

**MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING**

**CENG 491
SENIOR DESIGN PROJECT
USER GUIDE**

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

Installing Cstar Framework:

1) Extracting: Untar the file(MYDEV.RAR) to any directory you want (tar xvf mydev.tar)

In Mydev directory, there must be 2 directories (cf and lcc)

2) Setting path: Add the path to your path environment. You must open /home/username/.bashrc and place "export PATH=\$PATH:/YOURSETUPPATH/mydev/cf/exe" YOURSETUPPATH is the path where you untar the file.

3) Make: In MYDEV/cf/src directory, you can call makes.

all: generates doc tags stat srcs exes

doc: generates doxygen docs (Doxygen must be under cf/tools directory. Else, you can untar to a directory and add your bashrc file.)

exes: generates bcc (If you have chosen debugbuild mode dbccbin will be created, otherwise in releasebuild mode bbcbn will be created. Bbcbn is faster than dbccbin)

clean: deletes entire object files and libraries produced. (It can be necessary when doing some drastic changes.)

packbins: Source files are zipped in to a file. (This can be useful for security reasons.)

unpackbins: source files are extracted from zip file. You should set PASSWD=xxx to unpack the files that are packed with packbins.

stat: generates some statistics about the codes(That will add the number of lines of source and header files under cf/src and cf/inc directories.)

bin: generates srcs exes

srcs: generates fesrcs besrcs

fesrcs: generates front-end related sources

besrcs: generates back-end related sources

pch: generates the precompiled header file which contains a list of standard header files.

clobber: does clean and also deletes executables

timings: outputs the timings

BUILDMODE=DEBUGBUILD should be used for debug mode build

BUILDMODE=RELEASEBUILD should be used for release mode build

example usage In MYDEV/cf/src directory:

make clean

make clobber

make exes

Attention! Framework is ready for 64-bit processors. If you have a 32-bit processor, you must exchange packbins, which are for 32-bit, with you have created.

Usage:

Optimizations:

From the directory /mydev/cf/exe call dbccbin with level -O0 for Group Hellim optimizations.

```
./dbccbin TARGETFILE.C -O0 -target=x86/linux
```

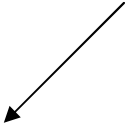
Optimization Manager:

The optimization manager will be handled via "optman.txt" file.

Example usage:

Add the below lines to "optman.txt"

```
TutConstFold(IRProgram(repeat_on_change))  
TutUnreachableCode(IRLoop(mem_clean_up))
```



In the file "optman.txt" you can find the names of Group Hellim optimizations and original CStar optimizations at the beginning of the file.

```
./dbccbin TARGETFILE.C -O0 -target=x86/linux
```

In this case Constant Folding and Unreachable Code annotations will be called from the framework.

Test Case Generator:

To call test case generator, you must put the below line in optman.txt

```
TestCase(IRProgram(repeat_on_change))
```

Generator Options for Test Case Generator

From the "tcg.txt" you can set the test case generation options like number of statements or generating a number seed value for random code generation.

Test Case Generator module needs a empty c file which only contains
int main(){return 0;}(which is a.c for simple testing in the framework)

Outputs:

Optimizations results are dumped to files in the directory `home/anil/mydev/`

Test Cases

From the `mytest.tar` file you can find the C files; which you can test the optimizations. This file also contains "optman.txt", "tcg.txt" and also the "dtconfig.cf2g" files.