning = raw_input ("Type \"exit\" to end program or [Enter]
         to continue: ")
354       while warning != "" and warning != "exit":
            warning = raw_input ("Type \"exit\" to end program or [
        Enter] to continue: ")
356
          if warning == "exit":
358         sys.exit (1)
360     elif mode == "A": #---------------------------------
        if directory == "F":
362       inputFileName = validatePath (inputFileName, True, False)
        else:
364       inputFileName = validatePath (inputFileName, True, True)
366     if directory == "D":
          dirD = "directory"
368     else:
          dirD = "file"
370
        clearScreen()
372     if moduleName == "":
          moduleName = "moduleMain"
374
        if mode == "A":
376
          if outputFileName == "":
378         outputFileName = "Archive_modMaker"
```

```
380     (head, tail) = os.path.split (inputFileName)
382     if head:
          outputFileName = os.path.join (head, outputFileName)
384
        showProgress ("Transforming %s \"%s\" into module \"%s\"" % (
        dirD, inputFileName, moduleName))
386     showProgress ("Old files will be put into directory %s" % (
        outputFileName))
388   elif mode == "R":
        showProgress ("Transforming %s \"%s\" into module \"%s\" which
        will replace the original %s" % (dirD, inputFileName,
        moduleName, dirD))
390     outputFileName = ""
392   elif mode == "C":
394     if outputFileName == "" and directory == "F":
          (head, tail) = os.path.split (inputFileName)
396       outputFileName = "module_" + tail
398       if head:
            outputFileName = os.path.join (head, outputFileName)
400
          outputFileName = sequenced (outputFileName)
402
        if outputFileName == "" and directory == "D":
404
          (head, tail) = os.path.split (inputFileName)
406       outputFileName = "Module_" + tail
408       if head:
            outputFileName = os.path.join (head, outputFileName)
410
          outputFileName = sequenced (outputFileName)
412
        showProgress ("Duplicating %s \"%s\" into a module under the
        name \"%s\" and filename \"%s\"" % (dirD, inputFileName,
        moduleName, outputFileName))
414     showProgress ("Original files will not be lost")
416   if directory == "D":
        displayMsg ("What file types should we focus on? Please
        optionally enter extensions one by one and end with a blank
        line.\n(Include \".\" and use small letters. Example: \".c\")\n
        ");
418     extension = list ()
        ext = raw_input ("Extension: ")
420     while ext != "":
          extension.append (ext)
422       ext = raw_input ("Extension: ")
424   return (inputFileName, outputFileName, mode, directory,
        moduleName, extension)
426
428 # Description:    Receive consent to speed up replacement process
    # Usage:      consents = (inputFileName, strings)
430
```

```
# Variables:    [str]   inputFileName:  the file to undergo the
    process
#         list[str] stringList:   the strings to look for
# Returns     list[bool]  consents:   the consents [True, True,
    False, ...] compiled
#         [bool]    continue:   whether or not the file should
    continue transformation

def receiveConsent (inputFileName, stringList):

  if len (stringList) == 0:
    return None

  if not os.path.exists (inputFileName):
    debug ("File doesn't exists")
    sys.exit (1)

  consent = []

  option = "s"
  while (option != "" and option != "N" and option != "n"):
    option = raw_input ("We are about to conduct transformation on
    %s.\nPress [Enter] to begin or key-in \"n/N\" to skip this file
    .  " % (inputFileName))

  if option == "":

    inputFile = open (inputFileName, 'r')
    lines = inputFile.readlines ()
    inputFile.close ()

    for string in stringList:

      displayArchive()
      lineNumber = 1
      counter = 0

      for line in lines:
        start = line.find (string)
        if start != -1:
          displayWithHighlight (lineNumber, line, start, start +
    len(string))
          counter += 1
        lineNumber += 1

      if counter != 0:
        option = "s"
        while (option != "" and option != "N" and option != "n"):
          option = raw_input ("We have found %d occurances of %s.\
    nPress [Enter] to authorize every change or key-in \"n/N\" to
    authorize one-by-one." % (counter, string))

        if option == "":
          consent.append (True)

          if counter == 1:
            showProgress ("%d %s has been replaced." % (counter,
    string))
          else:
            showProgress ("%d %s have been replaced." % (counter,
    string))
        else:
```

```
            consent.append (False)
484
            if counter == 1:
486           showProgress ("Suspected %s remains unchanged." % (
      string))
            else:
488           showProgress ("%d suspected %ss remain unchanged." % (
      counter, string))

490       else:
          consent.append (False)
492
      return (consent, True)
494
    else:
496
      for string in stringList:
498     consent.append (False)

500   return (consent, False)

502 def init (filename, tokenList):
    #global changed
504 #changed = False

506 file = open (filename, 'r')
    lines = file.readlines ()
508 file.close ()

510 lineCounter = 1

512 for line in lines:
      for token in tokenList:
514     if line.find (token) != -1:
          debug (token)
516       return True

518 return False

520 def createGlobalHeader (path):
    if os.path.isfile (path):
522   debug ("Directory path required to create global header.")
      return
524
    filename = os.path.join (path, "MODULE_GLOBAL.h")
526 file = open (filename, 'w')
    file.write ("#include \"IEL.h\"\n\nextern IEL_exec_info_t *
      exec_info;\n")
528 filename = os.path.join (path, "IEL.h")
    file = open (filename, 'w')
530 file.write ("#include \"mpi.h\"\n#ifndef IEL_H\n#define IEL_H\
      ntypedef struct module_depend_t {\n\tint mod_argc;\n\tchar**
      mod_argv;\n}module_depend_t;\n\n")
    file.write ("typedef struct IEL_exec_info_t {\n\tMPI_Comm
      module_copy_comm;\n\tstruct module_depend_t* modules;\n}
      IEL_exec_info_t;\n#endif")
532 file.close ()

534 return

536 def createMainHeader (moduleName, path):
```

```
538   global headerCreated

540   headerCreated = True

542   if not os.path.isfile (path):
        debug ("Main header file creation error. Require filepath to
        main.")
544     return

546   # ------------- Creating Module Header ---------------- #

548   name = os.path.splitext (path) [0]
      headername = name + ".h"
550
      if os.path.exists (headername):
552     debug ("Main header already exists. Please manual declare
        module in header.")
      else:
554     file = open (headername, 'w')
        file.write ("#include \"IEL.h\"\n")
556     file.write ("int " + moduleName + " (IEL_exec_info_t *
        iel_exec_info);")
        showProgress ("Header file \"" + headername + "\" created
        successfully.")
558     file.close ()

560   # ------------- Creating Test Main.c ---------------- #

562   name = os.path.dirname (path)
      filename = os.path.join (name, "IEL_main.c")
564   filename = sequenced (filename)
      debug (filename)
566
      file = open (filename, 'w')
568
      file.write ("#include \"MODULE_GLOBAL.h\"\n")
570   file.write ("#include \"")
      file.write (os.path.basename (headername))
572   file.write ("\"\n")
      file.write ("#include <stdlib.h>\n\n")
574
      file.write ("int main(int argc, char* argv[]){\n")
576   file.write ("\tMPI_Init (&argc, &argv);\n")
      file.write ("\tIEL_exec_info_t *temp = (IEL_exec_info_t *) malloc
        (sizeof (IEL_exec_info_t));\n")
578   file.write ("\ttemp -> module_copy_comm = MPI_COMM_WORLD;\n")
      file.write ("\ttemp -> modules = (module_depend_t *) malloc (
        sizeof (module_depend_t));\n")
580   file.write ("\ttemp -> modules -> mod_argc = argc;\n")
      file.write ("\ttemp -> modules -> mod_argv = argv;\n")
582   file.write ("\tmoduleMain (temp);\n")
      file.write ("\tfree (temp -> modules);\n")
584   file.write ("\tfree (temp);\n")
      file.write ("\tMPI_Finalize();\n")
586   file.write ("\treturn 0;\n")
      file.write ("}")
588
      file.close ()
590   showProgress ("Test main created successfully.")

592   return
```

```
594 def setGlobal (line):
596   global changedMain
598   location1 = line.find ("iel_exec_info")
      location2 = line.find ("{", location1)
600   if location2 != -1:
        debug ("Main is changed here")
602     changedMain = True
        return (line[:location2 + 1]  + "\n\texec_info = iel_exec_info
        ;\n" + line[location2 + 1:])
604   else:
        changedMain = False
606     return line
608 def addGlobal (line):
610   global changedMain
612   location2 = line.find ("{")
      if location2 == -1:
614     return line
      else:
616     changedMain = True
        debug ("Main is changed here")
618     return (line[:location2 + 1]  + "\n\texec_info = iel_exec_info
        ;\n" + line[location2 + 1:])
620
    def getMain ():
622   return (foundMain, changedMain, headerCreated)
624 ################## Checking Functions #################
626 # Wrapper function for checking all function titles
    def checkFunctionTitle (line, moduleName):
628   line = checkMain (line, moduleName)
      return line
630
    # Wrapper function for checking all in-line statements
632 def checkStatement (line):
      line = checkInit (line)
634   line = checkFinalize (line)
      return line
636
    # Wrapper function for checking all in-line variables
638 def checkVariable (line, consents):
      if consents [0]:
640     line = line.replace ("MPI_COMM_WORLD", "exec_info->
        module_copy_comm")
      else:
642     line = checkMPICommWorld (line)
644 # if foundMain:
    #   line = checkArgc (line)
646 #   line = checkArgv (line)
648   return line
650 # Defines parameters to check the main
    def checkMain (line, moduleName):
652   msgB = []
```

```
    msgB.append ("Only one Main can exist and it belongs in the
      driver.c")
654 msgB.append ("Main has been changed to " + moduleName)
    msgB.append ("Suspected Main remained unchanged")
656 replacement = "IEL_exec_info_t* exec_info;\n" + "int " +
      moduleName + " (IEL_exec_info_t *iel_exec_info)"
    return baseCheck (line, ["int", "main"], "}", ")", msgB,
      replacement, "")
658
# Defiens parameters to check MPI_Init
660 def checkInit (line):
    msgB = []
662 msgB.append ("Only one Init can exist and it belongs in the
      driver.c")
    msgB.append ("MPI_Init has been removed")
664 msgB.append ("Suspected MPI_Init remained unchanged")
    return baseCheck (line, ["MPI_Init"], ";", ";", msgB, "", "int")
666
# Defines parameters to check MPI_Finalize
668 def checkFinalize (line):
    msgB = []
670 msgB.append ("Only one Finalize can exist and it belongs in the
      driver.c")
    msgB.append ("MPI_Finalize has been removed")
672 msgB.append ("Suspected MPI_Finalize remained unchanged")
    return baseCheck (line, ["MPI_Finalize"], ";", ";", msgB, "", "
      int")
674
# Defines parameters to check MPI_COMM_WORLD
676 def checkMPICommWorld (line):
    msgB = []
678 msgB.append ("MPI_COMM_WORLD has been divided into
      subcommunicators")
    msgB.append ("MPI_COMM_WORLD has been replaced")
680 msgB.append ("Suspected MPI_COMM_WORLD remained unchanged")
    return baseCheck (line, ["MPI_COMM_WORLD"], "", "", msgB, "
      exec_info->module_copy_comm", "")
682
# Defines parameters to check argc
684 # def checkArgc (line):
#   msgB = []
686 #   msgB.append ("argc is not directly visible at this level")
#   msgB.append ("argc has been replaced")
688 #   msgB.append ("Suspected argc remained unchanged")
#   return baseCheck (line, ["argc"], "", "", msgB, "exec_info->
    modules->mod_argc", "")
690
# Defines parameters to check argv
692 # def checkArgv (line):
#   msgB = []
694 #   msgB.append ("argv is not directly visible at this level")
#   msgB.append ("argv has been replaced")
696 #   msgB.append ("Suspected argv remained unchanged")
#   return baseCheck (line, ["argv"], "", "", msgB, "exec_info->
    modules->mod_argv", "")
698

700 # Description:    Base checking function which is versatile and
      powerful
# Usage:      line = baseCheck (...)
702
# Variables:    [str] line:     the line in question
```

```
704 #         [list]  tokenList:    list of tokens to match to
    #         [str] prevSep:    Separator that is likely to lead the
        segment in question. Replacement starts right afterwards.
        Starts from start of token match if omitted.
706 #         [str] nextSep:    Separator that is likely to follow the
        segment. Replacements continues to this point. Ends at end of
        token matach if omitted.
    #         [list]  msgBundle:    0: contains "reasons", why the
        segment needs to be replaced / removed
708 #                     1: contains "successful msg", showing that
        the segment was properly dealt with
    #                     2: contains "unchanged msg", showing that the
         segment was not affected
710 #         [str] replacement:  if omittied, substring in concern is
        commented out. Otherwise, provides replacement for substring.
    #           [str] returnsValue: if set, the replacement concerns a
        function which returns value type specified. This only affects
        "commenting out"
712 # Returns:      line that can be altered or unchanged

714 def baseCheck (line, tokenList, prevSep, nextSep, msgBundle,
        replacement, returnsValue):

716   global foundMain
      firstPos = []
718
      for token in tokenList:
720     pos = line.find (token)
        if pos == -1:
722       return line
        else:
724       firstPos.append (pos)

726   candidates = findCandidates (line, tokenList, firstPos)

728   if replacement == "":
        replace = False
730   else:
        replace = True
732
      for (begin, end) in reversed (candidates):
734     valid = True
        if begin != 0:
736       if isAlphanumeric (line [begin - 1]):
            valid = False
738
        if isAlphanumeric (line [end]):
740       valid = False

742     if valid:

744       typePos = -1

746       if nextSep != "":
            end = line.find (nextSep, end + 1) + 1
748
          if returnsValue != "":
750         typePos = findLastSeperator (line, begin, returnsValue)

752       if prevSep != "":
            temp = findLastSeperator (line, begin, prevSep)
754         if temp != -1:
```

```
            begin = temp + 1
        else:
            begin = 0

      if promptChange (line, lineCounter, begin, end, msgBundle
    [0]):
        if replace:
            output = ""
            output = output + line [:begin]
            output = output + replacement
            output = output + line [end:]
            showProgress (msgBundle [1])
            line = output
        else:
            output = ""

            if typePos != -1 and typePos >= begin and typePos < end:
                equals = line.find ("=", typePos)
                if equals != -1 and equals > typePos:
                    if line[equals - 1] == " ":
                        output = output + line [:equals - 1]
                    else:
                        output = output + line [:equals]

                    output = output + ";\n"
                elif begin != 0:
                    output = output + line [:begin]
                    output = output + "\n"

            output = output + "//"
            output = output + line [begin : end]
            if line[end] != "\n":
                output = output + "\n"
            output = output + line [end:]
            showProgress (msgBundle [1])
            line = output

#       changed = True
        if len(tokenList) > 1 and tokenList [1] == "main":
            foundMain = True
            line = setGlobal (line)
      else:
        showProgress (msgBundle [2])

  return line




# Description:    Verstaile token matching function
# Usage:      candidates = findCandidates (line, tokenList,
    firstPos)

# Variables:    [str] line:     the line in question
#         [list]  tokenList:    list of tokens to match to
#         [str] prevSep:    Separator that is likely to lead the
    segment in question. Replacement starts right afterwards.
    Starts from start of token match if omitted.
#         [str] nextSep:    Separator that is likely to follow the
    segment. Replacements continues to this point. Ends at end of
    token matach if omitted.
```

```
#         [list] msgBundle:    0: contains "reasons", why the segment needs to be replaced / removed
#                              1: contains "successful msg", showing that the segment was properly dealt with
#                              2: contains "unchanged msg", showing that the segment was not affected
#         [str] replacement:  if omittied, substring in concern is commented out. Otherwise, provides replacement for substring.

def findCandidates (line, tokenList, firstPos):

  locationDictionary = []
  result = []
  counter = 0
  arraySize = len (tokenList)

  for token in tokenList:

    locationArray = []

    while (firstPos [counter] != -1):
      locationArray.append (firstPos [counter])
      firstPos [counter] = line.find (tokenList[counter], firstPos [counter] + 1)

    counter += 1
    locationDictionary.append (locationArray)

  accessArray = []
  for i in xrange (arraySize):
    accessArray.append (0)

  while (True):
    valid = True
    for i in xrange (arraySize - 1):
      j = accessArray [i]
      k = accessArray [i + 1]

      if locationDictionary [i][j] > locationDictionary [i+1][k]:
        valid = False
        accessArray [i + 1] += 1
        if accessArray [i + 1] >= len (locationDictionary [i + 1]):
          return result
      elif not spaceOnly (line, locationDictionary [i][j] + len (tokenList [i]), locationDictionary [i + 1][k]):
        valid = False
        for c in xrange (i + 1):
          accessArray [c] += 1
          if accessArray [c] >= len (locationDictionary [c]):
            return result
    if valid:
      begin = locationDictionary [0][accessArray [0]]
      end = locationDictionary [arraySize - 1][accessArray [arraySize - 1]] + len (tokenList [arraySize - 1])
      result.append ((begin, end))
      accessArray [i] += 1
      if accessArray [i] >= len (locationDictionary [i]):
        return result

##################
```

worker.py