

Concordance for CppETS 2001 1.0 Test Buckets

Susan Elliott Sim

12 January, 2002 2:58 PM

accuracy

accuracy/preproc

Purpose of preproc group is to determine whether the extractor sees the source code before or after preprocessing and the correctness of the facts generated about preprocessor directives and the resulting source code. An extractor that sees the source code before preprocessing often does not extract the correct information about the resulting source code. An extractor that sees the source code after preprocessing does not get information about preprocessor directives such as macros.

Scoring for this group will be a binary categorisation (before/after preprocessing) and a letter grade.

1 accuracy/preproc/1

This test bucket contains a number of macros and conditional compilation directives, including macros that are expanded to create source code, and a redefinition of a keyword.

1.1 Direct Questions

	Mark Scheme	Q#	Out of
number of preprocessor directives in file	6 different preprocessor keywords, 6 counts for each	1	12
number of macros	4 combinations, 4 counts	2	8
length of function (in lines) created by macro expansion	3 combinations, 3 counts	3	6
length of main (in lines)		5	1

1.2 Indirect Questions

	Mark Scheme	Q#	Out of
length (in characters) of a function	3 combinations, 3 counts	4	6

2 accuracy/preproc/2

This test bucket contains a number of predefined macros.

2.1 Direct Questions

	Mark Scheme	Q#	Out of
location of source line (tests when the code is seen by extractor)	1 for file, 1 for line	1	2
representation of <code>__LINE__</code> in output	1 for showing representation, 1 for explanation	2	2
representation of <code>__STDC__</code> in output	1 for showing representation, 1 for explanation	3	2

3 accuracy/preproc/pragma

This test bucket contains `#pragma` directives.

3.1 Direct Questions

	Mark Scheme	Q#	Out of
start line for main		1	1
representation of <code>#pragma</code> in output	1 for showing representation, 1 for explanation	5	2

3.2 Indirect Questions

	Mark Scheme	Q#	Out of
recognise local variable in main		2	1
identify uses of a variable	5 uses	3	5
distinguish between reads and writes	5 distinctions	3	5
identify string constants	9 string constants	4	9
record length of string constants	9 lengths	4	9

accuracy/syntax

Purpose of syntax group is to test handling of C++ language features.

4 accuracy/syntax/array

Tests handling of array data structures. In C/C++, arrays are blocks of memory addressed by a pointer. Consequently, array and pointer usage often mask each

other. It is also difficult to distinguish use of array entries from use of the array as a whole.

4.1 Direct Questions

	Mark Scheme	Q#	Out of
pointer to primitive type (float) vs. pointer to an object	1 for showing each representation, 1 for explanation	1	3
use of array elements vs. entire array	1 for showing each representation, 1 for explanation	3	3
pointer arithmetic vs. array subscripts	1 for showing each representation, 1 for explanation	4	3

4.2 Indirect Questions

	Mark Scheme	Q#	Out of
representation of parameters in function definitions (pointer to primitive type vs. pointer to object)	1 for showing each representation, 1 for explanation	2	3

5 accuracy/syntax/enum

Test handling of enumeration data structures.

5.1 Direct Questions

	Mark Scheme	Q#	Out of
distinguishing between enumeration constants and the enum structure	1 for showing each representation, 1 for explanation	2	3

5.2 Indirect Questions

	Mark Scheme	Q#	Out of
uses C++ semantics for iterators over enums		1	1
uses C semantics for iterators over enums		1	1
values of enumeration (more generally constants)		3	1

6 accuracy/syntax/exceptions

Tests handling of exceptions (try/catch blocks.)

6.1 Direct Questions

	Mark Scheme	Q#	Out of
detection of try/catch blocks	1 for try block, 2 for catch blocks	1	3
detection of exception types	2 types	2	2
matching try block with the matching catch blocks	2 matches	3	2

7 accuracy/syntax/fcns

Tests detection and resolution of local, global, extern, and library function declaration, definition, and uses. Includes implicit calls to functions via pointers.

7.1 Direct Questions

	Mark Scheme	Q#	Out of
resolution of function uses to linkage points	6 functions, 6 resolutions	1	12
resolving a function with a duplicated name		2	1
resolving calls to a function by name and via pointer	4 calls (1 by name, 3 by pointer)	4	4

7.2 Indirect Questions

	Mark Scheme	Q#	Out of
location information given in bytes from start of file	1 for start, 1 for end	3	2

8 accuracy/syntax/inheritance

Tests recognition of various inheritance keywords and forms. C++ has permission attributes (private, protected, and public) for members methods as well as for inheriting from one or more classes. There are also virtual methods and virtual inheritance (but no virtual classes). Classes can also have friend function. This test case attempts to use as many of these features as possible. Most of these relationships are can be recorded as attributes of something straightforward, although more complex representations can be used.

8.1 Direct Questions

	Mark Scheme	Q#	Out of
--	-------------	----	--------

virtual inheritance (for resolving a dominated hierarchy)	2 points for each of 2 hierarchies, 1 point for noting differences	1	5
using pointer + new vs. variable to instantiate object	1 for showing each representation, 1 for explanation	2	3
resolving an inheritance hierarchy	8 possible classes	3	8
friend function using a variable from an object	1 for identifying friend, 1 for resolving variable use	4	2

9 benchmark/accuracy/namespace

Tests the recognition of the namespace keyword. Also compares the representation of named namespaces with implicit namespaces.

9.1 Direct Questions

	Mark Scheme	Q#	Out of
recognition of namespace keyword	3 uses of namespace keyword	1	3
storage of explicit and implicit namespace information	1 for each namespace (4 implicit, 3 explicit)	2	7
declaration vs. definition of a function from a named namespace	1 for declaration, 1 for definition	3	2

10 accuracy/syntax/operators

Test overloaded addition and function call operators.

10.1 Direct Questions

	Mark Scheme	Q#	Out of
addition.cpp definition of overloaded addition operator	1 for definition	1	1
use of overloaded operator	1 for each use (3)	1	3
fcall.cpp definition of overloaded function call operator	1 for definition	2	1
use of overloaded function call operator	1 for each use (2)	2	2

11 accuracy/syntax/struct

Tests the struct data structure. Includes detection of data fields and function fields.

11.1 Direct Questions

	Mark Scheme	Q#	Out of
detection of struct field types	2 variables, 1 function, plus one for each type (3)	1	6
comparison of data fields and function fields	1 for showing representation of variable, 1 for function, and 1 for comparison	2	3
detection and representation of anonymous struct	1 for showing representation of anonymous struct, 1 for named, and 1 for explanation	3	3

12 accuracy/syntax/templates

Test the templated classes and functions. (This test case is arguably superfluous because the use of cout and << throughout the benchmark are really calls to the Standard Template Library.) There are also different ways to represent templates, as a single abstract entity and as filled-in versions of the code.

12.1 Direct Questions

	Mark Scheme	Q#	Out of
classmember.cpp			
detection of template syntax		1	2
template expansion		2	2
calls to a templated function using primitive types vs. templates		3	3
function.cpp			
template expansion		4	1
expansion representation in extractor output		5	1
representation of generic template		6	1

13 accuracy/syntax/union

Tests definition and use of union.

13.1 Direct Questions

	Mark Scheme	Q#	Out of
use of union data type	3 variables	1	3
use of fields from union	7 uses, 7 locations	5	14

13.2 Indirect Questions

	Mark Scheme	Q#	Out of
detecting the absence of local variables		2	1
size of a variable in memory unit		3	1
size of a struct		4	1

14 accuracy/syntax/vars

Tests recognition and resolution of declarations, definitions, and uses of local, global, scope, and member variables.

14.1 Direct Questions

	Mark Scheme	Q#	Out of
definition and use of variables (global, scope, member, parameter) with the same name	6 different variables, 10 uses	1	16
find definition of specified variable		2	1
overloaded function using different variables with the same name	1 for resolving each variable	4	1

14.2 Indirect Questions

	Mark Scheme	Q#	Out of
value of specified variable at a given line		2	1
definition and use of an overloaded function	1 for showing representation of each of two function signatures , 1 for explanation	3	3

robustness

robustness/missing

The purpose of the missing information group is to see how the extractor copes with missing files (header, source, or library). Software reverse engineers often receive incomplete source code from their clients. The omission may be a simple oversight or the client may not have the right to send third party files. These test cases evaluate whether an extractor can handle these omissions gracefully.

15 robustness/missing/header

Tests whether the extractor can handle missing header files. This test bucket uses the same source code as accuracy/syntax/namespace, but with the header removed. Consequently, users can copy over the "missing header" to verify their results.

15.1 Direct Questions

	Mark Scheme	Q#	Out of
can extractor produce output?	1 for yes/no answer, 1 for showing output, 1 for explanation	1	3
distinguishing struct vs. class without def in header	1 for struct representation, 1 for class representation, 1 for explanation	2	3

15.2 Indirect Questions

	Mark Scheme	Q#	Out of
assumptions made about existence/absence of other missing source	1 for answer, 1 for explanation	3	2

16 robustness/missing/source

Tests whether extractor can gracefully handle a program with missing source code. One file, CardPile.cc, in the program is missing.

16.1 Direct Questions

	Mark Scheme	Q#	Out of
full extraction?	1 for yes/no answer, 1 for showing	1	3

	output, 1 for explanation		
resolve call to method in missing file	1 for showing resolution, 1 for explanation	2	2

16.2 Indirect Questions

	Mark Scheme	Q#	Out of
inheritance from templated classes	1 for showing representation	3	1
recognise const functions	1 for each name	4	5
recognise pure virtual function		5	1
identify unused functions	1 for each function	6	2
identify non-public inheritance	1 for showing answer, 1 for explanation	7	2
file and start line of function decl and def	1 mark for each: decl file, decl line, def file, def line	8	4
identify non-virtual methods	1 for answer, 1 for explanation	9	2
recognise inline methods	1 for each method	10	4

17 robustness/missing/library

The source code in this test bucket has been generated by VisualAge C++ and includes a GUI created by the IBM Open Class Library that is missing. (To be completely accurate, the header files are missing as well and the code uses some VAC++ compiler extensions as well.) Consequently is it being used as an example of generated source code and as missing a library.

Questions 1 and 2 focus on issues associated with a missing library. Questions 3-5 focus on generated source code. The remaining questions (6-9) concern general aspects of the output.

17.1 Direct Questions

	Mark Scheme	Q#	Out of
can extractor produce output?	1 for yes/no answer, 1 for showing output	1	2
how?	1 for explanation	2	1

17.2 Indirect Questions

	Mark Scheme	Q#	Out of
identify classes declared with file and line numbers	1 for each class, 1 for each file, 1 for each location (6 + 6 + 7)	6	19
name public methods in Hello class	1 for each method	7	6

file and line numbers for above	1 for each file, 1 for each start line, 1 for each end line	8	18
type of private data member	1 for type	9	1
use of private data member	1 for each referring function, 1 for each file, 1 for each line (8 + 8 + 8)	9	24

robustness/dialects

Purpose of the dialects group is to evaluate how the extractor handles source code with extensions for the different compilers. These extensions can be viewed as extra keywords or syntax.

18 robustness/dialects/g++

This test bucket evaluates how the extractor handles source code with extensions for the g++ compiler.

18.1 Direct Questions

	Mark Scheme	Q#	Out of
__attribute__() extension	1 for showing representation, 1 for explanation	1	2

18.2 Indirect Questions

	Mark Scheme	Q#	Out of
location of declaration of a data member	1 for line number	2	1
initialization of data member	1 for line number	2	1
representing hexadecimal numbers in extractor output	1 for showing representation of each hexadecimal number	3	5

19 robustness/dialects/msvcpp

This test bucket evaluates how the extractor handles extensions from the Microsoft Visual C++ compiler.

19.1 Direct Questions

	Mark Scheme	Q#	Out of
__cdecl keyword	1 for showing representation, 1 for explanation	1	2

20 robustness/dialects/vacpp

This test bucket evaluates how the extractor handles extensions from the IBM VisualAge C++ compiler.

20.1 Direct Questions

	Mark Scheme	Q#	Out of
extensions (_Optlink, _System, __stdcall, __cdecl, and _Pascal)	1 for showing representation, 1 for explanation	1	5

robustness/hetero

Purpose of the hetero group is to evaluate how the extractor handles a program with mixed source languages. Programming languages are often combined to perform special purpose tasks, for example embedded SQL for interfaces with databases and FORTRAN for scientific computing.

21 robustness/hetero/cfortran

This test bucket contains a file with FORTRAN code and a call to it from the C++ code.

21.1 Direct Questions

	Mark Scheme	Q#	Out of
extern "C" { ... } construct	1 for showing representation, 1 for explanation	1	2
output for FORTRAN file	1 for showing representation, 1 for explanation	4	2

21.2 Indirect Questions

	Mark Scheme	Q#	Out of
other calling constructs	1 for explanation	2	1
asm keyword	1 for explanation	3	1

22 robustness/hetero/esql

This test bucket contains C code written within the Pro*C environment with Embedded SQL statements. ESQL statements are normally preprocessed into normal C (or C++ code), but the original file has been included for two reasons.

1) The file as written is the programmer's view of the system and is the

maintenance artifact. 2) Software reverse engineers don't always have access to the preprocessor and similar tools for legacy code.

22.1 Direct Questions

	Mark Scheme	Q#	Out of
EXEC SQL statements	1 for showing representation, 1 for discussion	2	2
extraction from SQL scripts	1 for showing representation, 1 for discussion	3	2

22.2 Indirect Questions

	Mark Scheme	Q#	Out of
calls to functions without definitions and/or declarations	Identification and discussion	1	5

robustness/generated

Purpose of the generated group is to evaluate how the extractor handles maintenance artifacts that are used to generate C or C++ source code. These files as written by programmers do not look like C/C++ at all, but are often maintained with the rest of the system. Again, software reverse engineers do not always have access to the tools needed to generate the code.

23 robustness/generated/lexyacc

This test bucket contains lex and yacc files in addition to the C files that call them. Lex and yacc are parser-generators that are widely used on UNIX-style operating systems.

23.1 Direct Questions

	Mark Scheme	Q#	Out of
parsing C code within lex and yacc files	1 for showing representation, 1 for explanation	1	2
resolution of data references between .l, .y, .h, and .c files	2 for showing relations, 1 for explanation	3	3

23.2 Indirect Questions

	Mark Scheme	Q#	Out
--	-------------	----	-----

			of
unusual features of generated code	Discussion	2	3

24 robustness/generated/stateflow

This test bucket contains .mdl files from the Stateflow application that are used for modelling state charts for user interfaces (I think). These .mdl files are used to generate code to be used within the Metrowerks CodeWarrior to build applications for the Palm platform.

24.1 Direct Questions

	Mark Scheme	Q#	Out of
extraction from files	1 for showing representation, 1 for discussion	1	2

24.2 Indirect Questions

	Mark Scheme	Q#	Out of
use of stateflow diagrams in pictorial representations of source	Discussion	2	3

25 robustness/generated/vacapp

The source code in this test bucket has been generated by VisualAge C++ and includes a GUI created by the IBM Open Class Library that is missing. (To be completely accurate, the header files are missing as well and the code uses some VAC++ compiler extensions as well.) Consequently is it being used as an example of generated source code and as missing a library.

Questions 1 and 2 focus on issues associated with a missing library. Questions 3-5 focus on generated source code. The remaining questions (6-9) concern general aspects of the output.

25.1 Direct Questions

	Mark Scheme	Q#	Out of
file suffix handling	1 for file suffix used in extraction (.CPP, .H, .APP, .HPP)	3	4
unusual features of generated code?	Discussion	4	3
extraction from resource (.RC, .RCI, .RCX, .VBB) files?	1 for each file	5	4

25.2 Indirect Questions

	Mark Scheme	Q#	Out of
identify classes declared with file and line numbers	1 for each class, 1 for each file, 1 for each location (6 + 6 + 7)	6	19
name public methods in Hello class	1 for each method	7	6
file and line numbers for above	1 for each file, 1 for each start line, 1 for each end line	8	18
type of private data member	1 for type	9	1
use of private data member	1 for each referring function, 1 for each file, 1 for each line (8 + 8 + 8)	9	24