C++ CODE CHECKER

Project work done at P-790

Think Business Networks, Coimbatore.

PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree of

Master of Computer Applications of Bharathiar University, Coimbatore.

Submitted by

Balu.N
Reg.No 9938M0601

Guided by

Internal guide
Mrs. S.Devaki B.E, M.S.

External guides
Mr. S.Chandra Kumar.
Mr. C.Senthilnathan.

Department of Computer Science
Kumaraguru College of Technology
Coimbatore-641006
May-2002

Department of Computer Science

Kumaraguru College of Technology

(Affiliated to Bharathiar University)

Coimbatore-641006.

CERTIFIACTE

This is to certify that the project work entitled

C++ CODE CHECKER

Done by

Balu.N
9938M0601

Submitted in partial fulfillment of the requirements for the award of degree of
Master Of Computer Applications of Bharathiar University.

[illegible] 30/4/02
Professor and Head

[illegible] 30/4/2002
Internal Guide

Submitted for University Examination held on 09.05.2002.

[illegible] 9/5/02
Internal Examiner

[illegible]
External Examiner

ink

ETWORKS (P) Ltd.

April 29, 2002

The Principal
Kumaraguru College of Technology
Chinnavedampatti
Coimbatore.

Sir:

Sub: Project by Balu N - Certification

This is to certify that **Balu N (Reg. No: 9938M0601)** doing **MCA** in your college, has undertaken the semester project as a Student Project Trainee in our company and has completed the project, as per the details below :

Name of the Project	: C++ Code Checker
Duration	: 5 months
Period of Project Training	: December 2001 to April 2002
Project Report Submission Status	: Submitted

During the project training, the student has evinced keen interest in learning and his overall performance has been evaluated as Satisfactory as per the evaluation by our Student Project Management Committee.

We wish the student the very best in all his endeavors.

We thank you and look forward to a continued and fruitful relationship.

Yours truly,

[illegible]

M.Radhakrishnan
Head - HR

CC: Balu N

DECLARATION

I here by declare that the project entitled C++ CODE CHECKER submitted to Bharathiar University as a project work of Master Of Computer Applications degree is a record of original work done by me under the supervision and guidance of Mr. S.Chandrakumar, Mr. Senthilnathan of Think Business Networks Coimbatore and Mrs. S.Devaki B.E, M.S. Asst Professor Kumaraguru College of Technology, Coimbatore.

Place: COIMBATORE

[illegible]
Signature of Student

Date: 30.4.2002

[illegible] 30/4/2002

Internal Guide

Mrs. S.Devaki B.E, M.S,
Asst Professor,
Department of Computer Science & Engineering,
Kumaraguru College Of Technology,
Coimbatore.

ACKNOWLEDGEMENT

I take this opportunity to express my gratitude to all, whose contribution in this project work can never be forgotten.

I am extremely grateful to **Dr.K.K.Padmanabhan**, Principal, Kumaraguru College of Technology for having given me an opportunity to serve the purpose of my education.

I am indebted to Prof. **Dr. S.Thangasamy**, Head of the Department of Computer Science and Engineering, for his valuable guidance and useful suggestions during the course of the project.

I am deeply indebted to my project guide, **Mrs. S.Devaki, B.E., M.S.,** Assistant Professor of Department of Computer Science And Engineering, Kumaraguru College of Technology for her helpful guidance and valuable support given to me throughout the project.

With pleasure, I express my esteemed gratitude to **Mr.Arun Ulag,** President of Think Business Networks for providing me the opportunity to do the project in his reputed organization.

I am equally grateful to **Mr.Chandrakumar and Mr.Senthilnathan** for providing technical guidance and all others of the organization for their constant motivation during the course of the project.

Above all, I owe my gratitude to my friends **Mr. M.Deepak Charanyan B.E. and Mr. H.Ashwin Prabhu B.E.** parents and relatives for their support and to God Almighty, for showering abundant blessings on me.

SYNOPSIS

C++ Code checker is an application intended to assess the programmer's C++ source code and make recommendations, which aid in the improvement of program performance and efficiency.

C++ Code checker aims to provide a customizable interface, which allows the user to select fragments of code, which he/she wishes to improve. It contains a set of rules which act as a guideline to write C++ code.

It's a new system developed as an application, which checks the efficiency of the C++ code. C++ code Checker is an object-oriented system.

C++ code checker is developed at Think Business Networks Ltd, Coimbatore. Screen Design module includes Menu design, source code window, rule list window, debug window, workspace window and plug-in window.

Dialog has been created for open, save, close, clipboard operations, rule selection, rule processing against the code and display warning in debug window.

Rules are implemented as Win-32 DLL files and the modules are designed as easy-to-use functions.

C++ Code Checker is programmed in Visual C++. It uses Microsoft Foundation Classes (MFC) for implementing data structures.

Screen interfaces are designed using Win32 API (Application Programmers Interface).

TABLE OF CONTENTS

1.1 PROJECT OVERVIEW

C++ Code Checker is a software tool used to check the effectiveness of C++ code. It contains a set of rules. The C++ Code checker is a software application intended to assess the programmer's C++ source code and to make recommendations, which aid in the improvement of program performance and efficiency. The C++ Code checker aims to provide a customizable interface, which will allow the user to select fragments of code, which he/she wishes to improve upon.

This tool is useful for C++ programmers in Think Business Networks. This tool will help the users to add new rules in the form of Plug-ins. The programmers at Think Business Networks can use this tool to find errors, or use this as a guideline for quality and structured coding.

This software will attempt to identify portions of source code, which contain programming loopholes, inefficient usage of programming language features and will also put forward suggestions to effectively utilize memory resources.

This helps the programmer to reduce debugging time and minimize efforts to correct errors in the program. This software aims to fill a presently existing gap for a fully functional and user-friendly program to check the merit of C++ source code. The software also aims to satisfy the requirements of a wide array of users ranging from beginners to professional programmers.

The motive of the application is only to make recommendations and provide suggestions. The final implementation of these suggestions is at the discretion of the respective users.

The application assumes that all source code has already been compiled by industry standard compilers and so will concentrate only on suggestions, which enhance the quality of code.

RULES

Rules in C++ code checker are logical errors, runtime errors and efficiency conditions which are implemented as functions. The C++ source code is passed in to the rule function for checking against the conditions applied. These functions are stored as Dynamic Link Library files (DLL).

The advantage of using DLL files is that they get loaded into the application's address space, where data can be passed by reference to the functions.

RULES	**SUGGESTIONS**
Do not use C – style comments	C –style comments don't support nested comments.
Do not use C- style memory allocation	The malloc/free do not know about constructors and destructors.
Case without break	The control may get to next case block
No semicolon after while statement	The control evaluates the while condition without executing its body.
Do not support public members	The values assigned by class users may be illegal. It leads to production of runtime errors. No validations can be done for the values.
Memory leak, Resource leaks	Reduces the computer efficiency

Constraints

- C++ Code to be inspected should be free from compilation errors.
- This system is mainly used by professional C++ programmers
- All rules should be stored as DLL files.
- Both rule functions and DLL files should bear the same name. If they are not the same, C++ Code Checker won't execute the rule functions.

1.2 ORGANIZATION PROFILE

Think Business Networks Limited is a premier software company located in Coimbatore. This is a firm of dynamic young individuals, qualified in areas of software engineering, e-business and telecommunications. TBN was primarily established with a view to bridge the gap, between Cotton City (Coimbatore) and Silicon Valley (USA).

Four individuals led by Arun Ulag, an IRTT graduate in Coimbatore as a small firm, started TBN. From a humble beginning the firm has grown to one with over 100 employees having their own office at Ramanthapuram next to the city's biggest building The Stock Exchange Building.

TBN mainly concentrates on offshore projects. They get projects mainly from The United States, Italy and Sweden. They have a customer base of about 28 firms spread across Europe and Australia.

TBN has achieved CMM level 4 from SEI (Software Engineering Institute) of Carnegie Mellon University (USA) .The Company has branches at Chennai and New York.

2.1. EXISTING SYSTEM AND ITS LIMITATIONS

In the existing system, the programmer manually checks and identifies portions of source code, which contain programming loopholes.

Users have to walkthrough the code and find the errors.

C++ programmers require more time to debug the errors. The quality of code is not better in the existing system.

It is hard to find errors when user reads the code line by line. All the tests were done manually. The user can't have control over the code.

Author of code can easily find errors but quality assurance team may not find it.

2.2 PROPOSED SYSTEM

In the proposed system, the C++ code checker is very much flexible from the programmer's point of view. It is a new system developed as an application, which checks the efficiency and errors in the C++ code.

This helps the programmer in reducing debugging time and the effort to correct errors in the program. This software application aims to fill a presently existing gap for a fully functional and user-friendly program to check the merit of C++ source code.

The product function suite will include options and standard rules, which the user can select and check for. The software provides for an extendable interface wherein new rules can be installed as plug-ins.

The software also aims to satisfy the requirements of a wide array of users ranging from beginners to professional industry standard programmers.

The application assumes that all source code has already been compiled by industry standard compilers and so will concentrate only on suggestions, which enhance the quality of code.

2.3 REQUIREMENTS OF THE NEW SYSTEM

Scope

C++ Code checker is an application intended to assess the programmer's C++ code and make recommendations, which aid in the improvement of program performance and efficiency. C++ Code checker aims to provide a customizable interface, which allows the user to select fragments of code, which he/she wishes to improve.

The software attempts to identify portions of source code, which contain programming loopholes, inefficient usage of programming language features and also puts forward suggestions to effectively utilize memory resources.

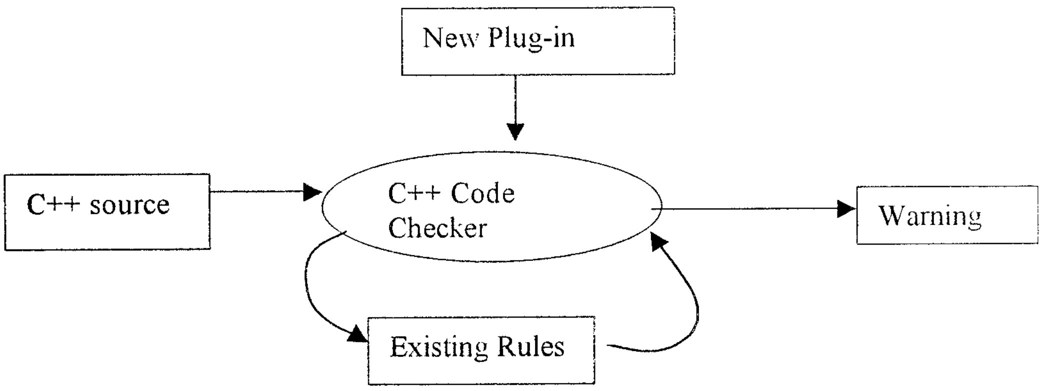
Definitions, Acronyms and Abbreviations

Rule	A logical error and run time error in the program.
Plug in	New rules can be added to the application.
Inspection	The C++ code is inspected against the rule.
DLL	Dynamic Link Libraries, which contain the rules.
C++CC	C++ Code checker

Overall Description

a) Product Perspective

It's a new system of its kind developed as an application, which checks the efficiency of the C++ code. The proposed system is an object-oriented system.



b) Product Functions

This helps the programmer in reducing debugging time and effort to correct errors on the program.

This application aims to fill a presently existing gap for a fully functional and user-friendly program to check the merit of C++ source code.

The product function suite will include options and standard rules, which the user can select and check for. The software will provide for an extendable interface wherein new rules can be installed as plug-ins.

The application also aims to satisfy the requirements of a wide array of users ranging from beginners to professional programmers.

The motive of the application is only to make recommendations and provide suggestions. The final implementation of these suggestions is at the discretion of the respective users.

The application assumes that all source code has already been compiled by industry standard compilers and will concentrate only on suggestions, which enhance the quality of code.

Specific Requirements

Functionality

a) Introduction

When the user runs the application, he is given a set of rules to choose from. After selecting the set of rules he has to inspect the given C++ code. A description of the rules is as follows:

- Provides the user with a suite of rules to check for. The rules are conveniently grouped under functional groupings.
- This software provides the user with the flexibility of choosing only a selected set of rules by means of an interactive interface.
- The software is designed to be a stand-alone application into which rules in the form of plug-ins can be fitted. This provides the user with a high degree of scalability and flexibility.

The application checks for rules that fall under the following subsets:

- Migrating from C to C++
- Objects and classes
- Pointers
- Declarations
- Memory utilization
- Processor utilization

b) List of Inputs

1. The product can check the efficiency of code (Inputs) C++ code that may be as

    i. C++ source files
    ii. C++ header files
    iii. C++ classes
    iv. C++ Functions
    v. C++ Blocks

2. Rules

    i. Existing rules are selected .
    ii. New rule is added in the form of DLL.

c) Information Processing required

The following processing is required at the application level at various points during the life of the product

i. The C++ code should be pre-complied against the syntax error.
ii. User is not allowed to inspect non-C++ code.
iii. No duplication of rules.

Performance Requirements

Security

a. Newly added rules should be having less space and time complexity.
b. User can add only DLL files as plug-ins.
c. Never affect the Operating systems functions.

Availability

It's a probability measure that a program is operating according to requirements. System is available for all professional C++ programmers.

Capacity

Application can inspect any number of lines of code in the program. It can check any numbers of function/classes. Application can inspect a number of programs at a time.

Response Time

The response time for the application depends on the processor speed, amount of memory and efficiency of rule programs in DLL.

Usability

- User is provided with a comprehensive tutorial and help package that will aid the user in getting accustomed to the application.
- The GUI has been designed keeping in mind the user requirements of simplicity and functionality.

User Interface, Screen formats

Win32 API is used for creating user interfaces (editor window, menus).

It includes a rich text editor used for displaying C++ source file and ActiveX controls to display rules. Each rule is displayed as a check box, list box etc to display error after inspecting code resources used for menu displays.

Other Requirements

Operations required by user

Rule selection

User can inspect the default rules, or he can inspect specific categories (Memory).He can also make specific rules as default ones.

Add plug-ins

New plug-ins added if user/programmer has written own rule.

Inspection

User can inspect the whole program and can also inspect a class in C++ files and function.

2.4 USER CHARACTERISTICS

The system has been designed for use with ease. The whole system is partitioned into two parts: That used by programmers and that used by quality assurance teams.

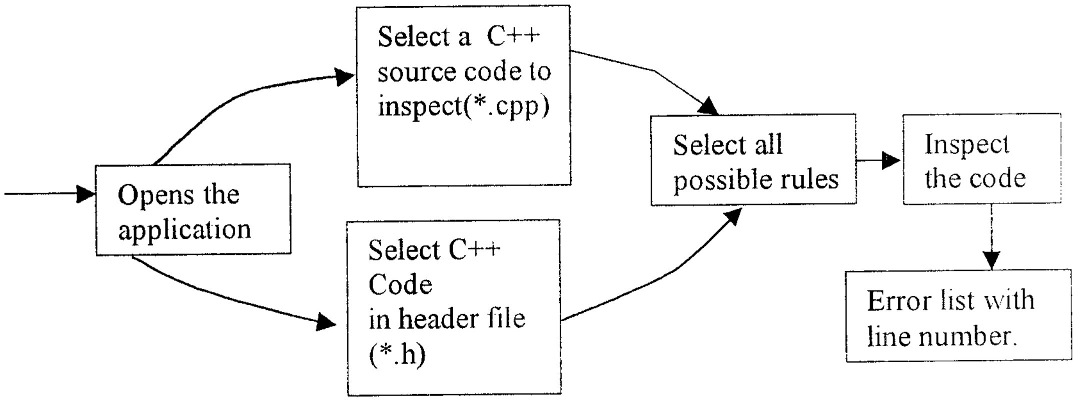
Programmers

Helps the programmer to check C++/header files for given set of rules. The programmer is the user of the system.

Quality Assurance team

This team plays a very important role in software quality. They check for various quality factors.

User View


Opens the application
Select a C++ source code to inspect(*.cpp)
Select C++ Code in header file (*.h)
Select all possible rules
Inspect the code
Error list with line number.

3.1 HARDWARE CONFIGURATION

Processor	:	Pentium Celeron
Clock Speed	:	700 MHz
Main Memory	:	128MB
External Cache	:	128KB
Hard Disk	:	10 GB
Floppy Disk Drive	:	1.44MB
CD-ROM	:	32x Creative
Display	:	SVGA Samtron Color monitor
Mouse	:	Logitech 3-button
Keyboard	:	105 keyboards

3.2 SOFTWARE AND PACKAGES USED

Platform	:	Windows 98.
Programming Language	:	C++.
Compiler	:	Microsoft Visual C++ 6.0.
GUI Designer	:	Win32 Application development (VC++).
Rule Developer	:	Win32 Dynamic Link Library, MFC.
Intermediate Files	:	INI files.
Help Files	:	HTML.

Platform Windows 98

This system is developed on Microsoft Windows 98 Operating system.

Reason for Choosing Windows-98:

1. It provides right user interface.
2. It is most widely used by companies
3. It reduces the training required for users.
4. It supports COM and COM+ standards for reusing binary codes

Programming Language C++

The source code and rules are written in C++. It's an object-oriented language. So implementation of design is easy.

Reason for Choosing C++:

1. It is an object oriented language, supporting concepts such as classes, polymorphism, inheritance, virtual functions, interfaces.

2. C++, like C, is a language that is heavily reliant on a rich set of Library functions to provide the following:

   a. Portable operating-system interface (file and screen I/O)

   b. String and buffer manipulation

   c. Floating-point math transformations

   d. Supports Cross Compiling.

Compiler: Microsoft Visual C++ (cl.exe)

The source codes written in C++ classes are compiled using Microsoft VC++.

The reasons for choosing Visual C++ are the following:

1. It's a product of Microsoft Corporation.
2. Extensions to C++ Code Checker to system can be done easily.
3. It supports many features like COM and COM+.
4. Low-level /hardware programming can be done easily.
5. It supports many predefined libraries.
6. Exception handling is done accurately.

Choose INI Files to store intermediate results

1. Text file operations are hard when compared with INI files.
2. INI file supports sections and tags and so we need not care about white spaces
3. Getprivateprofilestring and Setprivateprofilestring are APIs for accessing INI files.

UML: Unified Modeling Language

The Unified Modeling Language is a powerful new tool for developers to use when working on object-oriented systems. Because its purpose is to document and model a software system using a language-independent methodology, software designers can easily communicate their designs to other designers and to those who ultimately implement the software.

4.1 INPUT DESIGN

Input design is a part of overall system that requires careful attention. most. The input should be a compiled free of errors. The system is assumed that it gets a code that doesn't have Compile-Time error. So if user supplies a code that contains a compiler error, system will not produce the required result. Objectives during input design are as follows:

a. Achieve high level accuracy
b. Ensure input is free of ambiguity

The input design involves converting the user-originated inputs into a computer-based format. The aim of input design is to make data entry easy and logical error free. It helps us to filter errors in the input data that might have brought in a lot of inconsistency.

It involves procedures for capturing data, verifying and then passing them onto the system. After choosing input medium, attention is focused on designing of error handling, control, and grouping the validation procedures.

During application development, care has been taken to make our system extremely user-friendly and organize our screens such that the possibilities of making error are reduced.

List of possible inputs are C and C++ files which are provided to the user for selecting the inputs. This makes system less error prone .

Warnings are displayed on the screen for wrong entries such as non C++ files, error prone rules, duplicate rules, missing of INI and (rule) DLL files.

The inputs to the system are

1. C++ source code (*.cpp).
2. C++ header code (*.h).
3. New rules as plug-ins (*.dll).
4. Selecting existing rules by using the check box.

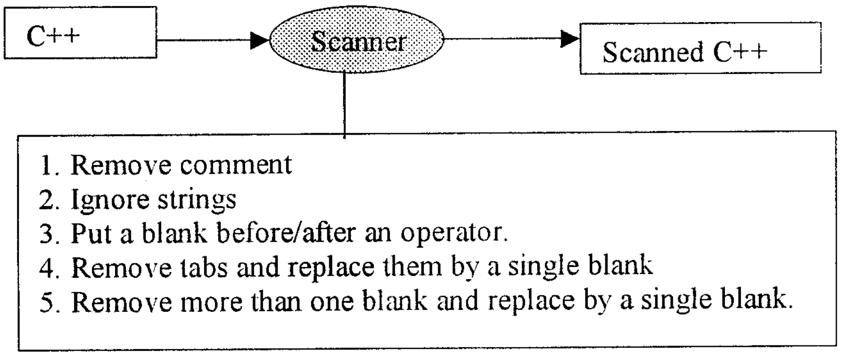

C++
Scanner
Scanned C++
1. Remove comment
2. Ignore strings
3. Put a blank before/after an operator.
4. Remove tabs and replace them by a single blank
5. Remove more than one blank and replace by a single blank.

Figure 1: C++ Code Checker scanner

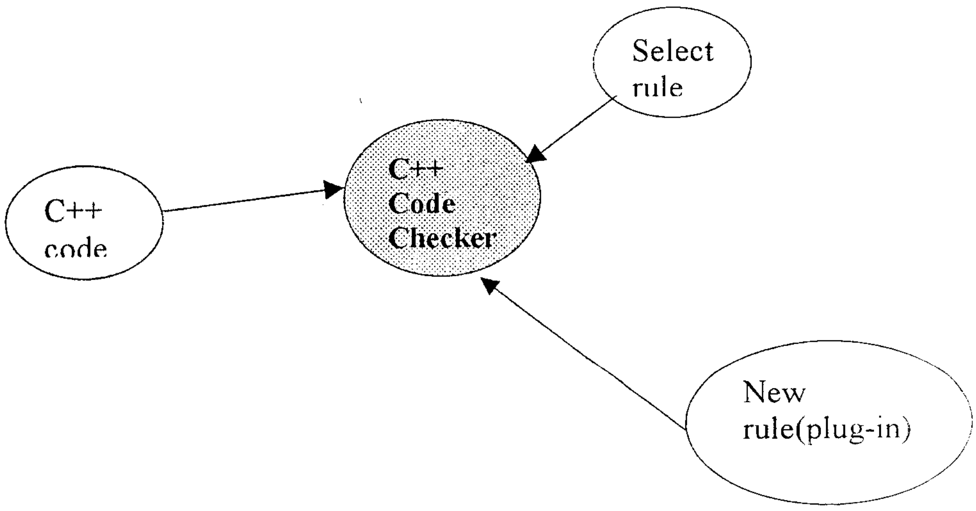


Figure 2: Input View of C++ Code Checker

INPUT TYPES

One of the early activities of input design is to determine the nature of input data. The different types of inputs handled by our system are:

External	: Prime input to system
Internal	: Communication with system
Operational	: Programming team communicating with system.
Computerized	: Input to the computer media from other internal sources.
Interactive	: Input entered during a dialog.

Menus:

The menus in the application are as follows

File menu

Edit menu

View menu

Debug menu

Plug-in menu

Help

File Menu and its operations

It contains standard file related operations, for manipulating C++ code present in source code window.

Edit Menu and its Operations

It contains options for performing clipboard related operations (cut, copy, paste), find and replace options, selecting whole file, selecting a function and selecting a class.

View Menu and its operations

It contains options for viewing

1. Rule window- Contains rules where user can make selection
2. Workspaces Tree like structure describing C++ files in tree view
3. Debug window Displays warning messages in a list box.

Debug Menu and its Operations

Check submenu is used to check the source code against the rules selected by users. Also supports user to pass the following to the application:

1 C++ classes

2 C++ functions

3 C++ blocks

Add Plug-in Menu and its operations

User can add plug-in to system using this menu.

Help Menu

It contains two submenus

1. About - Describes information about developers of C++ Code checker
2. Online Help - It provides link to help file which is in Windows standard Help format

4.2 OUTPUT DESIGN

An inevitable activity in the system is the proper design of input and output in a form acceptable to the user. Outputs from the system are required primarily to communicate the result of processing to user.

An output also provides a permanent copy of the results for later consultation. An intelligible output design will improve system relationships with the user and help in the decision-making process.

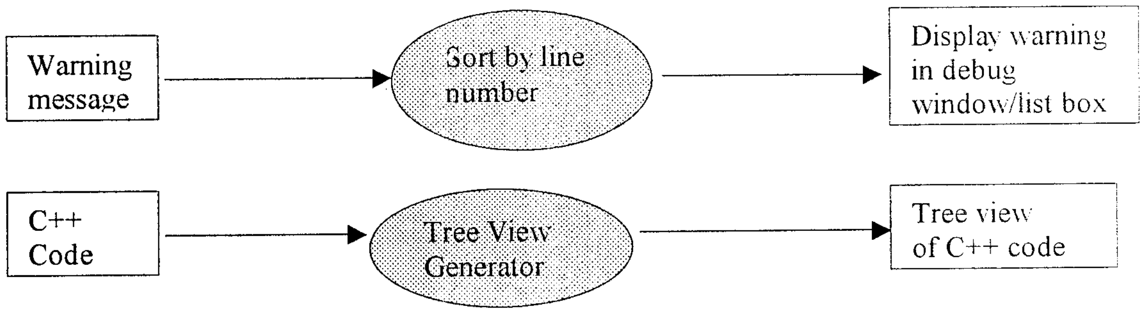


Figure 3: Output View of C++ Code Checker

The various types of outputs required by most systems are:

External Outputs: Whose destination is outside the organization which require special attention

Internal Outputs: Whose destination is within the organization and which require careful design because they are user's main interface with computer

Operational Outputs: Whose use is purely within the computer department.

Interactive Outputs: Which involves the user in communicating with the computer. The approach to output design is very dependent on the type of output and nature of data. Special attention has to be made to data editing. The choice of appropriate output medium is also an important task.

The selection may be affected by the following kinds of considerations.

1. Response time.
2. Experience of user with C++.
3. Cost.
4. Software / hardware.
5. Suitability of the device for application concerned.

The output design must be specified and documented. Data items have to be defined and arranged for clarity and easy comprehension. The other two objectives that were taken care were:

I. The interpretation of the results of the computer part of the system to users in a form that they can understand and also meets the requirements.

II. The output design specification is made in such a way that it is unambiguous.

III. Comprehensive and capable of being translated into a programming language.

Screens, which are major form of output, are designed in various modules in the system. Screen generated are:

1. Debug window
2. Tree view of C++ files

Debug window

It displays warning messages for C++ code against the given set of rules. It contains warning message, line number, error number and class name. These messages are displayed in list box where user can click code and move to corresponding line.

Tree view of C++ files

It represents C++ files in a hierarchical structure. Root node will be class's list. The class name is in parent node and the child nodes are represented by function name with its prototype. Any level of nesting of classes should be displayed.

4.3 PROCESS DESIGN

A computer procedure is a series of operations designed to manipulate data to produce output from a computer system. A procedure may be a single program or a set of programs.

Various types of procedures are

1. Validations
2. Sorting
3. Process files
4. Process database
5. Printing

Validations

Data enters the computer-based system via validation procedures. Often they are catered for by a generalized input validation package tailored to the need of the particular system.

Error handling

Error procedures must be specified in detail showing decisions, actions and exceptions.

There are various ways of error handling such as

- Rejecting the items of input and processing the next item
- Writing an error record onto an error file.
- Signaling operator by messages on type writer
- Not updating or processing the file or batch in which error occurred
- Dumping the content to the file where the error was detected
- Going back to an earlier stage in processing and starting again from that point
- Halting the program execution.

- Allowing system to correct the error internally
- Manual procedure for correcting the error.

Sorting

Data frequently has to be sorted before a further phase of processing can be carried out. The time spent sorting, which can represent 40% of total processing time, is really non-productive and must be reduced as much as possible.

Sorting in general sense means arranging data according to a particular rule or pattern. Sorting is classified as internal sorting and external sorting.

The process design is the heart of the system design. The system specification is designed here. In C++ Code Checker, rule processing is the main activity. A set of rules is added to the system in the form of DLL files.

Following activities will be helpful in the process design

1. Add Rule to system as plug-in (Stored in DLL files in /plugins directory).
2. The new rules added have the following validations.
   i. Rule DLLs should be in /plugins directory.
   ii. Name of DLL and rule function should be same.
   iii. Rule should not be added to the system.
3. Rule name is added with its description in rule.ini.
4. Rule list window is activated for selecting rules.
5. The selected rule status is stored in an array/list.
6. Processing of C++ code comprise of the following steps
   i. Press run button in tool bar.
   ii. In rule.ini get file name and from list/array get the status.

```
iii. If status = ok then
        a. Load DLL file into address space using Load
           Library API
        b. Get the address of rule function and store in
           pointer to function
        c. Pass the selected data in rule window to rule
           function.
        d. If rule is not accepted then an error warning is sent
           to sendmsg.ini
7. Repeat step 6 until all rules are processed.
8. If all rules are processed then get warning from the sendmsg.ini
   and display in list box/debug window.
```

Procedures/processes in C++ Code Checker

1. Validations
2. Reading /writing of INI files (intermediate)
3. Checking correct rules/plugins
4. Sorting warning.ini by line number.

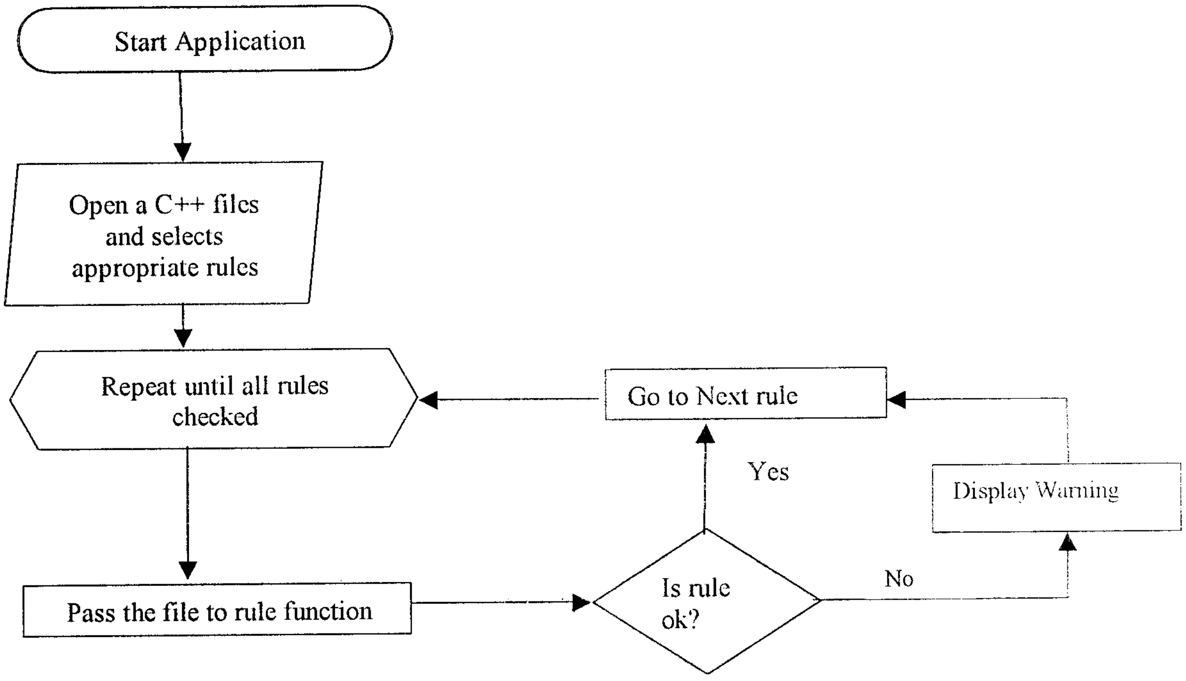

Start Application
Open a C++ files and selects appropriate rules
Repeat until all rules checked
Pass the file to rule function
Is rule ok?
Yes
No
Go to Next rule
Display Warning

Figure 4: Process view of C++ Code Checker

5.1 SYSTEM IMPLEMENTATION

A crucial phase in the system life cycle is the successful implementation of the new candidate system. Implementation simply means converting a new system design into operation. This involves checking computer compatibility, training the operating staff and installing the necessary hardware and terminals before the system is up and running.

Implementation is the stage of project when the theoretical design is turned into a working system. It involves careful planning, investigation of current system and its constraints on implementation, design of methods to achieve the changeover, training of staff in the changeover procedures and evaluation of changeover methods.

Therefore, implementation is the process of converting a new or a revised system design into an operational one. Thus during this stage the theoretical design is turned into a working system. If the implementation stage is not carefully planned and controlled, it can cause chaos. Generally, there are three types of implementation.

They are,

- Implementation of a computer system to replace a manual system.
- Implementation of a new computer system to replace the existing one.
- Implementation of a modified application to replace the existing one uses the same existing system.

5.2 SYSTEM TESTING

Quality Assurance

The amount and complexity of software stagger the imagination. Consequently, some controls must be developed to ensure a quality product. Basically, Quality assurance defines the objectives of the project and reviews the overall activities so that errors are corrected early in the development process.

Thus, the quality assurance defines the factors that contribute to the quality of the candidate system. The following factors were studied for determining the major qualities of the system.

- Correctness of the system.
- System reliability.
- Efficiency of computer resources.
- Accuracy in input, computations & output.
- Error tolerance.
- Communication – How descriptive they are
- The inputs and outputs of the system.

Testing is a process of executing a program with the intention of finding errors. A good test case is one that has a high probability of finding as yet undiscovered errors.

A successful test is one that uncovers as yet undiscovered errors. System testing is the stage of implementation, which is aimed at ensuring that the system works efficiently and accurately before live operation commences.

System testing requires a test plan that consists of several key activities and steps for program, system and user acceptance testing.

C++ Code Checker is tested with the following criteria in mind

- The program execution is correct or not.
- To select more then one rule at a time.
- Usability documentation and procedure for the user-friendly nature of the system.
- Recovery and security.

Software testing is an important phase in the development of the software. Testing is done for software quality assurance. Software is developed for a long use in the future and so it is necessary that a well planned testing be done to overcome errors. Static Analysis is used to investigate the structural properties of the source code. Dynamic test cases are used to investigate the behavior of source code by executing the program on the test data. As before, we use the term "Program Unit" to denote a routine or a collection of routines or a routine implemented by an individual programmer. In a well-designed system, a program unit is stand-alone or a function of a large system.

UNIT TESTING:

Unit testing comprises the set of tests performed by an individual programmer prior to integration of the unit into a larger system. The situation is illustrated as follows,

Coding debugging ------ Unit testing --- Integration testing

A program unit is usually small so that the programmer who developed it can test it in great detail. There are four categories of tests a programmer will typically perform on a program unit.

- *Functional test*
- *Performance test*
- *Stress test*
- *Structure test*

Functional test cases involve exercising the code with nominal input values for which the expected results are known, as well as boundary values like minimum values, and values on the functional boundaries and special values, such as logically related inputs, the identity matrix, files of identical elements and empty files.

Performance test determines the amount of execution time spent in various parts of the unit, program throughput and response time and device utilization by the program unit. A certain amount of performance tuning may be done during unit testing; however, caution must be exercised to avoid expending too much of fine tuning of a program unit that contributes little to the overall performance of the entire system. Performance testing is most productive at the subsystem and system levels.

Stress tests are those tests designed to intentionally break the unit. A great deal can be learned about the strengths and limitations of the program by examining the manner in which a program unit breaks.

Structure tests are concerned with exercising the internal logic of the program and traversing particular execution paths. The main activities in structural testing are deciding which paths to exercise, deriving test data to exercise those paths, determining the test coverage criterion to be used, executing the test cases and measuring the test coverage achieved when the test cases are exercised.

A test coverage criterion must be established for unit testing, because program units usually contain too many paths to permit exhaustive testing. Even if it were possible to successfully test all paths through a program, correctness would not be guaranteed by path testing because the program might have missing paths and computational errors that were not discovered by the particular test cases chosen. A missing path error occurs when the branching statement and the associated computations are accidentally omitted. Missing path errors can only be detected by functional test cases from the requirement specifications. Thus, tests based solely on the program structure cannot detect potential errors in a source program.

Unit testing focuses on verification. In the smallest unit of software design the module unit testing is done for each module to ensure that it functions properly as a unit. In unit testing, the module interface is tested to ensure that information properly flows into and out of the program under test.

Unit testing is done to recover errors of the following types.

- *Erroneous initialization*
- *Incorrect variable names*
- *Inconsistent data type*
- *Underflow, overflow and addressing exceptions*
- *Computation errors.*

SYSTEM TESTING:

System testing is a series of different tests whose purpose is to exercise the computer-based system fully. There are two kinds of activities viz. integration testing and acceptance testing. Strategies for integrating software components into a functional product include the bottom up strategy, the top down strategy and the sandwich strategy.

Careful planning and scheduling are required to ensure that the modules will be available for integration into the evolving software product when needed. Acceptance testing involves planning and execution of functional tests, performance tests and stress tests to verify that the implemented system satisfies its requirements. Acceptance tests are typically performed by the quality assurance and/or customer organizations. Depending on local circumstances, the development group may or may not be involved in the acceptance testing.

INTEGRATION TESTING:

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to recover errors associated with interface. Bottom up integration is the traditional strategy to integrate the components of the software system into a functional whole. Bottom up integration consists of unit testing, followed by subsystem testing, and testing of the entire system. Unit testing has the goal of discovering errors in the individual modules of the system. Modules are tested in isolation from one another in an artificial environment known as "Test Harness" which consists of the driver programs and data necessary to exercise the modules.

Unit testing should be as exhaustive as possible to ensure that each representative case handled by a module has been tested. Unit testing is eased by a system structure that is composed of small, loosely coupled modules.

A subsystem consists of several modules that communicate with each other through well-defined interfaces. Normally, a subsystem implements a major segment of the total system. The primary purpose of subsystem testing is to verify operation of the interfaces between modules in the subsystem. Both control and data interfaces must be tested. Large software systems may require several levels of subsystem testing: Lower-level subsystems are successively combined to form higher-level subsystems. In most software systems, exhaustive testing of subsystem capabilities is not feasible due to the complexity in the combination of the module interfaces; therefore test cases must be carefully chosen to exercise the interfaces in the desired manner.

System testing is concerned with subtleties in the interfaces, decision logic, control flow, recovery procedures and throughput, capacity and timing characteristics of the entire system. Careful test planning is required to determine the extent and nature of the system testing to be performed and to establish criteria by which the results will be evaluated.

Disadvantages of bottom up testing include the necessity to write and debug test harnesses for the modules and subsystems, and the level of complexity those results got by combining modules and subsystems into larger units. All the modules are then linked and executed in a single integration run. This is the “Big Bang” approach to integration testing. The main problem with big-bang integration is the difficulty of isolating the sources of errors.

Top down integration testing starts with the main routine and one or two immediately subordinate routines in the system structure. After this top- level "Skeleton" has been thoroughly tested, it becomes the test harness for its immediately subordinate routines. Top down integration requires the use of program stubs to simulate the effort of lower level routines that are called by those being tested. Disadvantages of top down integration are,

- *System integration is distributed throughout the implementation phase. Modules are integrated as they are developed.*
- *Top-level interfaces are tested first and most often.*
- *The top-level routines provide a natural test harness for lower-level routines.*
- *Errors are localized to the new modules and interfaces that are being added.*

While it may appear that top-down integration is always preferable, in many situations it is not possible to adhere to a strict top-down coding and integration strategy.

For example it may be difficult to find top-level input data that will exercise a lower level module in a particular desired manner.

Sandwich integration is predominantly top-down, but bottom-up techniques are used on some modules and subsystems. This mix alleviates many of the problems encountered in pure top-down testing and retains the advantage of top-down integration at the subsystem and system level.

ACCEPTANCE TESTING

Acceptance testing involves planning and execution of functional tests, performance tests and stress tests in order to demonstrate that the implemented system satisfies its requirements. It is not unusual for two sets of acceptance tests to be run: those developed by quality assurance group and those developed by the customer.

In additional to functional and performance tests, stress tests are performed to determine the limitations of the system. Typically, acceptance tests will incorporate test cases developed during unit testing and integration testing. Additional test cases are added to achieve the desired level of functional, performance and stress testing of the entire system.

6. CONCLUSION

C++ Code Checker provides the user with a suite of rules to check for. The rules are conveniently grouped under functional groupings. This software provides the user with the flexibility of choosing only a selected set of rules by means of an interactive GUI.

Each rule is implemented as separate function; the content of file is passed as parameter to the system. The software is designed to be a stand-alone application into which rules in the form of plug-ins can be fitted.

If error or warning is present in the code, message is stored in the intermediate file along with line number and sorting based on line number occurs. C++ Code to be inspected should be compiler error free. This system is mainly used by C++ programmers.

7. SCOPE FOR FUTURE DEVELOPMENT

Suggestions are made here for further developments. The language has adequate scope for further development in the future.

At the moment this project handles C++ codes. But in future it can be extended to all languages like Java, Visual Basic, etc...

The code checker only gets input as compiler error free format. But in future enhancements, you can add C++ compiler in this project to check for syntax errors.

You can extend the project to all platforms like UNIX, LINUX, etc., Now only the limited rules are defined, you can add more rules in future using QT+ software and WINE Interfaces.

The rule functions are in the DLL files. In next version of the system it will be converted to ActiveX-DLL (In-process servers).

REFERENCES

Books:

Scott Mayer	-	Effective C++ Addison Wesley Publications.
Scott Mayer	-	More Effective C++ Addison Wesley Publications.
Deitel & Deitel	-	C++ How to Program Pearson Education Asia.
Bjarne Stroustrup	-	The C++ Programming Language
Herbert Schlitz	-	"Windows-98 Programming" Tata McGraw-Hill

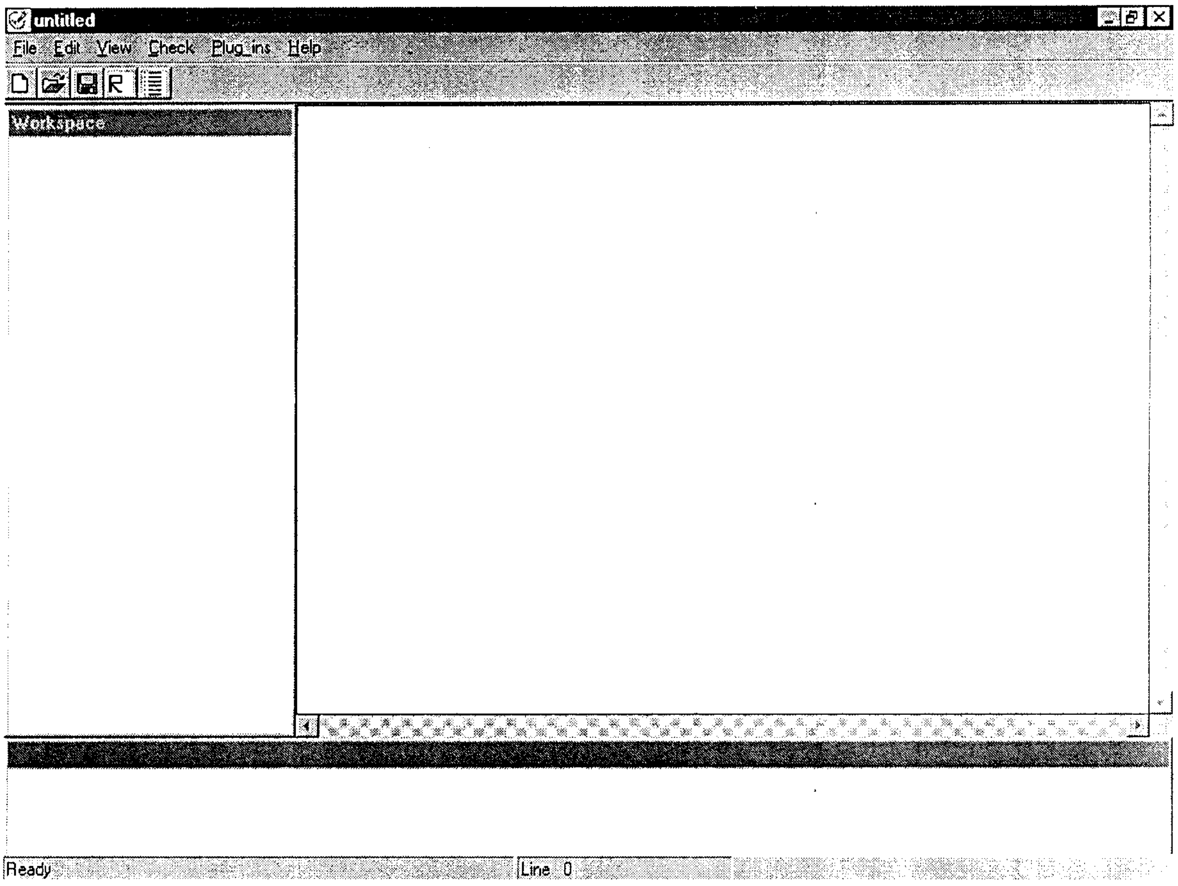
Websites:

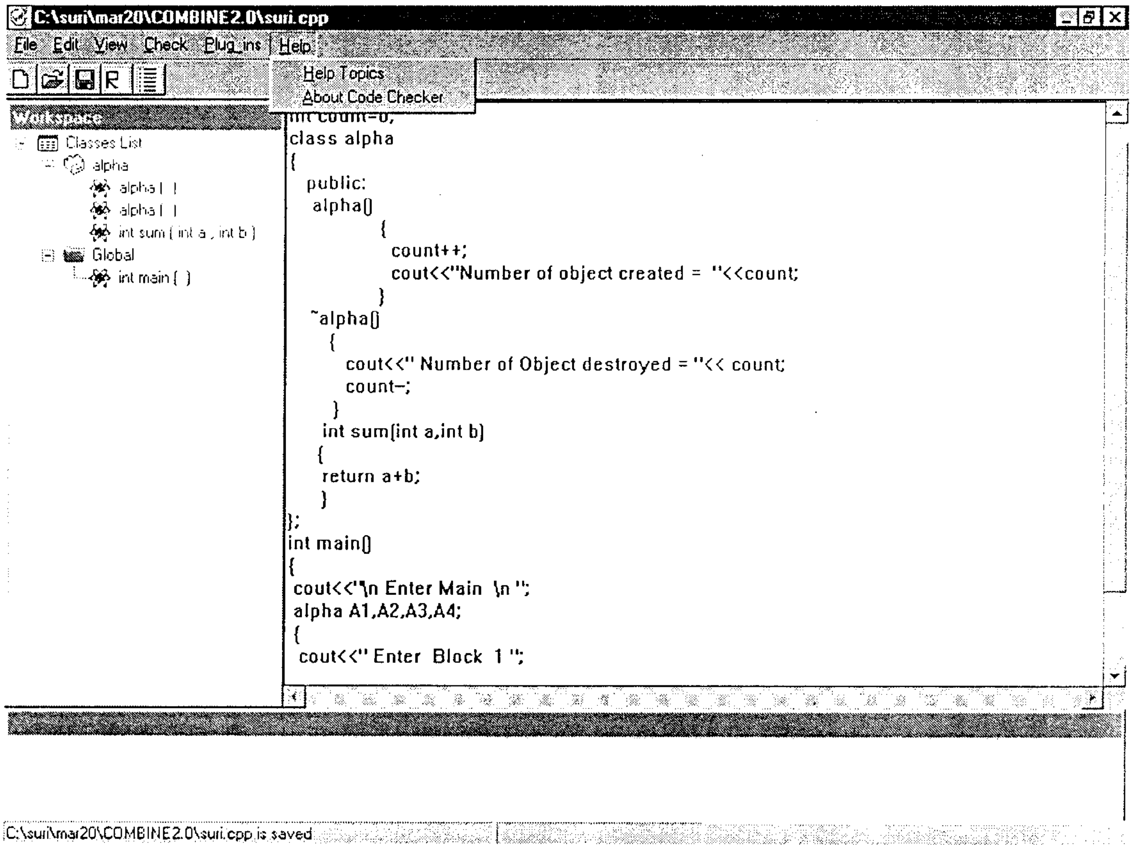
www.yokasoft.com
www.gimple.com

Additional References

1) Chandra Kumar Team Leader (Mentor)
2) Senthil Nathan Software Engineer (Mentor)

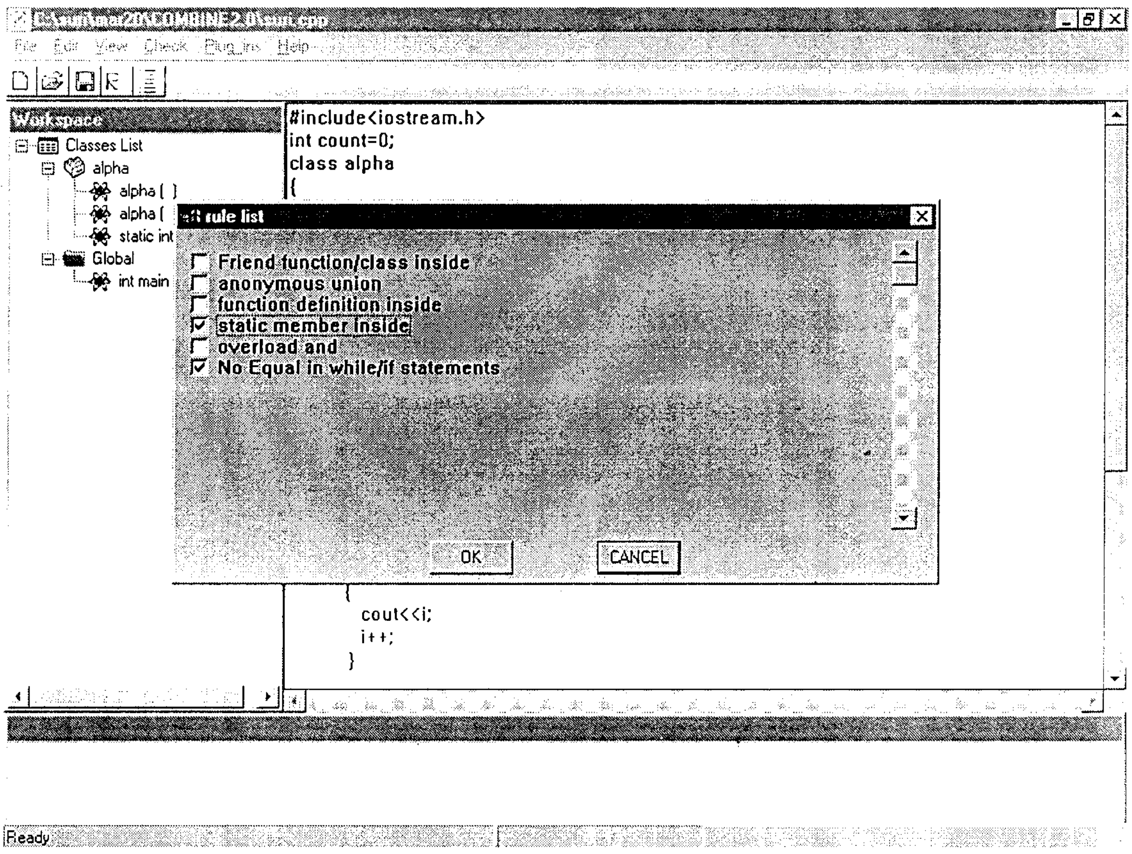
Think Business Network Ltd, Coimbatore


untitled
File Edit View Check Plug ins Help
Workspace
Ready
Line 0



```
int count=0;
class alpha
{
   public:
    alpha()
              {
                count++;
                cout<<"Number of object created =  "<<count;
              }
   ~alpha()
      {
         cout<<" Number of Object destroyed = "<< count;
         count--;
       }
     int sum(int a,int b)
     {
      return a+b;
      }
};
int main()
{
 cout<<"\n Enter Main  \n ";
 alpha A1,A2,A3,A4;
 {
  cout<<" Enter  Block  1 ";
```

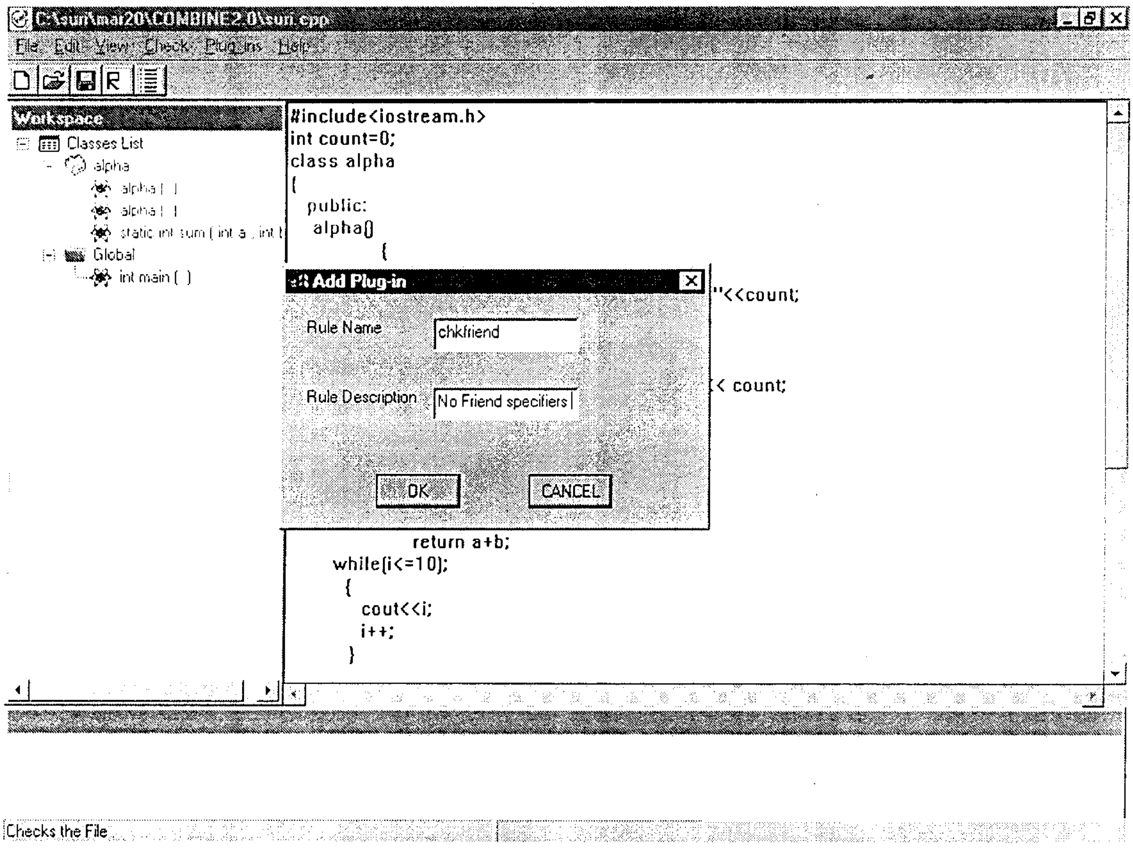
C:\suri\mar20\COMBINE2.0\suri.cpp is saved



```
#include<iostream.h>
int count=0;
class alpha
{
```

```
    {
      cout<<i;
      i++;
    }
```

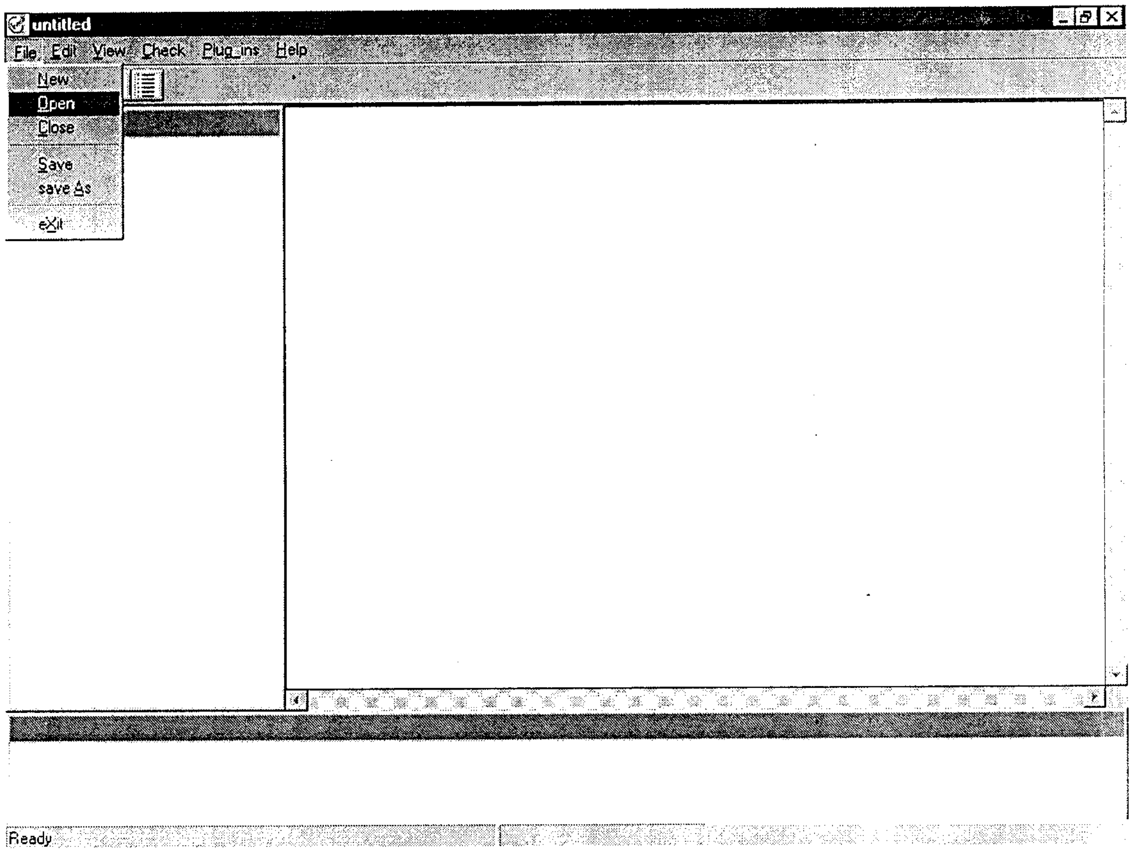
- [ ] Friend function/class inside
- [ ] anonymous union
- [ ] function definition inside
- [✓] static member inside
- [ ] overload and
- [✓] No Equal in while/if statements

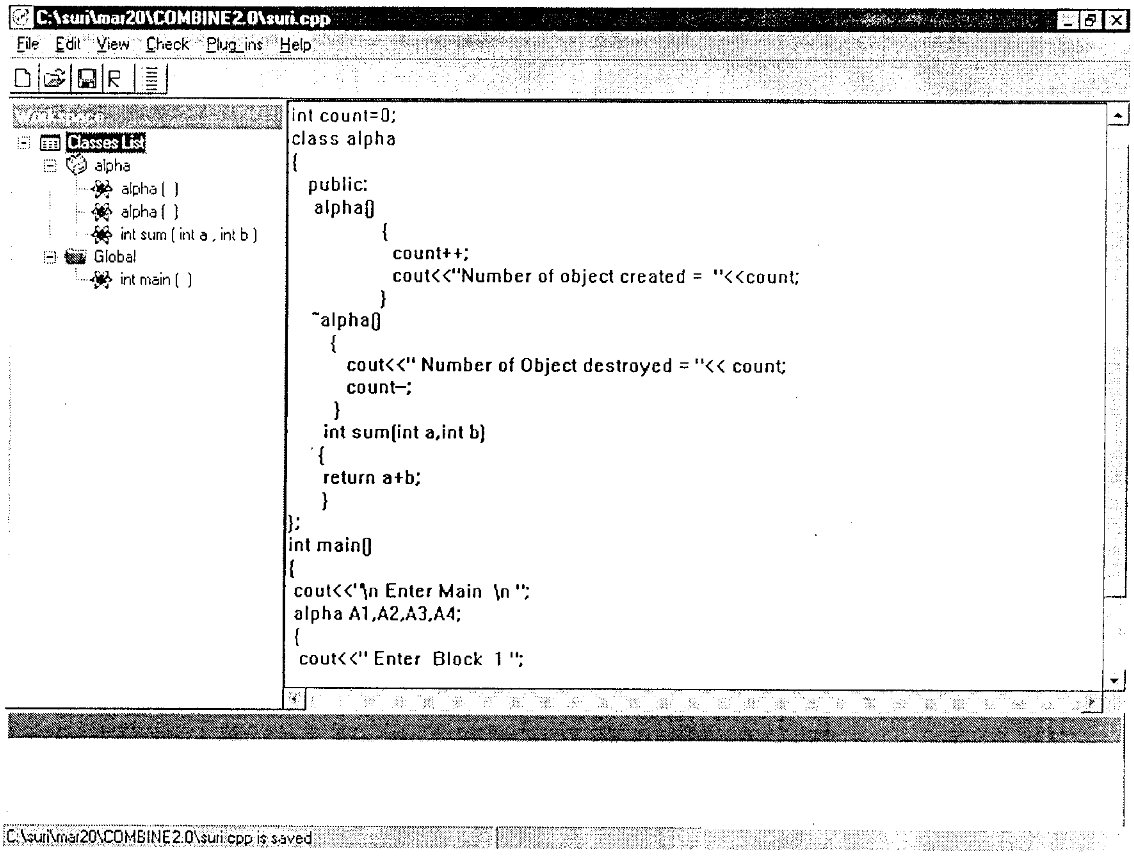

C:\suri\mar20\COMBINE2.0\suri.cpp
File Edit View Check Plug-ins Help
Workspace
Classes List
alpha
alpha ( )
alpha ( )
static int sum (int a , int b
Global
int main ( )
Add Plug-in
OK
CANCEL
Checks the File

Rule Name: chkfriend

Rule Description: No Friend specifiers

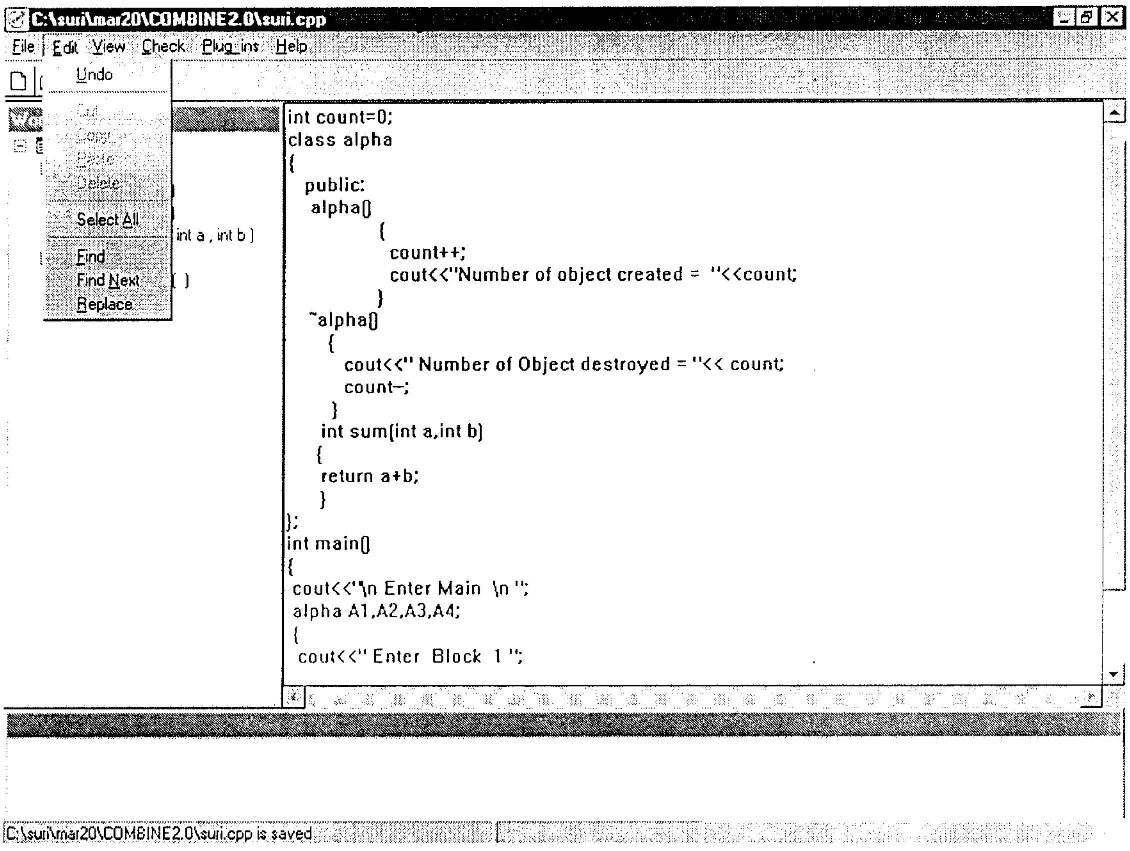
```
#include<iostream.h>
int count=0;
class alpha
{
  public:
   alpha()
         {
                                                "<<count;

                                                < count;

                return a+b;
       while(i<=10);
        {
          cout<<i;
          i++;
        }
```


untitled
File
Edit
View
Check
Plug_ins
Help
New
Open
Close
Save
save As
eXit
Ready


C:\suri\mar20\COMBINE2.0\suri.cpp
File Edit View Check Plug ins Help
Workspace
Classes List
alpha
alpha ( )
alpha ( )
int sum (int a , int b)
Global
int main ( )

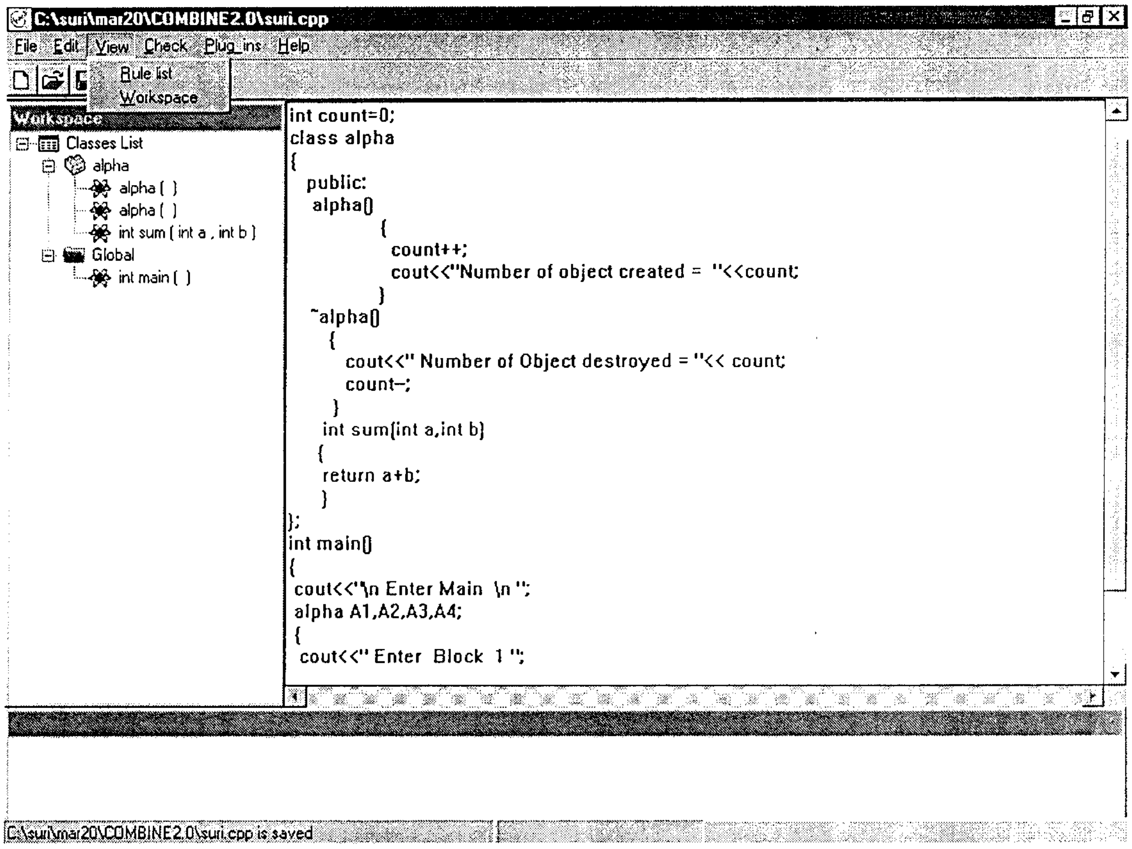
```
int count=0;
class alpha
{
  public:
   alpha()
            {
              count++;
              cout<<"Number of object created =  "<<count;
             }
   ~alpha()
       {
         cout<<" Number of Object destroyed = "<< count;
         count--;
       }
     int sum(int a,int b)
     {
     return a+b;
     }
};
int main()
{
 cout<<"\n Enter Main  \n ";
 alpha A1,A2,A3,A4;
 {
  cout<<" Enter  Block  1 ";
```

C:\suri\mar20\COMBINE2.0\suri.cpp is saved



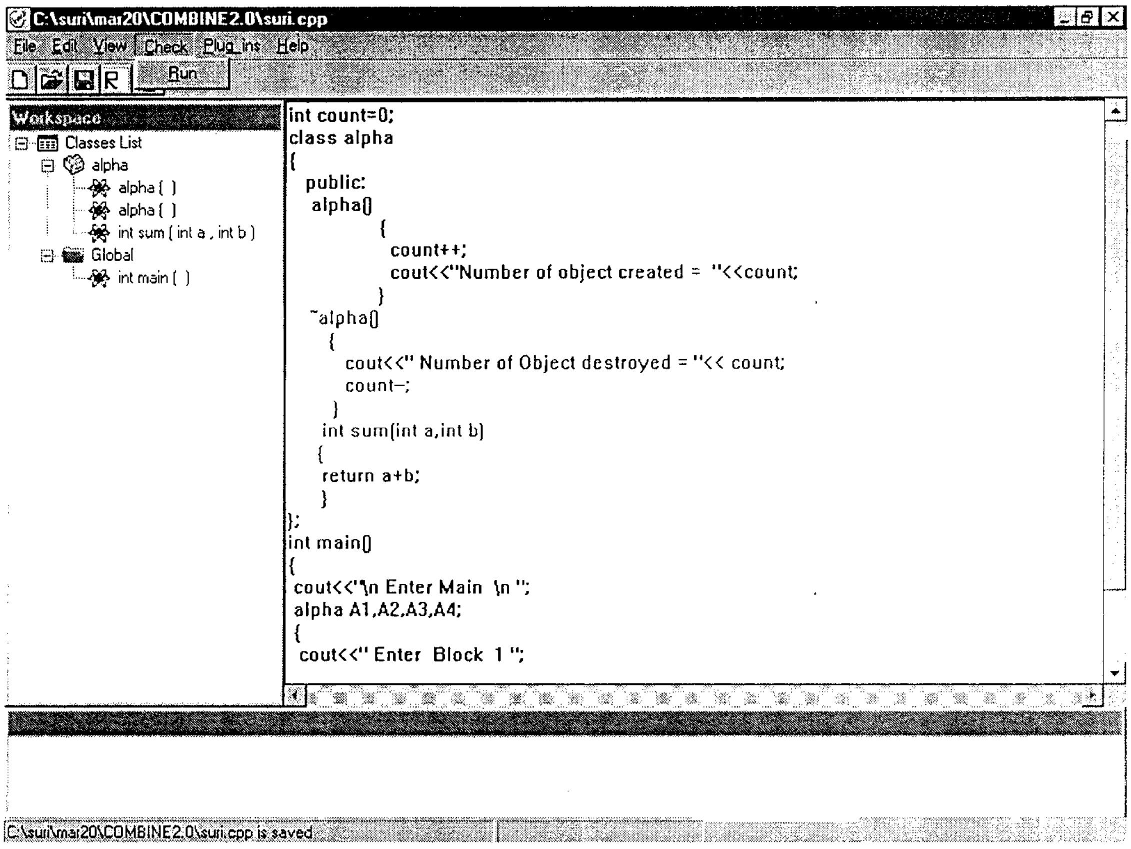
```
int count=0;
class alpha
{
   public:
    alpha()
             {
               count++;
               cout<<"Number of object created =  "<<count;
             }
   ~alpha()
      {
        cout<<" Number of Object destroyed = "<< count;
        count--;
      }
     int sum(int a,int b)
    {
     return a+b;
     }
};
int main()
{
 cout<<"\n Enter Main  \n ";
 alpha A1,A2,A3,A4;
 {
  cout<<" Enter  Block  1 ";
```

C:\suri\mar20\COMBINE2.0\suri.cpp is saved


C:\suri\mar20\COMBINE2.0\suri.cpp
File Edit View Check Plug_ins Help
Rule list
Workspace
Workspace
Classes List
alpha
alpha ( )
alpha ( )
int sum (int a , int b)
Global
int main ( )

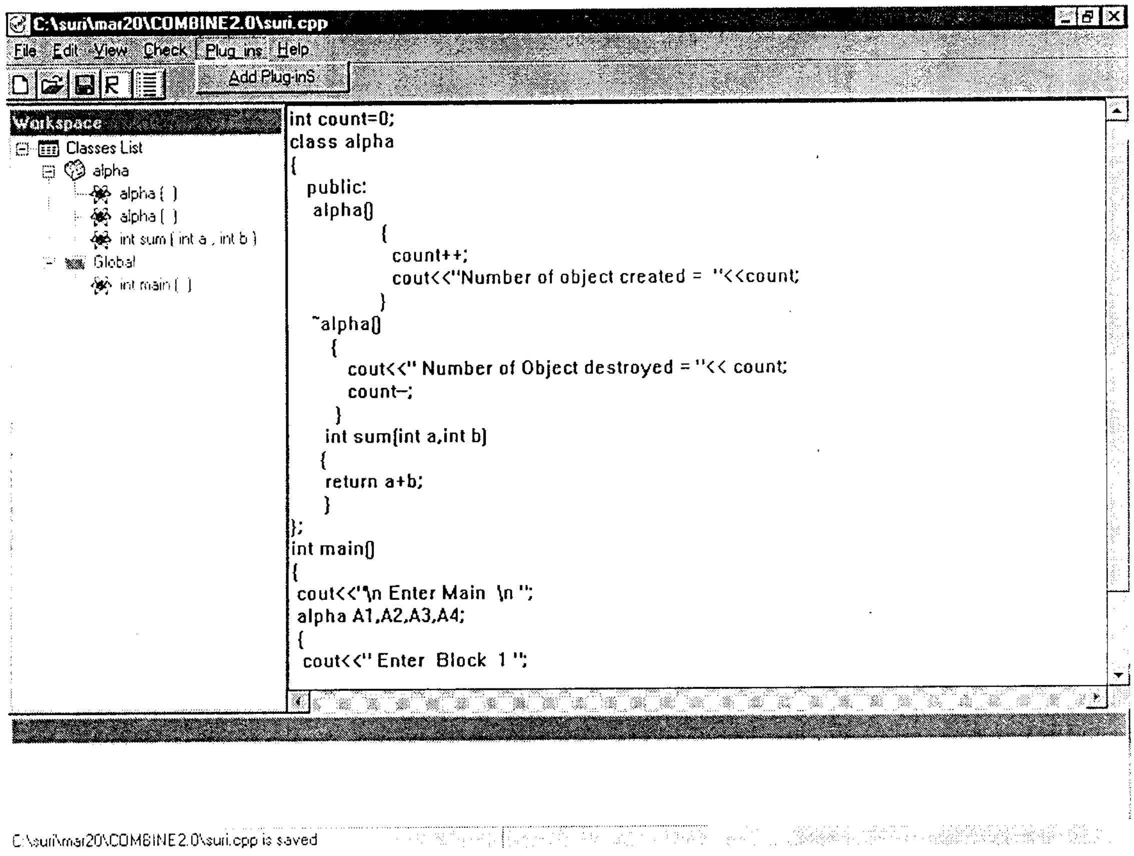
```
int count=0;
class alpha
{
   public:
    alpha()
            {
              count++;
              cout<<"Number of object created =  "<<count;
            }
   ~alpha()
      {
         cout<<" Number of Object destroyed = "<< count;
         count--;
       }
     int sum(int a,int b)
     {
      return a+b;
      }
};
int main()
{
 cout<<"\n Enter Main  \n ";
 alpha A1,A2,A3,A4;
 {
  cout<<" Enter  Block  1 ";
```

C:\suri\mar20\COMBINE2.0\suri.cpp is saved



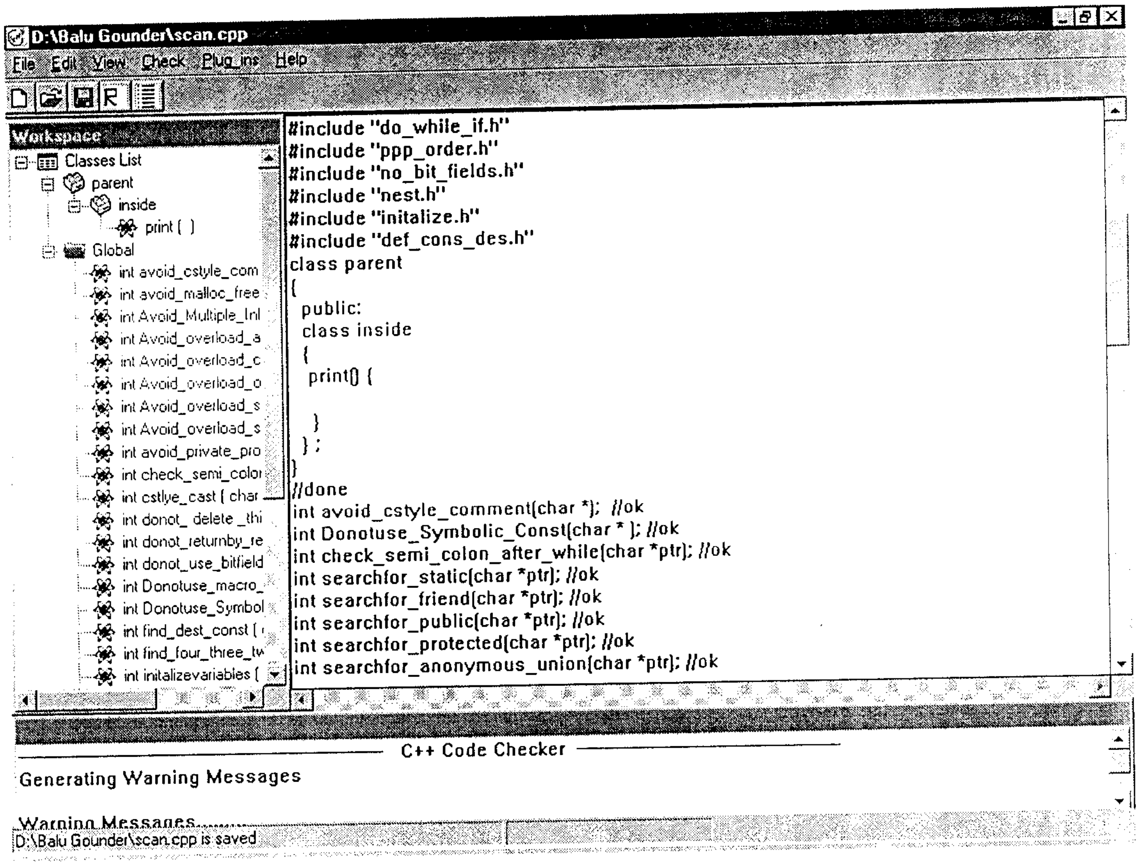
```
int count=0;
class alpha
{
  public:
   alpha()
            {
              count++;
              cout<<"Number of object created =  "<<count;
            }
   ~alpha()
      {
        cout<<" Number of Object destroyed = "<< count;
        count–;
       }
     int sum(int a,int b)
     {
     return a+b;
     }
};
int main()
{
 cout<<"\n Enter Main  \n ";
 alpha A1,A2,A3,A4;
 {
  cout<<" Enter  Block  1 ";
```

C:\suri\mar20\COMBINE2.0\suri.cpp is saved


C:\suri\mar20\COMBINE2.0\suri.cpp
File Edit View Check Plug ins Help
Add Plug-inS
Workspace
Classes List
alpha
alpha ( )
alpha ( )
int sum (int a , int b)
Global
int main ( )

```
int count=0;
class alpha
{
   public:
    alpha()
            {
              count++;
              cout<<"Number of object created =  "<<count;
            }
   ~alpha()
      {
        cout<<" Number of Object destroyed = "<< count;
        count--;
      }
     int sum(int a,int b)
    {
     return a+b;
     }
};
int main()
{
 cout<<"\n Enter Main  \n ";
 alpha A1,A2,A3,A4;
 {
  cout<<" Enter  Block  1 ";
```

C:\suri\mar20\COMBINE2.0\suri.cpp is saved



```
#include "do_while_if.h"
#include "ppp_order.h"
#include "no_bit_fields.h"
#include "nest.h"
#include "initalize.h"
#include "def_cons_des.h"
class parent
{
 public:
 class inside
 {
  print() {

  }
 } ;
}
//done
int avoid_cstyle_comment(char *); //ok
int Donotuse_Symbolic_Const(char * ); //ok
int check_semi_colon_after_while(char *ptr); //ok
int searchfor_static(char *ptr); //ok
int searchfor_friend(char *ptr); //ok
int searchfor_public(char *ptr); //ok
int searchfor_protected(char *ptr); //ok
int searchfor_anonymous_union(char *ptr); //ok
```

```
C++ Code Checker
Generating Warning Messages

Warning Messages.........
```

D:\Balu Gounder\scan.cpp is saved