1. System Manager File: “System Manager 1.c++”

1.1. Non-Class Function Pseudo-code

1.1.1. void main(void)

SET terminal window character columns and rows, SET top-left position of terminal window
Output all ulists - instantiates overloaded output stream operators for all ulists.
Call Initialize() to enable graphing inside window to take place.
Initialise calculator class:
    Set constant_list static data member of calculator class by calling:
        calculator::build_internal_constants(calc_constants, number_of_constants);
(Gives all calculators (when instantiated) a constant list specified by name_objects in calc_constants)
Instantiate a single preprocessor object - give [user function name -> OPERATION] mapping array (labels array) and [OPERATION -> token_characters] array from calculator class. Preprocessor object can now translate from user function name to calculator token character.
Initialise complex_container class:
    Set complex_container::postprocessing to point to preprocessor object above.
(complex container objects can now postprocess equation strings so that they appear on screen with calculator tokens expanded to user function names)
Instantiate calculator_manager object - pass to it pointer to the preprocessor object
Instantiate data_manager and graph_manager objects.
Instantiate single graph_device object - can display one graph on screen at once.
Blank graph and draw default axes.

Call data manager display_help to display data manager CLI commands
Call data manager interface
Call graph manager display_help to display graph manager CLI commands
Call graph manager interface

Call calculator manager display_help to display calculator manager CLI commands
LOOP FOREVER
    Call calculator manager interface
    IF extracting graph specs from ‘graph’ calculator successful
        call display_graph() providing calculator manager and graph_device parameters
    ELSE
        Output to screen: "ERROR: No Graph calculator\n"

1.1.2. status extract_graph_specs(calculator_manager &calc_manager, graph_device &graph_)

Backup name of current calculator in calculator_manager.
If setting current calculator to “graph” calculator fails
    RETURN ERROR

SET x_min to real componet of evaluation of calculator order ‘x_min_string’
SET x_max to real componet of evaluation of calculator order ‘x_max_string’
SET horiz_range in graph_device to x_min, x_max values

SET x_scale to real component of evaluation of calculator order ‘x_scale_string’
SET x_tick to real component of evaluation of calculator order ‘x_div_string’
SET horiz_scale in graph_device to x_scale, x_tick values

Repeat above operations for equivalent ‘y’ axis variables, storing results in graph_device

SET var_min to real component of evaluation of calculator order ‘var_min_string’
SET var_max to real component of evaluation of calculator order ‘var_max_string’
SET sample_res to real component of evaluation of calculator order ‘sample_res_string’

Take modulus of sample_res.
Restore current calc to backup calculator (see top of this pseudo code)
RETURN SUCCESS

1.1.3. *void display_graph(calculator_manager &calc_manager, graph_device &graph_)*

Blank graph and draw axes and scales. (by call to graph_.blank_graph())

Get first co-ordinate on graph:
SET order string to “var=%f” where %f is floating point number, var_min.
Send order string to current calculator in calculator manager - assigns var in variable list of calculator.
SET complex ‘y’ to evaluation got from sending order ‘y_string’ to current calc
SET complex ‘x’ to evaluation got from sending order ‘x_string’ to current calc
Move pen to location (graph_.translate_x(x),graph_.translate_y(y)) in graphing window.
Place cross at current pen location.
Set foreground colour to blue - graph line colour connecting crosses

FOR remaining sample points (‘var’ iterates from var_min+sample_res to var_max, sample_res steps)
SET order string to “var=%f” where %f is var (for loop iterator variable)
Send order string to current calculator in calculator manager.
SET double ‘y’ to real component of evaluation got from sending order ‘y_string’ to current calc
SET double ‘x’ to imag component of evaluation got from sending order ‘x_string’ to current calc
Draw line to location (graph_.translate_x(x),graph_.translate_y(y)) in graphing window.
Place cross at current pen location.
ENDFOR

1.1.4. *void Initialize(void)*

Code obtained from CodeWarrior example file. Initialises necessary GUI managers.

1.2. Class Definitions

None

1.3. Additional Declarations

1.3.1. *Constants*

preprocessor_array_length=50
const int number_of_constants=3

1.3.2. *Arrays*

```
const user_label labels[preprocessor_array_length]={
    user_label ("sqrt", "SQUARE_ROOT"),       user_label ("cbrt", "CUBE_ROOT"),
    user_label ("rootx", "ROOTX"), user_label ("sin", "SINE"), user_label ("cos", "COSINE"),
    user_label ("tan", "TANGENT"), user_label ("asin", "ARCSINE"),
    user_label ("acos", "ARCCOSINE"), user_label ("atan", "ARCTANGENT"),
    user_label ("sinh", "SINE-H"), user_label ("cosh", "COSINE-H"),
    user_label ("tanh", "TANGENT-H"), user_label ("asinh", "ARCSINE-H"),
    user_label ("acosh", "ARCCOSINE-H"), user_label ("atanh", "ARCTANGENT-H"),
    user_label ("ln", "NATURAL_LOG"), user_label ("log10", "LOG10"),user_label ("log2", "LOG2"),
    user_label ("logx", "LOGX"), user_label ("arg", "ARGUMENT"), user_label ("MILLI", "MILLI"),
    user_label ("MICRO", "MICRO"), user_label ("NANO","NANO"), user_label ("PICO","PICO"),
    user_label ("FEMTO", "FEMTO"), user_label ("KILO", "KILO"), user_label ("MEGA", "MEGA"),
    user_label ("GIGA", "GIGA"), user_label ("TERA", "TERA"), user_label ("PETA", "PETA"),
    user_label ("EXA",  "EXA"), user_label ("SUM", "SUMMATION"),
    user_label ("PROD", "PRODATION"), user_label ("re", "REAL"), user_label ("im", "IMAGINARY"),
    user_label ("wind", "WINDOW")}

name_object calc_constants[number_of_constants]={
    name_object ("pi", complex_container (CONSTANT, complex(3.1415926535897932385,0))),
    name_object ("e", complex_container (CONSTANT, complex(2.7182818284590452354,0))),
    name_object ("j", complex_container (CONSTANT, complex(0,1)))}
```

1.3.3. *String_class globals: (these and respective double variables used only in this file)*

```
x_min_string="xmin";        x_max_string="xmax";        x_scale_string="xscale";
x_div_string="xdiv";        y_min_string="ymin";        y_max_string="ymax";
y_scale_string="yscale";    y_div_string="ydiv";        var_min_string="varmin";
var_max_string="varmax";    sample_res_string="res";    x_string="xgraph";
y_string="ygraph";
```

2. System Manager File: “System Manager 2.c++”

2.1. Non-Class Function Pseudo-code

2.1.1. void main(void)

Same as for <System Manager 1.c++> up to, “Blank graph and draw default axes”
Instantiate system_process object.
Display system manager commands.
Call system manager interface - passing calculator, data and graph managers, along with system_process and graph_device objects.

2.1.2. void interface(calculator_manager &calc_man, data_manager &data_man, graph_manager &graph_man, system_process &process, graph_device &graph_)

```
Output system manager prompt
WHILE standard input stream contains data, read first word into input_string
    IF input_string is “help”
            Call display_help(); GOTO prompt4: (end of while loop)
    IF input_string is “quit”
            BREAK from WHILE LOOP
    IF input_string is “reset”
            Call calc_man.reset_manager(), data_man.reset_manager(), graph_man.reset_manager()
            Initialise process object to default values.
            Output confirmation: "Global reset done\n"; GOTO prompt4
    IF input_string is “calcman”
            Call calc_man.display_help()
            Call calc_man.interface()
            GOTO prompt4
    IF input_string is “dataman”
            Call data_man.display_help()
            Call data_man.interface()
            GOTO prompt4
    IF input_string is “graphman”
            Call graph_man.display_help()
            Call graph_man.interface()
            GOTO prompt4
    IF input_string is “setprocess”
            Output process object parameters.
            Display process commands - process_help()
            Call set_process_object(process) - reads in all commands/parameters from standard input
            GOTO prompt4
    IF input_string is “viewprocess”
            Output process object parameters
            GOTO prompt4
    IF input_string is “doprocess”
            Output “Calculating process\n”
            Call calculate_process(calc_man, data_man, process)
            GOTO prompt4
    IF input_string is “graphprocess”
            Read in graphtype from input_stream (rect or cornu)
            IF graphtype is rect
                    Prompt user for x-axis/y-axis field names and start/end indexes
                    Read x-axis field, y-axis field, start index, end index from standard input
                    Call display_rect_graph, passing data manager, graph manager, process object,
            x-axis field, y-axis field, start index, end index and graph_device
            ELSE
                    IF graphtype is cornu
                            Prompt user for x-axis field name and start/end indexes
                            Read x-axis field, start index, end index from standard input.
                            Call display_cornu_graph passing data manager, graph manager,
                            process object,     x-axis field, start index, end index and graph_device
                    ELSE
                            Output "ERROR: Invalid graph type\n"
            GOTO prompt4
```

prompt4: (GOTO label)
Output system manager prompt
ENDWHILE

2.1.3. *void display_help()*

Output all system manager commands to standard output.

2.1.4. *void process_help()*

Output all set_process commands to standard output.

2.1.5. *status set_process_object(system_process &process)*

Read in single letter command.
Read in remainder of input up to newline into 'input' char array.
Use input string stream to access 'input'.
IF command is 'd', user is setting process.data_set_name
Read name from input_stream - store in process.data_set_name
ELSE
IF command is 'g', user is setting process.graph_spec_name
Read name from input_stream - store in process.graph_spec_name
ELSE
IF command is 's', user is setting process.set_input_names/set_input_fields lists
Call process.load_list to read input_stream and store set_input_names.
Prompt user for set_input_fields names.
Read input up to newline into 'input' char array
Use second input string stream to access 'input'
Call process.load_list to read input_stream2 and store set_input_fields.
ELSE
IF command is 'm', user is setting process.map_names
Call process.load_list to read input_stream and store map_names.
ELSE
IF command is 'c', user is setting process.calc_name
Read name from input_stream - store in process.calc_name
ELSE
Output "ERROR: Invalid process command\n"
RETURN ERROR

2.1.6. *status calculate_process(calculator_manager &calc_man, data_manager &data_man,system_process &process)*

Initialise 'target', pointer to data_set_obj, to NULL
IF set_current calculator (to process.calc_name) in calculator_manager fails OR
Check that all process.set_input_names names are present in data manager.set_input_list OR
Cross reference process.data_set_name, process.set_input_fields, process.map_names fails
[data_man.check_data_list(process.data_set_name, process.set_input_fields, process.
map_names, target)==ERROR]
RETURN ERROR
(target now points to data_set_obj used to store calculator input/output)
Apply each set_input referenced by process.set_input_names to 'target':
Reset transvers ptr of both process.set_input_names and process.set_input_fields to head of lists
DO
Get copy of current transverse node in set_input_names - store in string_object 'input_name'
Get copy of (corresponding) current transverse node in set_input_fields - store in string_object 'input_field'
Search for set_input_object (name corresponding to string in 'input_name') in data manager set_input_list:
Store result of search in set_input_object 'input_object' - now holds required set_input object
(input_object.get_data() now returns copy of required set_input object.
input_field.get_data() returns field of data_set that set_input object is to be applied to.)
Call target->load_real_input(input_object.get_data(), input_field.get_data()) to apply set_input to data_set
WHILE not at end of set_input_names list, progress transverse pointer in set_input_names and set_input_fields
(target data_set now has all inputs set up for evaluation by calculator)

(must now apply each IO_map referenced by IO_map_names to each row in target data_set array, in sequence)
Get length of target data array.

FOR all rows of elements in data_set data array [i]

reset transverse node of process.map_names list
DO
Get name stored in current transverse node in process.map_names
Search in data manager map list for IO_map_object with identical name, store copy of node in 'map'
Extract input_fields and output_fields contained in 'map' - store in respective variables
Declare complex 'result' object
FOR all fields in input_fields array [j]
(use associative element access with target data_set array)
Store in 'result', data in element [column 'input_fields[j]', row 'i'] of target array
IF out of range ERROR is produced
Output "out of range set_input\n"
RETURN ERROR
Build calculator order string in form "var=a+bj" (var is input_fields[j], a+bj is result)
Send order to current calculator in calculator manager
ENDFOR
(all inputs specified in this IO_map now read from data_set and stored in calculator)
(now evaluate outputs from calculator object (evaluation based on inputs just stored))
FOR all fields in output_fields array [j]
Set appropriate element of data_set to result of calculator order evaluating value of variable named by output_fields[j].
ENDFOR
delete input_fields and output_fields arrays
WHILE end of process.map_name list not reached, progress to next node in map_name list.
ENDFOR
RETURN SUCCESS

2.1.7. status display_rect_graph(data_manager &data_man, graph_manager &graph_man, system_process process, string_class x_axis, string_class y_axis, int start, int finish, graph_device &graph_) system_process process, string_class vector, int start, int finish, graph_device &graph_)

Search (and store found node in graph_spec_obj 'spec_obj') for graph_spec in graph_manager with name: process.graph_spec_name
IF named graph_spec not found
Output "graph spec not found"
RETURN ERROR
Load graph_device with graph_spec parameters contained in spec_obj.
Blank graph and draw scales and axes on graph.
Search (and store found node in data_set_obj 'data_obj') for data_set in data_manager with name: process.data_set_name
IF named data_set not found
Output "data set not found"
RETURN ERROR
IF startor finish indexes are outside bounds of data_obj data_set data array
Output "indices out of range for this data_set"
RETURN ERROR
Get element (field==x_axis, row int==start) associatively from data array - store in 'x'
Get element (field==y_axis, row int==start) associatively from data array - store in 'y'
Move pen to (graph_.translate_x(x.re),graph_.translate_y(y.re)) (use real components x/y)
Place cross at current pen position
Set foreground colour to blue - line colour
FOR all indexes from start+1 to finish, increment index by 1 each iteration [index]
Get element (field==x_axis, row int==index) associatively from data array - store in 'x'
Get element (field==y_axis, row int==index) associatively from data array - store in 'y'
Draw line to (graph_.translate_x(x.re),graph_.translate_y(y.re)) (use real components x/y)
Place cross at current pen position
ENDFOR
RETURN SUCCESS

2.1.8. *status display_cornu_graph(data_manager &data_man, graph_manager &graph_man, system_process process, string_class vector, int start, int finish, graph_device &graph_)*

Search (and store found node in graph_spec_obj 'spec_obj') for graph_spec in graph_manager with
    name: process.graph_spec_name
IF named graph_spec not found
    Output "graph spec not found"
    RETURN ERROR
Clear graph window - no axes.
Search (and store found node in data_set_obj 'data_obj') for data_set in data_manager with name:
    process.data_set_name
IF named data_set not found
    Output "data set not found"
    RETURN ERROR
IF startor finish indexes are outside bounds of data_obj data_set data array
    Output "indices out of range for this data_set"
    RETURN ERROR
Get element (field==vector, row int==start) associatively from data array - store in 'x'
(move pen to origin offset)
Move pen to (graph_.x_origin(spec.horiz_scale.MinTicks*spec.horiz_scale.MajScale),
        graph_.y_origin(spec.vert_scale.MinTicks*spec.horiz_scale.MajScale))
Place cross at current pen position
Set foreground colour to blue - line colour
FOR all indexes from start+1 to finish, increment index by 1 each iteration [index]
    Get element (field==x_axis, row int==index) associatively from data array - store in 'x'
    Draw Line by moving pen relatively from current position:
        Move pen right by x.re*spec.horiz_scale.MajScale   - real component
        Move pen up by -x.im*spec.horiz_scale.MajScale   - imag component
    Place cross at current pen position
ENDFOR
RETURN SUCCESS

2.1.9. *void Initialize(void)*

Same as for <System Manager 1.c++>

2.2. Class Definitions

None

2.3. Additional Declarations

2.3.1. *Constants*

Same as <System Manager 1.c++>

2.3.2. *Arrays*

Same as <System Manager 1.c++>

2.3.3. *String_class globals:*

None

3. Code File: "system_process.c++"

3.1. Non-Class Function Pseudo-code

none

3.2. Class Definitions

3.2.1. *Class "system_process"*

3.2.1.1. Private Member Function Pseudo-code

None

3.2.1.2. Public Member Function Pseudo-code

3.2.1.2.1. void system_process::load_list(istream &input_stream, int list)

Instantiate ‘names_list’ to temporarily store names read in.
WHILE there are names on input_stream (ie read upto newline) - read in single name
Call names_list.add_to_end to store single_name in string_object in names_list
Node is appended to tail of list - no alphabetic ordering on name
ENDWHILE
IF list is 0 copy names_list to set_input_names list
IF list is 1 copy names_list to set_input_fields list
IF list is 2 copy names_list to map_names list

3.2.1.3. Friend Member Function Pseudo-code

3.2.1.3.1. ostream& operator<<(ostream& output_stream, system_process& output_map)

Output "Calculator Name: “ followed by output_map.calc_name, newline
Output “Data Set Name: “ followed by output_map.data_set_name, newline
Output “Graph Spec Name: “ followed by output_map.graph_spec_name, newline
Output “Set Input Names: “ followed by output_map.set_input_names list
Output “Set Input Fields: “ followed by output_map.set_input_fields list
Output “Map Names: “ followed by output_map.map_names
RETURN output_stream

3.2.1.4. Static Member Function Pseudo-code

None

4. Code File: “CalculatorManager.c++”

4.1. Non-Class Function Pseudo-code

none

4.2. Class Definitions

4.2.1. Class “calculator_manager”

4.2.1.1. Private Member Function Pseudo-code

None

4.2.1.2. Public Member Function Pseudo-code

4.2.1.2.1. void calculator_manager::interface()

```
Output manager prompt
IF current_calc not set
    Output current calculator name as NULL
ELSE
    Output current calculator name

WHILE  standard input stream contains data, read first word into input_string
    IF input_string is “add”
            read calculator name from input.
            call add_calculator(name); GOTO prompt (end of while loop)
    IF input_string is “remove”
            read calculator name from input
            call remove_calculator(name); GOTO prompt
    IF input_string is “set_current”
            read calculator name from input
            call set_current_calc(name); GOTO prompt
    IF input_string is “strorage”
            call current_storage(cout); GOTO prompt
    IF input_string is “errors”
            call current_errors(cout); GOTO prompt
    IF input_string is “clear_errors”
            call current_clear_errors(); GOTO prompt
    IF input_string is “clear_memory”
            call clear_memory(); GOTO prompt
    IF input_string is “clear”
            read name from input
            call clear_single_memory(name); GOTO prompt
    IF input_string is “resetmanager”
            call reset_manager(); GOTO prompt
    IF input_string is “verify”
            call validate_current(); GOTO prompt
    IF input_string is “auto_verify_on”
            call auto_verify_on(); GOTO prompt
    IF input_string is “auto_verify_off”
            call auto_verify_off(); GOTO prompt
    IF input_string is “dump”
            cout << *this; GOTO prompt
    IF input_string is “order”
            Read in rest of user input up to newline, store in ‘input’
            Call result=process_order(input)
            IF calculator list not empty
                    IF current calculator has a non-empty error report
                            output error report
            output result of evaluating order; GOTO prompt
    IF input_string is “help”
            Call display_help(); GOTO prompt
    IF input_string is ”return”
            BREAK out of WHILE LOOP

    output error message, indicating that command has not been recognised
    prompt:  (GOTO label)
```

Output manager prompt
IF current_calc not set
Output calculator name as NULL
ELSE
Output calculator name
ENDWHILE

4.2.1.2.2. void calculator_manager::display_help()

Output all commands and syntax to standard output.

4.2.1.2.3. void calculator_manager::reset_manager()

Call calc_list.clearlist(), SET current_calc to NULL

4.2.1.2.4. calculator_manager::calculator_manager(calc_preprocessor *preprocess)

SET preprocessor to preprocess
Allocate new validator to ‘equation_checker’, using ‘preprocessor’ to provide processing service.
SET current_calc to no calculator (NULL)

4.2.1.2.5. calculator_manager::~calculator_manager()

DELETE equation_checker

4.2.1.2.6. calculator_manager::calculator_manager(calculator_manager &original)

Use overloaded = operator to copy original to *this
RETURN *this

4.2.1.2.7. calculator_manager& calculator_manager::operator=(calculator_manager &source)

Copy all data members from source to *this.

4.2.1.2.8. status calculator_manager::add_calculator(const string_class name)

IF call to calc_list.add fails (ordering on ASCENDING name)
output error message- new calculator must have unique name
RETURN ERROR
ELSE
set current_calc to new calculator
set new calculator’s validator to *this.equation_checker
RETURN SUCCESS

4.2.1.2.9. status calculator_manager::remove_calculator(const string_class name)

IF call to calc_list.remove(name) succeeds
set_current_calc to head of calc_list
RETURN SUCCESS
ELSE
output error message - name not found in calc list
RETURN ERROR

4.2.1.2.10. ostream& calculator_manager::current_storage(ostream &output_stream)

IF calculator list is empty
output error message to cout
output “Manager ERROR - current calculator not set\n” to output_stream
ELSE
output contents of *current_calc to output_stream (output_stream << *current_calc)
RETURN output_stream

4.2.1.2.11. ostream& calculator_manager::current_errors(ostream &output_stream)

IF calculator list is empty
output error message to cout
output "Manager ERROR - current calculator not set\n" to output_stream
ELSE
Call peek_errors(output_stream) to output error report to output_stream
RETURN output_stream

4.2.1.2.12. status calculator_manager::validate_current()

IF calculator list is not empty
    IF call to equation_checker->validate on current calculator succeeds
        output confirmation to cout
        RETURN SUCCESS
    ELSE
        output error message
        output error trace (equation_checker->error_trace)
        RETURN ERROR
ELSE
    Output error message: current calculator not set
    RETURN ERROR

4.2.1.2.13. string_class calculator_manager::current_clear_errors()

IF calculator list is empty
    output error message to cout
    RETURN "Manager ERROR - current calculator not set\n"
ELSE
    RETURN result of calling current_calc->flush_errors() (error report for current calc)

4.2.1.2.14. string_class calculator_manager::clear_memory()

IF calculator list is empty
    RETURN "Manager ERROR - current calculator not set\n"
ELSE
    Call current_calc->all_clear()
    RETURN “”

4.2.1.2.15. string_class calculator_manager::clear_single_memory(string_class name)

IF calculator list is empty
    RETURN "Manager ERROR - current calculator not set\n"
ELSE
    IF call to current_calc->clear_single_name(name) fails
        RETURN "Invalid name\n"
    ELSE
        RETURN “Cleared\n”

4.2.1.2.16. complex calculator_manager::process_order(const string_class &calc_order)

IF calculator list is empty
    output error message to standard output
ELSE
    Preprocess ‘calc_order’ string, storing result in ‘preprocessed_string’ (use preprocessor object)
    Evaluate order by calling current_calc->evaluate(preprocessed_string)
    RETURN evaluation result
RETURN NULLcomplex

4.2.1.2.17. void calculator_manager::auto_verify_on()

IF calculator list is empty
    output error message to standard output
ELSE
    IF error occurs when calling current_calc->auto_verify_on()
        output error message to standard output
    ELSE
        output confirmation message to standard output

4.2.1.2.18. void calculator_manager::auto_verify_off()

IF calculator list is empty
    output error message to standard output
ELSE
    output confirmation message to standard output
    call current_calc->auto_verify_off()

4.2.1.2.19. string_class calculator_manager::get_current_calc()

IF calculator list is empty

RETURN empty string, “”
ELSE
RETURN current_calc name by calling current_calc->get_index()

4.2.1.2.20. status calculator_manager::set_current_calc(const string_class name_string)

IF resetting transverse pointer of calc_list causes ERROR
IF name_string is empty
output: "Manager ERROR - can't set current calculator when calculator list is empty!\n"
SET current_calc to no calculator (NULL)
RETURN ERROR
IF name_string is empty
SET current_calc to head of calculator list (call calc_list.get_transverse_node_pointer())
RETURN SUCCESS
DO
Get transverse list pointer
IF name of calc_object pointed to by transverse pointer matches name_string
SET current_calc pointer to transverse pointer
WHILE end of calculator list not reached, progress transverse pointer
output error message to standard output - calc_object identified by name_string not found
RETURN ERROR

4.2.1.3. Friend Member Function Pseudo-code

4.2.1.3.1. ostream& operator<<(ostream& output_stream, calculator_manager &output)

Output "\nCalculator Manager List:\n\n" on output_stream
IF resetting transverse pointer of calculator list in ‘output’ causes ERROR
Output "No Calculators Present\n"
RETURN output_stream
DO
Get copy of calc_object pointed to by transverse pointer
Output using overloaded output of calc_object, name & variable/equation lists for single calculator
WHILE end of ‘output’.calc_list not reached, progress transverse pointer
RETURN output_stream

4.2.1.4. Static Member Function Pseudo-code

None

5. Code File: “rCalculatorUserTypes.c++”

5.1. Non-Class Function Pseudo-code

none

5.2. Class Definitions

5.2.1. Class “token_name”

5.2.1.1. Private Member Function Pseudo-code

None

5.2.1.2. Public Member Function Pseudo-code

5.2.1.2.1. token_name::token_name()

Initialise token to enum NAME.

5.2.1.2.2. token_name::token_name(const char *tokenname, const char token_character)

SET name to tokenname.
SET token to token_character

5.2.1.3. Friend Member Function Pseudo-code

5.2.1.3.1. ostream& operator<<(ostream& output_stream, const token_name output)

output_stream << output.name << " -> " << output.token

5.2.1.4. Static Member Function Pseudo-code

None

5.2.2. Class “calculator_symbol”

5.2.2.1. Private Member Function Pseudo-code

None

5.2.2.2. Public Member Function Pseudo-code

5.2.2.2.1. void calculator_symbol::clear()

Initialise all data members

5.2.2.3. Friend Member Function Pseudo-code

None

5.2.2.4. Static Member Function Pseudo-code

None

6. Code File: “extraclasses.c++”

6.1. Non-Class Function Pseudo-code

none

6.2. Class Definitions

6.2.1. Class “complex_container”

6.2.1.1. Private Member Function Pseudo-code

None

6.2.1.2. Public Member Function Pseudo-code

6.2.1.2.1. complex_container::complex_container()

Initialise all data members.

6.2.1.2.2. complex_container::complex_container(macro_type store_indicator, complex store_complex_number)

SET indicator to COMPLEX.
SET complex_number to store_complex_number.
IF store_complex_number is EQUATION
    Output warning message to cout - mismatched indicator setting for complex number storage.

6.2.1.2.3. complex_container::complex_container(string_class equation_string)

SET indicator to EQUATION.
SET equation to equation_string

6.2.1.3. Friend Member Function Pseudo-code

6.2.1.3.1. ostream& operator<<(ostream& output_stream, const complex_container container)

IF container.indicator is CONSTANT
    Output container.complex_number followed by ‘\n’
IF container.indicator is EQUATION
    Use postprocess function of ‘postprocessing’ to expand calculator tokens.
    Output “EQUATION=” followed by expanded equation followed by ‘\n’
IF container.indicator is neither CONSTANT/EQUATION
    Output error message
RETURN output_stream

6.2.1.3.2. istream& operator>>(istream& input_stream, complex_container& container)

Skip initial white space on input.
Putback last character read into input stream for re-reading.
Read string delimited by white space into ‘buffer’.
Report error if buffer was not large enough to contain entire input string.
IF buffer holds ‘CONSTANT’
    SET container.indicator to CONSTANT
    Read in complex number, storing in container.complex_number.
IF buffer holds ‘EQUATION’
    SET container.indicator to EQUATION
    Read in characters until end of line reached, store string in container.equation.
RETURN input_stream

6.2.1.3.3. compare compare_containers(complex_container left, complex_container right)

IF left.indicator<right.indicator: RETURN SMALLER

ELSE IF left.indicator>right.indicator: RETURN LARGER
ELSE IF left.indicator==CONSTANT
Compare real components of complex number
IF reals are equal compare imag components of complex number
RETURN result of real/imag comparisons
ELSE lexically compare left.equation and left2.equation
RETURN comparison result (LARGER/SMALLER/EQUAL)

6.2.1.4. Static Member Function Pseudo-code

None

7. Code File: "rCalculatorClass.c++"

7.1. Non-Class Function Pseudo-code

none

7.2. Class Definitions

7.2.1. Class "calculator"

7.2.1.1. Private Member Function Pseudo-code

7.2.1.1.1. void calculator::new_error_stream()

IF number_of_errors defined, delete it.
Allocate new number_of_errors - init to 0.
IF error_string defined, delete it.
Allocate new error_string - init to ""
IF error_stream defined, delete it.
Allocate new error_stream.

7.2.1.1.2. complex calculator::level1()

Offer control to Level2 - returns left operand - store result in 'left'
FOREVER
SWITCH (token in current_symbol)
CASE PLUS: (infix operation)
IF call to get_token returns END of input_stream
Call error("Incomplete binary operation")
RETURN NULLcomplex
Offer control to Level2 - returns right operand
SET left to addition of both operands
BREAK
CASE MINUS: (infix operation)
IF call to get_token returns END of input_stream
Call error("Incomplete binary operation")
RETURN NULLcomplex
Offer control to Level2 - returns right operand
SET left to subtraction of both operands (left-right)
BREAK
DEFAULT:
RETURN left (this level does not process current token)
ENDSWITCH
ENDFOR

7.2.1.1.3. complex calculator::level2()

Offer control to Level3 - returns left operand - store result in 'left'
FOREVER
SWITCH (token in current_symbol)
CASE MUL: (infix operation)
IF call to get_token returns END of input_stream
Call error("Incomplete binary operation")
RETURN NULLcomplex
Offer control to Level3 - returns right operand
SET left to multiplitcation of both operands
BREAK
CASE DIV: (infix operation)
IF call to get_token returns END of input_stream

```
                    IF call to get_token returns END of input_stream
                        Call error("Incomplete binary operation")
                        RETURN NULLcomplex
                    Offer control to Level3 - returns right operand
                    IF call invalid_operands finds divide by 0
                        Call error("divide by 0")
                        RETURN NULLcomplex
                    SET left to left divided by right operand
                    BREAK
                DEFAULT:
                    RETURN left (this level does not process current token)
            ENDSWITCH
        ENDFOR
```

7.2.1.1.4. complex calculator::level3()

```
        Offer control to Level4 - returns left operand - store result in 'left'
        FOREVER
            SWITCH (token in current_symbol)
                CASE POW:    (infix operation)
                    IF call to get_token returns END of input_stream
                        Call error("Incomplete binary operation")
                        RETURN NULLcomplex
                    Offer control to Level4 - returns right operand (exponent)
                    IF call invalid_operands returns ERROR
                        Call error("invalid POWER operands")
                        RETURN NULLcomplex
                    SET left to result of left raised to power of exponent
                    BREAK
                CASE ROOTX:  (infix operation)
                    IF call to get_token returns END of input_stream
                        Call error("Incomplete binary operation")
                        RETURN NULLcomplex
                    Offer control to Level4 - returns right operand
                    IF call invalid_operands returns ERROR
                        Call error("invalid ROOTX operands")
                        RETURN NULLcomplex
                    SET left to result of 'left' operand root of right operand
                    BREAK
                CASE LOGX:   (infix operation)
                    IF call to get_token returns END of input_stream
                        Call error("Incomplete binary operation")
                        RETURN NULLcomplex
                    Offer control to Level4 - returns right operand
                    IF call invalid_operands returns ERROR
                        Call error("invalid LOGX operands")
                        RETURN NULLcomplex
                    SET left to result of 'left' base log of right operand
                    BREAK
                DEFAULT:
                    RETURN left (this level does not process current token)
            ENDSWITCH
        ENDFOR
```

7.2.1.1.5. complex calculator::level4()

```
Offer control to Level5 - returns left operand - store result in ‘left’
FOREVER
    SWITCH (token in current_symbol)
        CASE FACTORIAL:     (postfix operation)
            Call get_token to store next token in current_symbol.
            IF call invalid_operands returns ERROR
                Call error(“invalid FACTORIAL operand: non-integer /  non-real”)
                RETURN NULLcomplex
            SET left to result of factorial of left
            BREAK
        DEFAULT:
            RETURN left (this level does not process current token)
    ENDSWITCH
ENDFOR
```

7.2.1.1.6. complex calculator::level5()

```
Offer control to Primary - returns left operand - store result in ‘left’
FOREVER
    SWITCH (token in current_symbol)
        CASE (ENGINEERING SYMBOL):          (postfix operation)
            Make backup of eng symbol token (from current_symbol.token)
            Call get_token to store next token in current_symbol.
            Call engineering_conversion to multiply ‘left’ by appropriate power of 10
            BREAK
        DEFAULT:
            RETURN left (this level does not process current token)
    ENDSWITCH
ENDFOR
```

7.2.1.1.7. complex calculator::primary()

```
SWITCH (token in current_symbol)
    CASE NUMBER:
        Call get_token to store next token in current_symbol.
        IF current token is NAME/NUMBER
            Call error(“Name/Number cannot proceed a Number”)
            RETURN NULLcomplex
        RETURN  current symbol number_value (still valid after get_token call)

    CASE NAME:
        Backup current_symbol name_string.
        IF no equation with name name_string in equation_list
            IF no variable with name name_string in variable_list
                IF no constant with name name_string in constant_list
                    Name not known
                ELSE
                    Flag Name as a CONSTANT
            ELSE
                Flag Name as a VARIABLE
        ELSE
            Flag Name as an EQUATION

        SWITCH on get_token result
            CASE NAME:
                Call error(“Name cannot proceed a Name”)
                RETURN NULLcomplex
                BREAK

            CASE ASSIGN_CONSTANT:     (= decoded from input_stream)
                IF name known as equation or constant
                    Call error(“Illegal Assignment to equation/constant”)
                    RETURN NULLcomplex
```

```
                    Call get_token to store next token in current_symbol.
                    Backup current number of errors.
                    Call Level1 - returns assign value - store in 'temp_complex'
                    IF current no. of errors == backup of no. of errors
                        Store variable with temp_complex value in variable_list
                    ELSE
                        Call error("Right hand side of assignment is faulty")
                    RETURN temp_complex
                    BREAK

                CASE DEFINE_EQUATION:    (: decoded from input_stream)
                    IF name known as variable or constant
                        Call error("Illegal tagging of variable/constant name")
                        RETURN NULLcomplex
                    Call get_definition() to store rest of input_stream in string.
                    Store equation name with definition string in equation_list
                    IF auto verification on
                        IF call to VALIDATE on equation_list returns ERROR
                            remove the new equation from equation_list
                            report error trace by calling error function.
                    RETURN NULLcomplex
                    BREAK

                DEFAULT:    (name present as reference in expression)
                    IF name not known
                        report erroneous name by calling error function
                        RETURN NULLcomplex
                    ELSE
                        SWITCH on data stored in name
                        CASE CONSTANT
                            RETURN complex number associated with name
                        CASE EQUATION
                            Extract equation associated with name
                            Instantiate sub_calculator object
                            Use sub_calculator to evaluate equation
                            Delete sub_calculator object
                            BREAK
                        ENDSWITCH
                BREAK
            ENDSWITCH

        CASE MINUS:        (UNARY MINUS)
            Call get_token to store next token in current_symbol.
            Offer control to Primary - returns operand.
            RETURN -operand

        CASE LP:        (Left Parenthesis)
            Increment Binding Level
            Call get_token to store next token in current_symbol.
            Offer control to Level1 - returns evaluation of expression inside this level of parentheses
            IF token of current symbol is not RP (right parenthesis)
                Reset Parenthesis Level to 0
                Call error(") expected")
                RETURN NULLcomplex
            Call get_token to store next token in current_symbol.
            Decrement Binding Level
            RETURN complex number evaluation of bounded expression

        CASE LM:        (Left Modulus)
            Increment Binding Level
            Call get_token to store next token in current_symbol.
            Offer control to Level1 - returns evaluation of expression inside this binding level
            IF token of current symbol is not RP (right parenthesis)
```

```
                Reset Parenthesis Level to 0
                Call error(“) expected”)
                RETURN NULLcomplex
        Call get_token to store next token in current_symbol.
        Decrement Binding Level
        Calculate complex modulus of result from Level1
        RETURN modulus calculation

CASE SQRT:          (square root)
        Call get_token to store next token in current_symbol.
        Offer control to Level3 - returns operand for calculation
        IF call invalid_operands returns ERROR
                Call error(“invalid SQRT operands”)
                RETURN NULLcomplex
        RETURN complex square root of operand

CASE CBRT:          (cube root)
        Call get_token to store next token in current_symbol.
        Offer control to Level3 - returns operand for calculation
        IF call invalid_operands returns ERROR
                Call error(“invalid CBRT operands”)
                RETURN NULLcomplex
        RETURN complex cube root of operand

CASE SIN/COS/TAN/ASIN/ACOS/ATAN/SINH/COSH/TANH/ASINH/ACOSH/ATANH:
CASE LN/LOG10/LOG2:
        Backup token in current_symbol.
        Call get_token to store next token in current_symbol.
        Offer control to Level3 - returns operand for calculation
        IF call invalid_operands returns ERROR
                Call error(“invalid trig/log operand”)
                RETURN NULLcomplex
        Look up relevant math function required in the math_func function pointer table.
        RETURN result of calling required function with operand as function parameter

CASE ARGUMENT:
        Call get_token to store next token in current_symbol.
        Offer control to Level3 - returns operand for calculation
        RETURN complex argument of operand

CASE REAL:
        Call get_token to store next token in current_symbol.
        Offer control to Level3 - returns operand for calculation
        RETURN real component of operand

CASE IMAGINARY:
        Call get_token to store next token in current_symbol.
        Offer control to Level3 - returns operand for calculation
        RETURN imaginary component of operand

CASE WINDOW:
        Call get_window_parameters to get variable, lower_bound and upper_bound
        If call to get_window_parameters returns ERROR
                RETURN NULLcomplex
        SET current symbol.token to NAME
        SET current symbol.name_string to variable to window on
        Offer control to Primary - returns complex number associated with variable
        IF result is within window range
                RETURN complex number=1+0j    (multiply factor of 1)
        ELSE
                RETURN NULLcomplex            (multiply factor of 0)
```

```
        CASE SUMMATION:
                RETURN result of calling compound(PLUS)

        CASE PRODATION:
                RETURN resutl of calling componund(MUL)

        CASE END:
                RETURN NULLcomplex

        DEFAULT:
                Call error("primary expected")
                RETURN NULLcomplex
    ENDSWITCH
```

7.2.1.1.8. complex calculator::compound(token_value method)

```
    Call get_compound_parameters to obtain variable, upper_bound, lower_bound
    If call to get_compound_parameters returns ERROR
        RETURN NULLcomplex
    IF doing a SUMMATION initialise total to 0+0j
    ELSE IF doing a PRODATION initialise total to 1+0j
    IF variable name references an equation in the calculator equation_list
            Call error("Illegal compound function variable name - clashes with valid equation name")
            RETURN NULLcomplex
    Record current offset position for input_stream.
    Set expression_start  char pointer to point to current position in input_stream.
    FOR all integers between lower_bound and upper_bound of variable
        Call get_token to store next token in current_symbol.
        SET variable in variable_list of calculator to current iteration value.
        SWITCH (method)
            CASE PLUS:
                Offer control to Level2 - returns calulation for this value of variable
                Add result to total
                BREAK
            CASE MUL:
                Offer control to Level2 - returns calulation for this value of variable
                Multiply total by result
                BREAK
        ENDSWITCH
        IF upperbound of variable NOT reached
            DELETE input_stream
            Allocate new input_stream pointing to expression_start
    ENDFOR
    RETURN total
```

7.2.1.1.9. status calculator::get_compound_parameters(token_value method, string_class &variable, int &lower_bound, int &upper_bound)

```
    SWITCH (method)
        CASE PLUS:
            SET error string to "SUMMATION: "
            SET lower_bound to 0
        CASE PLUS:
            SET error string to "PRODATION: "
            SET lower_bound to 1
    ENDSWITCH

    Call get_token to store next token in current_symbol.
    IF current symbol token is LP (left parenthesis)
        Call get_token to store next token in current_symbol.
        IF current symbol token is NAME
            SET variable to current symbol name_string
            Call get_token to store next token in current_symbol.
            IF current symbol token is COMMA
                Call get_token to store next token in current_symbol.
```

```
                    Offer control to Level1 - returns upper_bound
                    IF current symbol token is not RP (Right Parenthesis)
                        IF current symbol token is COMMA
                            Call get_token to store next token in current_symbol.
                            Set lower_bound to upper_bound
                            Offer control to Level1 - returns upper_bound
                            IF current symbol token is not RP (Right Parenthesis)
                                Call error(error_string+"Illegal syntax - RP absent")
                                RETURN ERROR
                            ELSE
                                IF lower_bound>upper_bound
                                    Call error(error_string+"Illegal bounds")
                                RETURN ERROR
                                ELSE
                                    RETURN SUCCESS(var, upper, lower
okay)
                        ELSE
                            Call error(error_string+"Illegal syntax - Comma/RP absent")
                            RETURN ERROR
                    ELSE
                        IF lower_bound>upper_bound
                            Call error(error_string+"Illegal bounds - lower>upper")
                            RETURN ERROR
                        ELSE
                            RETURN SUCCESS (var, upper params okay)
                ELSE
                    Call error(error_string+"illegal syntax - comma absent")
                    RETURN ERROR
            ELSE
                Call error(error_string+"illegal syntax - variable name absent")
                RETURN ERROR
        ELSE
            Call error(error_string+"illegal syntax - left parenthesis absent")
            RETURN ERROR
```

7.2.1.1.10. status calculator::get_window_parameters(string_class &variable, double &lower_bound, double &upper_bound)

```
Call get_token to store next token in current_symbol.
IF current symbol token is LP (left parenthesis)
    Call get_token to store next token in current_symbol.
    IF current symbol token is NAME
            SET variable to current symbol name_string
            Call get_token to store next token in current_symbol.
            IF current symbol token is COMMA
                    Call get_token to store next token in current_symbol.
                    Offer control to Level1 - returns upper_bound
                    IF current symbol token is COMMA
                            Call get_token to store next token in current_symbol.
                            Offer control to Level1 - returns upper_bound
                            IF current symbol token is not RP (Right Parenthesis)
                                    Call error(error_string+"Illegal syntax - RP absent")
                                    RETURN ERROR
                            ELSE
                                    IF lower_bound>upper_bound
                                            Call error(error_string+"Illegal bounds")
                                    RETURN ERROR
                                    ELSE
                                            RETURN SUCCESS(var, upper, lower okay)
                    ELSE
                            Call error(error_string+"Illegal syntax - Comma absent")
                            RETURN ERROR
            ELSE
                    Call error(error_string+"illegal syntax - comma absent")
                    RETURN ERROR
    ELSE
            Call error(error_string+"illegal syntax - variable name absent")
            RETURN ERROR
ELSE
    Call error(error_string+"illegal syntax - left parenthesis absent")
    RETURN ERROR
```

7.2.1.1.11. string_class calculator::get_definition()

```
SET definition_string to ""
DO
    Read in single_char from input_stream
    IF end of input_stream reached
            BREAK
    append single_char to definition_string
WHILE end of input_stream not reached
RETURN definition_string
```

7.2.1.1.12. void calculator::init_math_array()

```
IF math_function_array_initialised is 0 (ie no other calculator objects instantiated yet)
    SET math_function_array_initialised to 1
    Call initialise_math_function_array()
```

7.2.1.1.13. complex calculator::engineering_conversion(const token_value token, const complex x)

```
Multiply real and imag components of x by appropriate power of 10, depending on token value
RETURN x
```

7.2.1.1.14. int calculator::invalid_operands(const token_value token, const complex value1, const complex value2)

```
For each token type, return result of relational expression testing value1 and value2.
If expression evaluates to true, 1 is returned - signifying invalid operands.
```

7.2.1.1.15. token_value calculator::get_token()

DO
    Read single input_char from input_stream
    IF end_of stream reached
        IF not in a binding section and current symbol token is not a terminator token
            Call error("Illegal Terminating Token")
        SET current_symbol.token to END
        RETURN END
WHILE newline not reached and input_char is white space
SWITCH on input_char
    CASE any valid token character NOT (digit or decimalpoint)
        SET current_symbol.token to equivalent token value for token character
        RETURN current_symbol.token
    CASE 0,1,2,3,4,5,6,7,8,9, <decimalpoint>
        Putback last character read into input_stream
        Read entire floating point number into current_symbol.number_value.re (treat as real)
        SET current_symbol.token to NUMBER
        RETURN NUMBER
    DEFAULT: (this is an alphanumeric user name)
        IF (input_char isalpha)
            Putback last character read into input_stream
            Read entire name into current_symbol.name_string
            SET current_symbol.token to NAME
            RETURN NAME
        ELSE
            Call error("bad token");
            SET current_symbol.token to PRINT
            RETURN PRINT
ENDSWITCH

7.2.1.1.16. complex calculator::error(const char* s)

Output "error:" and char array 's' and newline character to error_stream
Increment *number_of_errors
RETURN NULLcomplex

7.2.1.1.17. complex calculator::error(const string_class s)

Output "error:" and string_class 's' and newline character to error_stream
Increment *number_of_errors
RETURN NULLcomplex

7.2.1.1.18. calculator::calculator(ulist<name_object> *v_list, ulist<name_object> *exp_list, ostrstream *errors, int *number_of_err)

SET equation_checker, input_stream, input_char_array, error_string to NULL
SET error_stream to errors, SET number_of_errors to number_of_err
SET binding_segment, auto_verify to 0
SET rank to SUB_CALCULATOR

7.2.1.1.19. status calculator::set_input(string_class input_string)

IF input_stream exists, call reset_input()
Allocate new input_char_array, and copy char array of input_string into this array.
Allocate new input_stream pointing to input_char_array.
RETURN SUCCESS

7.2.1.1.20. status calculator::reset_input()

IF input_stream exists, delete and set to NULL input_char_array and input_stream, RETURN SUCESS
ELSE output warning of attempt to reset non-set input_stream, RETURN ERROR

7.2.1.2. Public Member Function Pseudo-code

7.2.1.2.1. calculator::calculator()

SET equation_checker, input_stream, input_char_array, error_stream, error_string, number_of_errors to NULL.
Call init_math_array()
SET binding_segment to 0, SET auto_verify active, SET rank to SUPER_CALCULATOR

Allocate new variable and equation lists.
Call new_error_stream()

7.2.1.2.2. calculator::~calculator()

IF rank is SUPER_CALCULATOR
    DELETE variable and equation lists, error_string, error_stream and number_of_errors

7.2.1.2.3. calculator::calculator(const calculator &original)

Use overloaded = operation to do *this=original

7.2.1.2.4. calculator& calculator::operator=(const calculator &original)

Allocate new variable and equation lists.
Set *this lists to point to 'original's lists.
Copy from original to *this, values of: equation_checker, rank, binding_segment, current_symbol, auto_verify.
SET input_stream, char_input_array, error_string, error_stream and number_of_errors to NULL.
IF original.error_string is not empty
    Allocate new error_string/error_stream.
    Copy original.error_string into *this error_stream.
    Allocate new number_of_errors integer.
    Copy *original.number_of_errors into **this.number_of_errors
ELSE
    Call new_error_stream()
RETURN *this

7.2.1.2.5. ulist<name_object>* calculator::get_equation_list()

RETURN equation_list

7.2.1.2.6. ulist<name_object>* calculator::get_var_list()

RETURN var_list

7.2.1.2.7. complex calculator::evaluate(string_class input_string)

Call set_input(input_string) :sets input_stream to extract chars from input_string
Call current_symbol.clear()
WHILE end of input_stream not reached
    Call get_token()
    IF current symbol token is END
        BREAK out of while loop
    IF current symbol token is PRINT
        CONTINUE : go back to top of WHILE LOOP
    Call complex_result=level(1) : initiates recursive function chain
IF errors are present in the error stream
    output number of errors occurred since last clearing of errors.
Call reset_input()
RETURN complex_result

7.2.1.2.8. string_class calculator::flush_errors()

IF rank is SUPER_CALCULATOR terminate error_stream with NULL character ('\0')
Backup error_string.
Call new_error_stream()
RETURN backup of error_string.

7.2.1.2.9. ostream& calculator::peek_errors(ostream& output_stream)

IF rank is SUPER_CALCULATOR
    Get character offset of error_stream.
    IF error_stream contains error reports
        MEMcopy offset bytes out of error_stream and copy into 'errors' string
        Terminate 'errors' string with NULL character ('\0')
        Output 'errors' to output_stream
    ELSE
        Output "0 errors" to output_stream
RETURN output_stream

7.2.1.2.10. int calculator::get_number_of_errors()

RETURN *number_of_errors

7.2.1.2.11. void calculator::all_clear()

Call var_list->clearlist()
Call equation_list->clearlist()

7.2.1.2.12. status calculator::clear_single_memory(string_class name)

IF removal of 'name' from variable list fails
    IF removal of 'name' from equation list fails
        RETURN ERROR
    ELSE RETURN SUCCESS
ELSE RETURN SUCCESS

7.2.1.2.13. void calculator::auto_verify_off()

SET auto_verify to 0

7.2.1.2.14. status calculator::auto_verify_on()

IF call to validate all equations fails
    RETURN ERROR
ELSE
    SET auto_verify to 1
    RETURN SUCCESS

7.2.1.2.15. void calculator::set_validator(validator *checker)

SET equation_checker to checker

7.2.1.3. Friend Member Function Pseudo-code

7.2.1.3.1. ostream& operator<<(ostream& output_stream, calculator a)

Output on output_stream, "Variable List: " followed by contents of *variable_list
Output on output_stream, "Equation List: " followed by contents of *equation_list
RETURN output_stream

7.2.1.4. Static Member Function Pseudo-code

7.2.1.4.1. void calculator::initialise_math_function_array()

See source code: each token value affected is used an index to math_func array, where pointer to equivalent C math function is stored. eg math_func[(unsigned)SIN]=&sin
FOR all terminator_table array elements [i]
    Set element (indexed by value of (unsigned char)terminator_tokens[i]) of terminator_table to '!'

7.2.1.4.2. void calculator::build_internal_constants(const name_object *const_array,const int length)

Instantiate ulist<name_object> called build_list
FOR all name_object elements in const_array
    add current name_object to build_list (ASCENDING order of NAME)
Copy build_list to static data member calculator::constant_list

7.3. Other definitions/declarations

7.3.1. Simple Static Data Instantiations

```
int calculator::math_function_array_initialised=0;
char calculator::terminator_table[256];
double (*calculator::math_func[256])(double);
ulist<name_object> calculator::constant_list;
```

```
ulist<name_object> calculator::constant_list;
int calculator::ERROR_STREAM_SIZE=1000;
```

7.3.2. Compound Static Data Instantiations

```
const token_value calculator::terminator_tokens[15]={ token_value(NUMBER), token_value(NAME),
token_value(FACTORIAL), token_value(PRINT), token_value(MILLI), token_value(MICRO),
token_value(NANO), token_value(PICO), token_value(FEMTO), token_value(KILO), token_value(MEGA),
token_value(GIGA), token_value(TERA), token_value(PETA), token_value(EXA) };

const token_name calculator::token_names[50]={
token_name ("PLUS",'+'),
token_name ("MINUS", '-'),
token_name ("MULTIPLY", '*'),
token_name ("DIVIDE", '/'),
token_name ("POWER",'^'),
token_name ("FACTORIAL", '!'),
token_name ("SQUARE_ROOT", '@'),
token_name ("CUBE_ROOT", '£'),
token_name ("ROOTX", '$'),
token_name ("SINE",'¡'),
token_name ("COSINE", '™'),
token_name ("TANGENT",'#'),
token_name ("ARCSINE", '¢'),
token_name ("ARCCOSINE", '∞'),
token_name ("ARCTANGENT", '§'),
token_name ("SINE-H", '~'),
token_name ("COSINE-H", 'Ω'),
token_name ("TANGENT-H",'≈'),
token_name ("ARCSINE-H",'ç'),
token_name ("ARCCOSINE-H", '√'),
token_name ("ARCTANGENT-H", 'ʃ'),
token_name ("NATURAL_LOG", '¶'),
token_name ("LOG10", '•'),
token_name ("LOG2", 'ª'),
token_name ("LOGX", 'º'),
token_name ("PRINT", ';'),
token_name ("ASSIGN_CONSTANT",'='),
token_name ("LEFT_BRACKET",'('),
token_name ("RIGHT_BRACKET", ')'),
token_name ("LEFT_MODULUS",'['),
token_name ("RIGHT_MODULUS", ']'),
token_name ("ARGUMENT",'≠'),
token_name ("MILLI",'œ'),
token_name ("MICRO",'Σ'),
token_name ("NANO",'®'),
token_name ("PICO",'†'),
token_name ("FEMTO",'¥'),
token_name ("KILO", 'å'),
token_name ("MEGA", 'ß'),
token_name ("GIGA",'∂'),
token_name ("TERA",'ƒ'),
token_name ("PETA", '©'),
token_name ("EXA",'Δ'),
token_name ("DEFINE_EQUATION",':'),
token_name ("SUMMATION", '¬'),
token_name ("PRODATION", 'Π'),
token_name ("COMMA", ','),
token_name ("REAL", '≤'),
token_name ("IMAGINARY", '≥'),
token_name ("WINDOW",'÷')};
```

8. Code File: “Data Manager.c++”

8.1. Non-Class Function Pseudo-code

none

8.2. Class Definitions

8.2.1. Class “data_manager”

8.2.1.1. Private Member Function Pseudo-code

None

8.2.1.2. Public Member Function Pseudo-code

8.2.1.2.1. void data_manager::interface()

```
Output manager prompt
WHILE standard input stream contains data, read first word into input_string
    IF input_string is “help”
        Call display_help(); GOTO prompt2: (end of while loop)
    IF input_string is “newrecord”
        read record name from input.
        read remainder of input upto newline into ‘input’ string
        call new_record(input, name); GOTO prompt2
    IF input_string is “delrecord”
        read record name from input.
        call delete_record(name); GOTO prompt2
    IF input_string is “records”
        Output entire record_list to standard output using overloaded output operator
        GOTO prompt2
    IF input_string is “newdataset”
        read data_set name from input.
        read record_name from input.
        read length of data set array from input.
        call new_data(name, record_name, length); GOTO prompt2
    IF input_string is “deldataset”
        read data_set name from input.
        call delete_data(name); GOTO prompt2
    IF input_string is “datas”
        Output entire data_list to standard output using overloaded output operator
        GOTO prompt2:
    IF input_string is “newinput”
        read set_input name from input.
        read lower_index, upper_index, start_value and incrementer from input.
        call new_set_input(name, lower_index, upper_index, start_value, incrementer)
        GOTO prompt2
    IF input_string is “delinput”
        read set_input name from input.
        call delete_set_input(name); GOTO prompt2
    IF input_string is “inputs”
        Output entire set_input_list to standard output using overloaded output operator
        GOTO prompt2:
    IF input_string is “newmap”
        read IO_map name from input.
        read remainder of input line upto newline into ‘input’ string
        prompt user “now enter output fields\n”
        read input line upto newline into ‘output’ string
        call new_map(name, input, output)
        GOTO prompt2
    IF input_string is “delmap”
        read IO_map name from input.
        call delete_map(name); GOTO prompt2
    IF input_string is “maps”
        Output entire map_list to standard output using overloaded output operator
        GOTO prompt2
    If input_string is “resetmanager”
        Call reset_manager(); GOTO prompt2
```

IF input_string is ”return”
BREAK out of WHILE LOOP
prompt2: (GOTO label)
Output manager prompt
ENDWHILE

8.2.1.2.2. void graph_manager::display_help()
Output all commands and syntax to standard output.

8.2.1.2.3. status data_manager::new_record(string_class record_name, string_class record)
Extract char array from record - store in record_chars
Create an input string stream pointing to record_chars - called record_stream
Instantiate a string_object ulist, called field_names
WHILE end of record_stream not reached, read word into field_chars from record_stream
If adding a string_object with ‘field_chars’ data to field_names list causes ERROR
output error message indicating non-unique field name
RETURN ERROR
ENDWHILE
IF adding a record_object with record_name and field_names data to record_list causes ERROR
output error message indicating non-unique record_object name
RETURN ERROR
RETURN SUCCESS

8.2.1.2.4. status data_manager::delete_record(string_class name)
IF removing a record_object identified by ‘name’ from record_list causes ERROR
output error message indicating record_object not found
RETURN ERROR
ELSE
RETURN SUCCESS

8.2.1.2.5. void data_manager::output_record_list()
cout << record_list

8.2.1.2.6. status data_manager::new_data(string_class data_set_name, string_class record_name, int length)
IF search for record_obj in record_list with ‘record_name’ identifier unsuccessful
output error message indicating non-existent record_object with identifier ‘record_name’
RETURN ERROR
IF adding a data_set_obj with data_set_name and (record_name, length) to data_list causes ERROR
output error message indicating non-unique data_set_obj name
RETURN ERROR
ELSE
RETURN SUCCESS

8.2.1.2.7. status data_manager::delete_data(string_class name)
IF removing a data_set_obj identified by ‘name’ from data_list causes ERROR
output error message indicating data_set_obj not found
RETURN ERROR
ELSE
RETURN SUCCESS

8.2.1.2.8. void data_manager::output_data_list()
cout << data_list

8.2.1.2.9. status data_manager::check_data_list(string_class data_name, ulist<string_object> inputfields, ulist<string_object> mapnames, data_set_obj* &target)
Instantiate data_set_obj data.
SET found to 0
IF data list reset transverse causes ERROR
output "ERROR: data_list empty\n"
ERROR

(Search for data_set with data_name in data_list)

```
DO
    Copy transverse node from data_list into ‘data’
    IF ‘data’ has name matching data_name
            SET target to point to current transverse node (insitu of data_list)
            SET found to 1
            BREAK
WHILE end of data_list not reached, progress transverse pointer to next node in data_list

IF data_set not found, ie found=0
    output "ERROR: data object not found"
    RETURN ERROR

Store number of columns in data_set in int ‘width’.
Allocate array to store copy of fields array in data_set - called ‘fields’ , with ‘width’ elements
Call data.get_fields(fields) - stores copy of fields in data_set

(Check all inputfields strings arevalid field names in data_set)
Instantiate string_object extracted
IF inputfields list reset transverse causes ERROR
    output "ERROR: checking data_set fields against empty list of inputfields"
    RETURN ERROR
DO                                                  check out all names stored in inputfields list
    Copy transverse node from inputfields list into ‘extracted’
    SET found to 0
    FOR all elements in fields array [field_index]
            IF fields[field_index] matches extracted.get_data()
                    SET found to 1
                    BREAK out of FOR LOOP
    ENDFOR
    IF field name not found, ie found=0
            output “ERROR: field name not found”
            RETURN ERROR
WHILE end of inputfields list not reached, progress transverse pointer to next node in inputfields list

(Check all mapnames names found in map_list - if found check input/output fields are valid for
data_set)
Instantiate IO_map_object temp
IF mapnames reset transverse is successful
    DO                                              check out all names stored in mapnames
            Copy transverse node from mapnames into ‘string_node’
            Search (storing found node in temp) for IO_map in map_list with
                    name=temp.get_data()
            IF IO_map not found
                    output “ERROR: map object not found”
                    RETURN ERROR

            (check input fields in current IO_map are valid for data_set)
            Get number of input fields in temp, allocate array called ‘input_fields’ to hold all
elements.
            Copy input_fields from IO_map in temp to ‘input_fields’.
            FOR all input_fields elements [i]
                    SET found to 0
                    FOR all elements in ‘fields’ array [j] (these are names known to data_set)
                            IF fields[j] matches input_fields[i]
                                    SET found to 1
                                    BREAK
                    ENDFOR
                    IF field not found, ie found=0
                            output “ERROR: input field name not known”
                            RETURN ERROR
            ENDFOR

            (check output fields in current IO_map are valid for data_set)
```

Get no. of output fields in temp, allocate array called ‘output_fields’ - holds all elements.
Copy output_fields from IO_map in temp to ‘output_fields’.
FOR all output_fields elements [i]
SET found to 0
FOR all elements in ‘fields’ array [j] (these are names known to data_set)
IF fields[j] matches output_fields[i]
SET found to 1
BREAK
ENDFOR
IF field not found, ie found=0
output “ERROR: output field name not known”
RETURN ERROR
ENDFOR
WHILE end of mapnames list not reached, progress transverse to next node in mapnames list
RETURN SUCCESS
ELSE
output "ERROR: checking map list against empty list of names"
RETURN ERROR

8.2.1.2.10. status data_manager::new_map(string_class map_name, string_class input_fields,string_class output_fields)

Extract char array from input_fields - store in input_chars
Create an input stream pointing to input_chars - called input_stream
Instantiate a string_object ulist, called field_names
WHILE end of input_stream not reached, read word into field_chars from input_stream
IF adding a string_object with ‘field_chars’ data to field_names list causes ERROR
output error message indicating non-unique input field name
RETURN ERROR
ENDWHILE
Extract char array from output_fields - store in output_chars
Create an input stream pointing to output_chars - called input_stream2
Instantiate a string_object ulist, called field_names2
WHILE end of input_stream2 not reached, read word into field_chars from input_stream2
IF adding a string_object with ‘field_chars’ data to field_names2 list causes ERROR
output error message indicating non-unique output field name
RETURN ERROR
ENDWHILE
IF adding an IO_map_object with map_name and (field_names, field_names2) data to map_list causes ERROR
output error message indicating non-unique IO_map_object name
RETURN ERROR
ELSE
RETURN SUCCESS

8.2.1.2.11. status data_manager::delete_map(string_class name)

IF removing an IO_map_object identified by ‘name’ from map_list causes ERROR
output error message indicating IO_map_object not found
RETURN ERROR
ELSE
RETURN SUCCESS

8.2.1.2.12. void data_manager::output_map_list()

cout << map_list

8.2.1.2.13. status data_manager::new_set_input(string_class set_input_name, int lower_i, int upper_i, float start_val, float increment)

IF adding a set_input_object with 'set_input_name' and (lower_i, upper_i, start_val, increment) data to set_input_list causes ERROR
    output error message indicating non-unique set_input_object name
    RETURN ERROR
ELSE
    RETURN SUCCESS

8.2.1.2.14. status data_manager::delete_set_input(string_class set_input_name)

IF removing a set_input_object identified by 'set_input_name' from set_input_list causes ERROR
    output error message indicating set_input_object not found
    RETURN ERROR
ELSE
    RETURN SUCCESS

8.2.1.2.15. void data_manager::output_set_input_list()

cout << set_input_list

8.2.1.2.16. status data_manager::check_set_input_list(ulist<string_object> sourcelist)

Instantiate string_object string_node.
IF source list not empty, reset transverse node pointer of sourcelist
    DO
        Store node pointed to by sourcelist transverse pointer in string_node
        Search (storing found node in string_node) for set_input in set_input_list with
            name=string_node.get_data()
        IF NOT found
            output "ERROR: set_input object not found"
            RETURN ERROR
    WHILE end of sourcelist not reached, progress transverse to next node in source list
    RETURN SUCCESS
ELSE
    Output "ERROR: checking set_input list against empty list of names"
    RETURN ERROR

8.2.1.2.17. void data_manager::reset_manager()

Call record_list.clearlist(), call data_list.clearlist(), call map_list.clear_list()
Call set_input_list.clear_list()

8.2.1.3. Friend Member Function Pseudo-code

None

8.2.1.4. Static Member Function Pseudo-code

None

9. Code File: "data_set.c++"

9.1. Non-Class Function Pseudo-code

none

9.2. Class Definitions

9.2.1. Class "data_set"

9.2.1.1. Private Member Function Pseudo-code

None

9.2.1.2. Public Member Function Pseudo-code

9.2.1.2.1. data_set::data_set()

SET data_array, fields to NULL - SET record_name to ""
SET length, width to 0

9.2.1.2.2. data_set::data_set(record_object record_obj, const int array_length)

Store copy of stringobject list stored in record_obj in ‘field_list’
Store copy of name stored in record_obj in ‘record_name’
Count number of nodes in field_list - store in width
Allocate fields array to hold all strings stored in field_list nodes
Reset transverse of field_list to head of list.
FOR all nodes in field_list
    Store string in tranverse node of field_list into fields array
    Progress transverse node to next node in field_list
Allocate data_array with length*width elements
FOR all elements of data_array
    set complex number to 0+0j

9.2.1.2.3. data_set::data_set(const data_set &original)

Use overloaded = operator to copy - *this=original

9.2.1.2.4. data_set& data_set::operator=(const data_set &original)

Copy non-dynamic data from original to *this. (record_name, length, width)
IF fields array in original not NULL
    Allocate fields array in *this to be width elements large
    FOR all elements in original fields array
        Copy data in equivalent element in original fields array to *this fields array
ELSE
    set fields to NULL
IF data_array in original not NULL
    allocate data_array in *this of size length*width
    FOR all elements in *this data_array
        Copy data in equivalent element in original data_array to *this data_array
    ENDFOR
ELSE
    set data array to NULL

9.2.1.2.5. data_set::~data_set()

Deallocate data_array and fields

9.2.1.2.6. status data_set::get_element_assoc(complex &result, const int index, const string_class field)

IF data_array is not NULL
    SET int field_index to invalid value
    FOR all elements in fields array [i]
        IF element [i] of fields array matches ‘field’
            SET field_index to [i]
            BREAK
    ENDFOR
    IF field_index is invalid or ‘index’ is out of bounds (<0 or >=length)
        SET result to NULLcomplex
        RETURN ERROR
    SET result to complex number stored in element referenced by index and field_index
    Use array mapping function (AMF) for 2-d array: result=*(data_array+width*index+field_index)
    RETURN SUCCESS
ELSE
    SET result to NULLcomplex
    RETURN ERROR

9.2.1.2.7. status data_set::get_element(complex &result, const int index, const int field_index)

```
IF data_array is not NULL
    IF index or field_index out of bounds of data_array dimensions
            RETURN NULLcomplex
            RETURN ERROR
    SET result to complex number stored in element referenced by index and field_index: use AMF
    RETURN SUCCESS
ELSE
    SET result to NULLcomplex
    RETURN ERROR
```

9.2.1.2.8. status data_set::set_element_assoc(const complex new_value, const int index, const string_class field)

```
IF data_array is not NULL
    SET int field_index to invalid value
    FOR all elements in fields array [i]
            IF element [i] of fields array matches 'field'
                    SET field_index to [i]
                    BREAK
    ENDFOR
    IF field_index is invalid or 'index' is out of bounds (<0 or >=length)
            RETURN ERROR
    SET element referenced by index and field_index to 'new_value' : use AMF
    RETURN SUCCESS
ELSE
    RETURN ERROR
```

9.2.1.2.9. status data_set::set_element(const complex new_value, const int index, const int field_index)

```
IF data_array is not NULL
    IF index or field_index out of bounds of data_array dimensions
            RETURN ERROR
    SET element referenced by index and field_index to 'new_value' : use AMF
    RETURN SUCCESS
ELSE
    RETURN ERROR
```

9.2.1.2.10. int data_set::get_length()

```
RETURN length
```

9.2.1.2.11. int data_set::get_width()

```
RETURN width
```

9.2.1.2.12. string_class data_set::get_record_name()

```
RETURN record_name
```

9.2.1.2.13. void data_set::get_fields(string_class *storage)

```
IF data_array is not NULL
    FOR all elements in fields array [i]
            Copy string stored in element [i] of fields array to element [i] of storage array.
```

9.2.1.2.14. status data_set::clear_data_array()

```
IF data_array is not NULL
    FOR length*width elements in data_array [i]
            Set element [i] of data_array to 0+0j.
```

9.2.1.3. Friend Member Function Pseudo-code

9.2.1.3.1. ostream& operator<<(ostream& output_stream, data_set output_set)

```
Declare 'number' of complex object type.
IF output_set.data_array not NULL
    output "Record Base:" followed by record_name, newline
    output "Dimensions: " followed by length "x" width, newline
```

```
        output “Field Names: “ followed by width
        FOR all elements in fields array [i]
                output string stored in element [i] of fields array followed by single space
        output newline
        FOR all rows in data_array [i]
                output  [i] followed by tab
                FOR all columns in data_array [j]
                        Call get_element(number, i, j)
                        output ‘number’
                ENDFOR
                output newline
        ENDFOR
    ELSE
        output error message “data_set not initialised”
    RETURN output_stream
```

9.2.1.4. Static Member Function Pseudo-code

None

10. Code File: “set_input.c++”

10.1. Non-Class Function Pseudo-code

none

10.2. Class Definitions

10.2.1. Class “set_input”

10.2.1.1. Private Member Function Pseudo-code

None

10.2.1.2. Public Member Function Pseudo-code

10.2.1.2.1. set_input::set_input()

SET all data members to 0

10.2.1.2.2. set_input::set_input(int lower_i, int upper_i, float start_val, float inc)

Call set_parameters to set data members to values specified in function call

10.2.1.2.3. set_input::~set_input()

Do nothing - there is no dynamic data to deallocate.

10.2.1.2.4. set_input::set_input(set_input &original)

Copy all data members from original to *this

10.2.1.2.5. set_input& set_input::operator=(set_input &original)

Copy all data members from original to *this

10.2.1.2.6. void set_input::set_parameters(int lower_i, int upper_i, float start_val, float inc)

SET data members according to function parameter values

10.2.1.2.7. void set_input::get_parameters(int &lower_i, int &upper_i, float &start_val, float &inc)

Variable parameters of function SET to values of respective data members

10.2.1.3. Friend Member Function Pseudo-code

10.2.1.3.1. ostream& operator<<(ostream& output_stream, const set_input output_set)

Output "lower index=" followed by output.lower_index, newline
Output "upper index=" followed by output.upper_index, newline
Output "start value=" followed by output.start_value, newline
Output "incrementer=" followed by output.incrementer, newline
Output newline
RETURN output_stream

10.2.1.4. Static Member Function Pseudo-code

None

11. Code File: “IO_map.c++”

11.1. Non-Class Function Pseudo-code

None

11.2. Class Definitions

11.2.1. Class “IO_map”

11.2.1.1. Private Member Function Pseudo-code

None

11.2.1.2. Public Member Function Pseudo-code

11.2.1.2.1. IO_map::IO_map()

SET input_length & output_length to 0; SET input_fields, output_fields to NULL

11.2.1.2.2. IO_map::IO_map(ulist<string_object> in_fields, ulist<string_object> out_fields)

Store number of nodes in in_fields ulist in ‘input_length’.
Dynamically allocate memory for in_fields, number_of_elements=input_length
RESET transverse pointer of in_fields ulist
FOR each element of input_fields array [i]
    Copy DATA from transverse node in in_fields ulist to element [i] in input_fields array.
    Progress transverse pointer to next node in in_fields ulist.

Store number of nodes in in_fields ulist in ‘output_length’.
Dynamically allocate memory for out_fields, number_of_elements=output_length
RESET transverse pointer of out_fields ulist
FOR each element of output_fields array [i]
    Copy DATA from transverse node in out_fields ulist to element [i] in output_fields array.
    Progress transverse pointer to next node in in_fields ulist.

11.2.1.2.3. IO_map::IO_map(IO_map &original)

*this = original; USE overloaded = operator to achieve copy operation

11.2.1.2.4. IO_map& IO_map::operator=(const IO_map &original)

Copy non-dynamic data from oriignal to *this.
Initialise input_fields pointer to NULL.
Dynamically allocate sufficient memory to hold original.input_fields array.
Copy all elements from original.input_fields into *this.input_fields

Initialise output_fields pointer to NULL.
Dynamically allocate sufficient memory to hold original.output_fields array.
Copy all elements from original.output_fields into *this.output_fields

11.2.1.2.5. IO_map::~IO_map()

Delete input_fields, delete output_fields

11.2.1.2.6. int IO_map::get_number_of_input_fields()

11.2.1.2.7. void IO_map::get_input_fields(string_class *storage)

11.2.1.2.8. int IO_map::get_number_of_output_fields()

11.2.1.2.9. void IO_map::get_output_fields(string_class *storage)

For these functions, return relevant private data member.

11.2.1.3. Friend Member Function Pseudo-code

11.2.1.3.1. ostream& operator<<(ostream& output_stream, const IO_map output_map)

IF output.input_fields is not NULL
    Output all elements of input_fields array, and all elements of output_fields array.
ELSE
    Output "ERROR: IO_map not initialised\n"
RETURN output_stream

11.2.1.4. Static Member Function Pseudo-code

None

12. Code File: “Graph Manager.c++”

12.1. Non-Class Function Pseudo-code

none

12.2. Class Definitions

12.2.1. Class “graph_manager”

12.2.1.1. Private Member Function Pseudo-code

None

12.2.1.2. Public Member Function Pseudo-code

12.2.1.2.1. void graph_manager::interface()

```
Output manager prompt
WHILE  standard input stream contains data, read first word into input_string
    IF input_string is “help”
            Call display_help(); GOTO prompt3: (end of while loop)
    IF input_string is “newspec”
            read spec name from input.
            read spec_input data using overloaded input operator.
            Call new_spec(name, spec_input); GOTO prompt3
    IF input_string is “delspec”
            read spec name from input.
            call delete_spec(name); GOTO prompt3
    IF input_string is “specs”
            call output_spec_list(); GOTO prompt3
    IF input_string is “reset_manager”
            call reset_manager(); GOTO prompt3
    IF input_string is ”return”
            BREAK out of WHILE LOOP
    prompt3:                    (‘GOTO’ label)
    Output manager prompt
ENDWHILE
```

12.2.1.2.2. void graph_manager::display_help()

Output all commands and syntax to standard output.

12.2.1.2.3. status graph_manager::new_spec(string_class name, graph_spec new_spec)

```
If adding a graph_spec_obj with ‘name’ and ‘new_spec’ data to spec_list causes ERROR
    output error message indicating non-unique graph_spec_obj name
    RETURN ERROR
ELSE
    RETURN SUCCESS
```

12.2.1.2.4. status graph_manager::delete_spec(string_class name)

```
IF removing a graph_spec_obj identified by ‘name’ from spec_list causes ERROR
    output error message indicating graph_spec_obj not found
    RETURN ERROR
ELSE
    RETURN SUCCESS
```

12.2.1.2.5. void graph_manager::output_spec_list()

```
cout << spec_list
```

12.2.1.2.6. void graph_manager::reset_manager()

```
Call spec_list.clearlist()
```

12.2.1.3. Friend Member Function Pseudo-code

None

12.2.1.4. Static Member Function Pseudo-code

None

13. Code File: “graph_spec.c++”

13. Code File: “graph_spec.c++”

13.1. Non-Class Function Pseudo-code

none

13.2. Class Definitions

13.2.1. Class “graph_spec”

13.2.1.1. Private Member Function Pseudo-code

None

13.2.1.2. Public Member Function Pseudo-code

13.2.1.2.1. graph_spec::graph_spec()

SET scaleSize to 10; all other data members automatically defaulted by respective constructors

13.2.1.2.2. graph_spec::graph_spec(port_info port1, border_info border1, scale_info horiz_scale1, scale_info vert_scale1, range_info horiz_range1, range_info vert_range1, tick_info ticks1)

Copy each function parameter into respective data member

13.2.1.3. Friend Member Function Pseudo-code

13.2.1.3.1. ostream& operator<<(ostream &output_stream, graph_spec spec)

Output Horiz range label, followed by output_stream.horiz_range data.
Output Horiz scale label, followed by output_stream.horiz_scale data, followed by newline.
Output Vert range label, followed by output_stream.vert_range data.
Output Vert scale label, followed by output_stream.vert_scale data followed by newline.
RETURN output_stream

13.2.1.3.2. istream& operator>>(istream& input_stream, graph_spec& spec)

Prompt user for horizontal data
Use spec.horiz_scale input operator to read data for this data member.
Use spec.horiz_range input operator to read data for this data member.
Prompt user for vertical data
Use spec.vert_scale input operator to read data for this data member.
Use spec.vert_range input operator to read data for this data member.
RETURN input_stream

13.2.1.4. Static Member Function Pseudo-code

None

14. Code File: “extragraphclasses.c++”

14.1. Non-Class Function Pseudo-code

none

14.2. Class Definitions

14.2.1. Class “port_info”

14.2.1.1. Private Member Function Pseudo-code

None

14.2.1.2. Public Member Function Pseudo-code

14.2.1.2.1. port_info::port_info (double WIDTH, double HEIGHT)

Set data members to function parameters

14.2.1.2.2. port_info::port_info (double TOP, double LEFT, double WIDTH, double HEIGHT)

Set data members to function parameters

14.2.1.3. Friend Member Function Pseudo-code

14.2.1.3.1. ostream& operator<<(ostream& output_stream, port_info port)

Output each data member with label, RETURN output_stream

14.2.1.4. Static Member Function Pseudo-code

None

14.2.2. Class "border_info"

14.2.2.1. Private Member Function Pseudo-code

None

14.2.2.2. Public Member Function Pseudo-code

14.2.2.2.1. border_info::border_info (double TOP, double BOTTOM, double LEFT, double RIGHT)

Set data members to function parameters

14.2.2.3. Friend Member Function Pseudo-code

14.2.2.3.1. ostream& operator<<(ostream& output_stream, border_info border)

Output each data member with label, RETURN output_stream

14.2.2.4. Static Member Function Pseudo-code

None

14.2.3. Class "bound_info"

14.2.3.1. Private Member Function Pseudo-code

None

14.2.3.2. Public Member Function Pseudo-code

14.2.3.2.1. bound_info::bound_info()

Set all data members to 0

14.2.3.3. Friend Member Function Pseudo-code

14.2.3.3.1. ostream& operator<<(ostream& output_stream, bound_info bound)

Output each data member with label, RETURN output_stream

14.2.3.4. Static Member Function Pseudo-code

None

14.2.4. Class "scale_info"

14.2.4.1. Private Member Function Pseudo-code

None

14.2.4.2. Public Member Function Pseudo-code

14.2.4.2.1. scale_info::scale_info (double MAJ, double MIN)

Set data members to function parameters

14.2.4.3. Friend Member Function Pseudo-code

14.2.4.3.1. ostream& operator<<(ostream& output_stream, scale_info scale)

Output each data member with label, RETURN output_stream

14.2.4.3.2. istream& operator>>(istream& input_stream, scale_info& scale)

cout << "Scale Division? "
input_stream >> scale.MajScale
cout << "Inter-Divisions? "
input_stream >> scale.MinTicks
RETURN input_stream

14.2.4.4. Static Member Function Pseudo-code

None

14.2.5. Class "range_info"

14.2.5.1. Private Member Function Pseudo-code

None

14.2.5.2. Public Member Function Pseudo-code

14.2.5.2.1. range_info::range_info (double MIN, double MAX)

Set data members to function parameters

14.2.5.2.2. double range_info::range()

RETURN max-min

14.2.5.3. Friend Member Function Pseudo-code

14.2.5.3.1. ostream& operator<<(ostream& output_stream, range_info range)

Output each data member with label, RETURN output_stream

14.2.5.3.2. istream& operator>>(istream& input_stream, range_info& range)

cout << "Range Min? "
input_stream >> range.Min
cout << "Range Max? "
input_stream >> range.Max
RETURN input_stream

14.2.5.4. Static Member Function Pseudo-code

None

14.2.6. Class "tick_info"

14.2.6.1. Private Member Function Pseudo-code

None

14.2.6.2. Public Member Function Pseudo-code

14.2.6.2.1. tick_info::tick_info(double MAJ, double MIN)

Set data members to function parameters

14.2.6.3. Friend Member Function Pseudo-code

14.2.6.3.1. ostream& operator<<(ostream& output_stream, tick_info tick)

Output each data member with label, RETURN output_stream

14.2.6.4. Static Member Function Pseudo-code

None

15. Code File: "graph_device.c++"

15.1. Non-Class Function Pseudo-code

none

15.2. Class Definitions

15.2.1. Class "graph_device"

15.2.1.1. Private Member Function Pseudo-code

None

15.2.1.2. Public Member Function Pseudo-code

15.2.1.2.1. graph_device::graph_device()

SET mainPtr to NULL, SET scaleSize to 10 point

15.2.1.2.2. graph_device::graph_device(port_info newport, border_info newborder, scale_info newhoriz_scale, scale_info newvert_scale, range_info newhoriz_range, range_info newvert_range, tick_info newticks)

SET all data members to respective function parameter values.
SET mainPtr to NULL, SET scaleSize to 10 point

15.2.1.2.3. void graph_device::set_params(scale_info newhoriz_scale, scale_info newvert_scale, range_info newhoriz_range, range_info newvert_range)

SET horiz_scale to newhoriz_scale
SET vert_scale to newvert_scale
SET horiz_range to newhoriz_range
SET vert_range to newvert_range

15.2.1.2.4. void graph_device::showGraph()

Set graph port to graphing window, mainPtr
Call OS function, ShowWindow(mainPtr)
Call OS function, SelectWindow(mainPtr)

15.2.1.2.5. void graph_device::clear_window()

Call showGraph()
Set rectangle with parameters such that rectangle covers whole graphing window
Set graph port to graphing window, mainPtr
Call OS function, EraseRect passing rectangle described above - clears window.

15.2.1.2.6. void graph_device::blank_graph()

Call clear_window() - sets graph port to graphing window
Call vertical_ticks() and horizontal_ticks() to draw both axes and scales
Call draw_axes() to draw y=0, x=0 lines if in visible graph area

15.2.1.2.7. double graph_device::translate_x(double x)

Convert x axis location to x pixel location, return:
bound.left+(x-horiz_range.Min)*bound.width/horiz_range.range()

15.2.1.2.8. double graph_device::translate_y(double y)

Convert y axis location to y pixel location, return:
bound.bottom-(y-vert_range.Min)*bound.height/vert_range.range()

15.2.1.2.9. void graph_device::draw_x_line(double x, long int brightness)

Backup current foreground colour
SET foreground colour to grey colour (shade determined by brightness 0 black, 65535 white)
SET pen pattern to gray
Move pen to location (translate_x(x), bottom bound)
Draw line to location (translate_x(x), top bound)
Restore old foreground colour
Set pen pattern to black

15.2.1.2.10. void graph_device::draw_y_line(double y, long int brightness)

Backup current foreground colour
SET foreground colour to grey colour (shade determined by brightness 0 black, 65535 white)
SET pen pattern to gray
Move pen to location (left bound, translate_y(y))
Draw line to location (right bound, translate_y(y))
Restore old foreground colour
Set pen pattern to black

15.2.1.2.11. void graph_device::draw_axes()

SET foreground colour to red
Move pen to location (left bound, bottom bound)
Draw line to location (right bound, bottom bound) draw x-axis (bottom edge)
Move pen to location (left bound, bottom bound)
Draw line to location (left bound, top bound) draw y-axis (left edge)
SET foreground colour to yellow
SET pen pattern to gray
IF y=0 line is in visible graph area
    Move pen to location (left bound, translate_y(0))
    Draw line to location (right bound, translate_y(0)) draw y=0 line
IF x=0 line is in visible graph area
    Move pen to location (translate_x(0), bottom bound)
    Draw line to location (translate_x(0), top bound) draw x=0 line
SET foreground colour to black
SET pen pattern to black

15.2.1.2.12. void graph_device::vertical_ticks()

SET foreground drawing colour to red.
SET textsize to ‘scaleSize’ (10)
SET ‘y’ to bottom bound
SET ‘y_incrementer’ to inter-tick pixel distance on y-axis=
((vert_scale.MajScale/vert_range.range())/(vert_scale.MinTicks+1))*bound.height;
SET ‘y_val’ to vert_range.Min (Used to track numbering on axis)
SET ‘count’ to number of minor ticks between each major tick on y-axis
DO

Move pen to location (left bound, ‘y’)
Draw line to location (left bound-major tick size, ‘y’) Draws major tick
Move pen to location left of tick mark.
Place y_val in string
Set foreground colour to black
Draw y_val string at pen location Draw scale numbering
Set foreground colour to red
IF last major tick mark of axis reached
Call draw_y_line to draw major tick horizontal grid line across graph at end of axis.
BREAK out of DO loop
DO (Draws all minor ticks between each major tick)
decrease y by y_incrementer
Move pen to location (left bound, ‘y’)
Draw line to location (left bound-minor tick size, ‘y’) Draws minor tick
Draw minor tick horizontal grid line across graph at y location=
y_val+(count)*vert_scale.MajScale/(vert_scale.MinTicks+1)
WHILE count is not zero, decrease count by 1

reSET count to number of minor ticks between each major tick on y-axis
Call draw_y_line to draw major tick horizontal grid line at current y_val location.
Increase y_val by vert_scale.MajScale
FOREVER
SET foreground colour to black

15.2.1.2.13. void graph_device::horizontal_ticks()

SET foreground drawing colour to red.
SET textsize to ‘scaleSize’ (10)
SET ‘x’ to left bound
SET ‘x_incrementer’ to inter-tick pixel distance on x-axis=
((horiz_scale.MajScale/horiz_range.range())/(horiz_scale.MinTicks+1))*bound.width
SET ‘x_val’ to horiz_range.Min (Used to track numbering on axis)
SET ‘count’ to number of minor ticks between each major tick on x-axis
DO
Move pen to location (‘x’,bottom bound)
Draw line to location (‘x’, bottom bound+major tick size) Draws major tick
Move pen to location under tick mark.
Place x_val in string
Set foreground colour to black
Draw x_val string at pen location Draw scale numbering
Set foreground colour to red
IF last major tick mark of axis reached
Call draw_x_line to draw major tick vertical grid line at end of axis.
BREAK out of DO loop
DO (Draws all minor ticks between each major tick)
Increase x by x_incrementer
Move pen to location (x, bottom bound)
Draw line to location (‘x’, bottom bound+minor tick size) Draws minor tick
Draw minor tick vertical grid line down graph at x location=
x_val+count*horiz_scale.MajScale/(horiz_scale.MinTicks+1)
WHILE count is not zero, decrease count by 1

reSET count to number of minor ticks between each major tick on x-axis
Call draw_x_line to draw major tick vertical grid line at current x_val location.
Increase x_val by horiz_scale.MajScale
FOREVER
SET foreground colour to black

15.2.1.2.14. void graph_device::PlaceCross(int Width)

Backup current foreground drawing colour.
Set foreground drawing colour to green.
Move pen half-Width pixel distance left and up window.
Draw line, moving pen Width pixels down and left, diagonally.
Move pen Width pixel distance left.

Draw line, moving pen Width pixels up and right, diagonally.
Move pen half-Width pixel distance left and down - returning pen to original position.
Restore former foreground drawing colour.

15.2.1.2.15. void graph_device::set_port(port_info newport)

SET port to newport
IF graphing window exists, Deallocate window pointer
Set up rectangle, windRect, to hold port co-ordinates/size
Allocate new colour window to mainPtr, using windRect.
Set graph port to window pointed to by mainPtr. (all drawing sent to graphing new window now)
SET right bound to port width minus right border
SET bottom bound to port height minus bottom border
SET bound width to right bound minus left bound
SET bound height to bottom bound minus top bound

15.2.1.2.16. void graph_device::set_border(border_info newborder)

SET border to newborder
SET top bound to top border.
SET left bound to left border.
SET right bound to width of port minus right border
SET bottom bound to height of port minus bottom border
SET bound width to right bound minus left bound
SET bound height to bottom bound - top bound

15.2.1.2.17. void graph_device::set_horiz_scales(scale_info newhoriz_scale)

15.2.1.2.18. void graph_device::set_vert_scales(scale_info newvert_scale)

15.2.1.2.19. void graph_device::set_horiz_range(range_info newhoriz_range)

15.2.1.2.20. void graph_device::set_vert_range(range_info newvert_range)

15.2.1.2.21. void graph_device::set_ticks(tick_info newticks)

Set appropriate data member to value of function parameter.

15.2.1.2.22. double graph_device::x_origin(double x)

RETURN left bound pixel location + x

15.2.1.2.23. double graph_device::y_origin(double y)

RETURN bottom bound pixel location - y

15.2.1.3. Friend Member Function Pseudo-code

None

15.2.1.4. Static Member Function Pseudo-code

None

16. Code File: “validator.c++”

16.1. Non-Class Function Pseudo-code

none

16.2. Class Definitions

16.2.1. Class “validator”

16.2.1.1. Private Member Function Pseudo-code

16.2.1.1.1. status validator::verify(const string_class name_string, string_class dependents)

```
Define list to hold non_reserved_names
Define name_object ‘equation’ to hold current name_string.
SET *equation_list to point to connected_calculator equation_list
IF equation name_object not found in equation_list
    RETURN SUCCESS (name_string references variable/constant or undefined)
Get list of non_reserved_names in name_string using postprocess  in connected_preprocessor
IF non_reserved_name list is empty
    RETURN SUCCESS (contains no references to constants/variables/equations)
RECURSIVE SECTION:
DO
    Get current transverse (equation) node from equation_list - store in string_node
    IF string_node name is found in ‘dependents’
            append string_node name (single space separation) to dependents string
            SET error_trace to dependents string
            RETURN ERROR
    ELSE
            Make backup of dependents string
            append string_node name (single space separation) to dependents string
            Call VERIFY (this function) with string_node name and current dependents string params.
            IF  recursive VERIFY call returns error
                    RETURN ERROR (circular definition found - back propagate the error)
            ELSE
                    Restore dependents to backup state.
WHILE end of non_reserved_names list not reached
RETURN SUCCESS (no circular definitions found)
```

16.2.1.2. Public Member Function Pseudo-code

16.2.1.2.1. validator::validator(calc_preprocessor *preprocessor)

```
SET connected_processor to preprocessor.
```

16.2.1.2.2. status validator::validate(calculator *connect_calculator)

```
SET connected_calculator to connect_calculator
SET *equation_list to point to connected_calculator equation_list
Define single name_object called equation_node.
Initialise error_trace to empty “”.
Reset transverse pointer of equation_list to head of list.
IF no equations in equation_list
    RETURN SUCCESS
DO
    Get current transverse (equation) node from equation_list
    IF call to verify (dependents=””) on equation string stored in transverse node returns ERROR
            RETURN ERROR
WHILE end of equation_list not reached
Re-initialise error_trace to empty “”.
RETURN SUCCESS
```

16.2.1.3. Friend Member Function Pseudo-code

None

16.2.1.4. Static Member Function Pseudo-code

None

17. Code File: “preprocessorTypes.c++”

17.1. Non-Class Function Pseudo-code

none

17.2. Class Definitions

17.2.1. Class “user_label”

17.2.1.1. Private Member Function Pseudo-code

None

17.2.1.2. Public Member Function Pseudo-code

17.2.1.2.1. user_label::user_label()

Initialise both data members to “”

17.2.1.2.2. user_label::user_label(string_class i_string, string_class c_string)

Initialise input_string to i_string; initialise calc_string to c_string

17.2.1.3. Friend Member Function Pseudo-code

17.2.1.3.1. ostream& operator<<(ostream& output_stream, const user_label label)

output_stream << label.input_string << " -> " << label.calc_string

17.2.1.4. Static Member Function Pseudo-code

None

18. Code File: “calc_preprocessor.c++”

18.1. Non-Class Function Pseudo-code

none

18.2. Class Definitions

18.2.1. Class “calc_preprocessor”

18.2.1.1. Private Member Function Pseudo-code

18.2.1.1.1. status calc_preprocessor::set_input(string_class input_string)

IF input_stream already allocated - call reset_input()
Allocate new input_char_array
Extract char array out of input_string - store in input_char_array
Allocate new input_stream pointing to first character of input_char_array.
RETURN SUCCESS

18.2.1.1.2. status calc_preprocessor::reset_input()

IF input stream not already allocated:
    delete input_stream & input_char_array, resetting both to NULL.
    RETURN SUCCESS
ELSE
    Output warning to standard output that attempt was made to reset input before input was set.
    RETURN ERROR

18.2.1.2. Public Member Function Pseudo-code

18.2.1.2.1. calc_preprocessor::calc_preprocessor(const user_label *input_mappings, const token_name *token_mappings, int length)

SET input_stream & input_char_array to NULL.
SET array_length to length.
Allocate correlator as array of token_name[array_length]

BUILD CORRELATOR ARRAY:
FOR all elements in correlator array (up to array_length) [i]
    FOR all elements in token_mappings array [j]
        IF calc_string of input_mappings element [i] equals name of token_mappings element [j]
            SET name of correlator element [i] to input_string in input_mappings element [i]
            SET token of correlator element[i] to token in token_mappings element [j]
    ENDFOR
ENDFOR

BUILD INVERSE_CORRELATOR ARRAY:

BUILD INVERSE_CORRELATOR ARRAY:
FOR all elements in correlator array [k]
Use token char of element [k] in correlator array as index for inverse_correlator, and store in this location the name in element [k] of correlator.
ENDFOR

18.2.1.2.2. calc_preprocessor::~calc_preprocessor()

Delete correlator array

18.2.1.2.3. void calc_preprocessor::preprocess (string_class input_string, string_class &output_string)

Initialise output_string to "".
Call set_input(input_string) to set up new input_stream to point to copy of input_string.
WHILE characters remaining in input_stream
WHILE chars present in input_stream get a single char - continue looping until non-alpha found
append char got from input_stream to output_string
BREAK out of WHILE loop if end of stream reached
place last character read (non-alpha) back into input_stream
Read entire char string (delimited by non alpha-numeric char) into 'string_class single_name'.
LOOK UP single_name in correlator array:
FOR all elements in correlator array [i]
IF element [i] in correlator array matches single_name
SET single_name to token of element [i] in correlator array
BREAK out of FOR LOOP
ENDFOR
append single_name to output_string.
ENDWHILE
Call reset_input to delete input_stream and input_char_array.

18.2.1.2.4. void calc_preprocessor::preprocess (string_class input_string, string_class &output_string, ulist<string_object> *unidentified)

Initialise output_string to "".
Call set_input(input_string) to set up new input_stream to point to copy of input_string.
WHILE characters remaining in input_stream
WHILE chars present in input_stream get a single char - continue looping until non-alpha found
Append char got from input_stream to output_string
ENDWHILE
BREAK out of WHILE loop if end of stream reached
place last character read (non-alpha) back into input_stream
Read entire char string (delimited by non alpha-numeric char) into 'string_class single_name'.
LOOK UP single_name in correlator array:
SET found flag FALSE.
FOR all elements in correlator array [i]
IF element [i] in correlator array matches single_name
SET single_name to token of element [i] in correlator array
SET found flag TRUE
BREAK out of FOR LOOP
ENDFOR
IF single_name not FOUND in correlator array
Store single_name in a string_object and add this to unidentified list.
append single_name to output_string.
ENDWHILE
Call reset_input to delete input_stream and input_char_array.

18.2.1.2.5. void calc_preprocessor::postprocess (const string_class &input_string, string_class &postprocessed_string)

Call set_input(input_string) to set up new input_stream to point to copy of input_string.
Declare char array output_string
WHILE end of input_stream not reached
Read in single character from input_stream into 'input_char'
IF end of input_stream
BREAK out of WHILE LOOP
IF element 'input_char' (casted to int) of inverse_correlator is not empty string
Extract char array stored in this element.

Concatenate this char array to output_string
ELSE
Concatenate ‘input_char’ to output_string
ENDWHILE
Call reset_input to delete input_stream and input_char_array.
SET postprocessed string to output_string.

18.2.1.3. Friend Member Function Pseudo-code

None

18.2.1.4. Static Member Function Pseudo-code

None

19. Code File: “ulist.c++”

19.1. Non-Class Function Pseudo-code

none

19.2. Class Definitions

19.2.1. Class “ulist<node>”

19.2.1.1. Private Member Function Pseudo-code

19.2.1.1.1. node* ulist<node>::search(node searchitem, ordering method)

```
reset current to head of list
IF ordering is on index
    WHILE end of list not reached and current index != searchitem index
            SET previous to current
            progress current to next node in list
    ENDWHILE
ELSE
    WHILE end of list not reached and current data != searchitem data
            SET previous to current
            progress current to next node in list
    ENDWHILE
return previous pointer
```

19.2.1.1.2. void ulist<node>::find_neighbours(node *target, node* &preceding, node* &proceding, ordering comparisontype, orderproperty direction)

```
reset current to head of list
IF direction is ASCENDING
    make the relational comparison 'operation' = LARGER
else
    make the relational comparison 'operation' = SMALLER
IF ordering is on index
    WHILE end of list not reached and current index ('operation') than target index
            SET previous to current
            progress current to next node in list
    ENDWHILE
ELSE
    WHILE end of list not reached and current data ('operation') than target data
            SET previous to current
            progress current to next node in list
    ENDWHILE
IF insertion not at head of list
    SET 'proceding' to equal the next_object_pointer of *preceding.
ELSE
    SET 'proceding' to head.
```

19.2.1.2. Public Member Function Pseudo-code

19.2.1.2.1. ulist<node>::ulist()

```
Initialise head,tail,current,transverse pointers to NULL
```

19.2.1.2.2. ulist<node>::ulist(ulist &original)

```
Initialise head,tail,current,transverse pointers to NULL
copy using *this=original
```

19.2.1.2.3. void ulist<node>::clearlist()

```
IF list not empty
    WHILE list is non-empty
            Use remove_transverse to remove a single node
    ENDWHILE
```

19.2.1.2.4. ulist<node>::~ulist(void)

```
Call ulist<node>::clearlist()
```

19.2.1.2.5. status ulist<node>::add(node newitem, ordering order, orderproperty direction)

19.2.1.2.5. status ulist<node>::add(node newitem, ordering order, orderproperty direction)

```
Declare three node pointers - previous, following and newnode
Dynamically allocate a new node
If out of memory
    write error message to cout
    RETURN ERROR
SET *newnode to newitem (uses node copy constructor)
SET newnode pointer field to NULL
IF list empty
    SET head,tail, current and transverse to newnode address.
    RETURN SUCCESS
ELSE
    SET current to head of list (correct insertion point needs to be found)

Call find_neighbours( newnode, previous, following, order, direction);

IF index ordering required (order=COUNTER)
    IF newnode is non-unique on index when compared to preceding or proceding neighbouring nodes
            RETURN ERROR
ELSE IF data ordering required (order=DATA)
    IF newnode is non-unique on data when compared to preceding or proceding neighbouring nodes
            RETURN ERROR

IF addition required at end of list (following = NULL)
    SET the 'previous' node to point new node
    SET current and tail to address of new node
ELSE
    IF addition required before head of list (previous = NULL)
            make new node object point to head of list
            make new node the head of the list
            SET current to equal head
    ELSE
            Addition in middle of list required:
            Set the 'previous' node to new node.
            Set the new node to point to 'following' node.
            SET current to newnode address
RETURN SUCCESS
```

19.2.1.2.6. status ulist<node>::remove(node item, ordering method)

```
Call search function to search for removeitem using ordering method.
IF node not found  (current==NULL)
    RETURN ERROR
IF removal from head of list required (previous==NULL)
    progress current to next node (node after node to be removed)
    IF transverse pointer points to head
            progress transverse pointer to point to next node in list
    IF there is one node in the list
            SET tail to NULL
    DELETE node pointed to by head
    SET head to new head of list = current
ELSE removal in middle or end of list required
    SET *previous next_word_pointer to *current next_word_pointer
    IF tail / last node being removed from list
            SET tail to 'previous' node address
    IF the node pointed to by transverse is being removed
            IF current != tail of the list
                    SET transverse to point to next node after node to be deleted
            ELSE
                    SET transverse to equal 'previous'.
    DELETE node pointed to by current
    SET current to equal 'previous'.
RETURN SUCCESS
```

19.2.1.2.7. status ulist<node>::update(node reference, node new_info, ordering update_field, ordering structure_order, orderproperty direction)

Declare two size comparators, left_size=right_size=SMALLER
Search for reference node on 'structure_order' field, call previous=search(reference, structure_order)
IF node not found in ulist
    RETURN ERROR
Declare a pointer, 'following', pointing to the node following 'reference' node in ulist.
Declare two ordering comparators, left_comparator & right_comparator.
IF ASCENDING ordering required
    new_info must be LARGER than preceding node to update node, left_comparator=LARGER
    new_info must be SMALLER than proceding node to update node, right_comparator=SMALLER
ELSE IF DESCENDING ordering required
    new_info must be SMALLER than preceding node to update node left_comparator=SMALLER
    new_info must be LARGER than proceding node to update node right_comparator=LARGER
Now decide whether node being updated needs to be moved to maintain ordering.
Declare flag: move_original to hold the decision.
IF list is ordered on index (COUNTER)
    SET left_size & right_size values by comparing on index preceding & proceding nodes to new node
    IF preceding & proceding nodes are not ordered on index according to left_comparator & right_comparator values (when compared to left_size & right_size).
        SET move_original_node=1
    ELSE
        SET move_original_node=0
ELSE IF list is ordered on data (DATA)
    SET left_size & right_size values by comparing on data preceding & proceding nodes to new node
    IF preceding & proceding nodes are not ordered on data according to left_comparator & right_comparator values (when compared to left_size & right_size).
        SET move_original_node=1
    ELSE
        SET move_original_node=0

IF move is required
    IF updating index (COUNTER) field
        Grab data field from node being updated in list and store in reference node
    ELSE IF updating data field
        Grab index field from node being updated in list and store in reference node
    remove the original node in the list, call: remove(reference, structure_order)
    add new node to list in correct position, call: add(new_info, structure_order, direction)
ELSE no move required
    IF preceding or proceding node has equal structure_order field value
        RETURN ERROR
    ELSE
        IF updating index (COUNTER) field
            Copy index value of new_info into update node in list
        ELSE IF updating data field
            Copy data value of new_info into update node in list
RETURN SUCCESS

19.2.1.2.8. status ulist<node>::maintain(node new_info, ordering structure_order, orderproperty direction)

Search for new_info node on structure_order field.
IF not found in ulist
    add new_info node to ulist, call: add(new_info, structure_order, direction)
ELSE
    IF ordering of ulist is on index (COUNTER) field
        SET update_field to DATA
    ELSE IF ordering of list is on DATA field
        SET update_field to COUNTER
Update new_info node in ulist, searching on structure_order field, updating !structure_order field, call: update (*current, new_info, update_field, structure_order, direction).

19.2.1.2.9. status ulist<node>::add_to_end(node newitem)

```
Allocate memory for new node.
IF out of memory
    Output error message
    RETURN ERROR
IF list is empty
    IF index is not set in newitem (index=node::undefined_index)
            set newitem index to first index
    Set head, tail and transverse to point to newnode
ELSE
    IF index is not set in newitem (index=node::undefined_index)
            SET newitem index by calling set_to_next_index on tail index
    SET *tail next_object_pointer to new node pointer
    SET tail to new node address
RETURN SUCCESS
```

19.2.1.2.10. status ulist<node>::search_node(node &search_item, ordering method)

```
Call private member function:  node* previous = search(search_item, method);
IF node not found
    RETURN ERROR
ELSE
    Copy COUNTER & DATA fields from found node into search_item
    RETURN SUCCESS
```

19.2.1.2.11. status ulist<node>::reset_transverse()

```
IF list is empty
    RETURN ERROR
ELSE
    SET transverse to head of list
    RETURN SUCCESS
```

19.2.1.2.12. status ulist<node>::get_transverse(node &gotitem)

```
IF list is not empty
    use overloaded= operator to copy ... gotitem = *transverse
    RETURN SUCCESS
ELSE
    RETURN ERROR
```

19.2.1.2.13. status ulist<node>::progress_transverse()

```
IF list is not empty and transverse is not at end of list
    set transverse to point to the next node in the ulist
    RETURN SUCCESS
ELSE
    RETURN ERROR
```

19.2.1.2.14. status ulist<node>::remove_transverse()

```
IF list is not empty
    initialise current=head, preceding=NULL
    WHILE  current does not point to the node pointed to by transverse
            SET preceding to current
            progress current to point to next node in the list
    IF removal from head required
            progress transverse to point to next node in the list
            DELETE node pointed to by head
            SET head to equal current to equal transverse
            IF list is empty
                    SET tail = NULL
    ELSE removal from anywhere except head required
            SET *preceding next_object_pointer to equal *transverse next_object_pointer
            IF removing from tail
                    SET tail to preceding
            DELETE node pointed to by current;
            IF tail has been removed
```

SET transverse to preceding (the new tail of the list)
ELSE
SET transverse to point to that node following *preceding in the list
SET current to head of list
RETURN SUCCESS - node removed
RETURN ERROR - list empty

19.2.1.2.15. node* ulist<node>::get_transverse_node_pointer()

RETURN tranverse pointer

19.2.1.2.16. ulist<node>& ulist<node>::operator=(ulist<node> &source)

Instantiate a single ‘working’ node
Call: clearlist to remove all nodes from ‘this’ list
IF the source list contains any nodes
DO
use source.get_transverse to load tranverse node into working word_object.
call this.add_to_end(working) to add copy of working node to list.
WHILE end of source list not reached
RETURN *this

19.2.1.3. Friend Member Function Pseudo-code

19.2.1.3.1. ostream& operator<<(ostream& output_stream, const ulist a)

IF not an empty list
SET current to head
DO
call current->print_node()
progress current to next node in list
If not at end of the list output a single space
WHILE (current not at end of list)
ELSE
Output 'This u-list is empty' message.
RETURN output_stream

19.2.1.4. Static Member Function Pseudo-code

none

19.3. Other definitions/declarations

19.3.1. Explicit ulist instantiations - one for each type of ulist required

```
template class ulist<name_object>;
template class ulist<string_object>;
template class ulist<calc_object>;
template class ulist<IO_map_object>;
template class ulist<data_set_obj>;
template class ulist<record_object>;
template class ulist<set_input_object>;
template class ulist<graph_spec_obj>;
```

19.3.2. Explicit instantiations of friend functions for each ulist required

```
ostream& operator<<(ostream& output_stream, ulist<name_object> a);
ostream& operator<<(ostream& output_stream, ulist<string_object> a);
ostream& operator<<(ostream& output_stream, ulist<calc_object> a);
ostream& operator<<(ostream& output_stream, ulist<IO_map_object> a);
ostream& operator<<(ostream& output_stream, ulist<data_set_obj> a);
ostream& operator<<(ostream& output_stream, ulist<record_object> a);
ostream& operator<<(ostream& output_stream, ulist<set_input_object> a);
ostream& operator<<(ostream& output_stream, ulist<graph_spec_obj> a);
```

20. Code File: “record_object.c++”

20.1. Non-Class Function Pseudo-code

none

20.2. Class Definitions

20.2.1. Class “record_object”

20.2.1.1. Private Member Function Pseudo-code

None

20.2.1.2. Public Member Function Pseudo-code

20.2.1.2.1. record_object::record_object()
20.2.1.2.2. record_object::record_object(string_class name_string)
20.2.1.2.3. record_object::record_object(string_class name_string, ulist<string_object> data_list)
20.2.1.2.4. record_object::record_object(record_object &original)
20.2.1.2.5. record_object& record_object::operator=(record_object &original)
20.2.1.2.6. string_class record_object::get_index()
20.2.1.2.7. status record_object::set_index(const string_class setting)
20.2.1.2.8. void record_object::set_to_first_index()
20.2.1.2.9. void record_object::set_to_next_index(const string_class ref)
20.2.1.2.10. compare record_object::compare_index(const string_class index1, const string_class index2)
20.2.1.2.11. ulist<string_object> record_object::get_data()
20.2.1.2.12. status record_object::set_data(ulist<string_object> setting)
20.2.1.2.13. compare record_object::compare_data(ulist<string_object> name1, ulist<string_object> name2)
20.2.1.2.14. void record_object::set_pointer_to(record_object *p)
20.2.1.2.15. record_object* record_object::get_pointer()
20.2.1.2.16. void record_object::print_node()

See pseudo-code for equivalent member functions of calc_object class.

20.2.1.3. Friend Member Function Pseudo-code

20.2.1.3.1. ostream& operator<<(ostream& output_stream, record_object a)

See pseudo-code for equivalent member functions of calc_object class.

20.2.1.4. Static Member Function Pseudo-code

None

21. Code File: “data_set_obj.c++”

21.1. Non-Class Function Pseudo-code

none

21.2. Class Definitions

21.2.1. Class “data_set_obj”

21.2.1.1. Private Member Function Pseudo-code

None

21.2.1.2. Public Member Function Pseudo-code

21.2.1.2.1. data_set_obj::data_set_obj()
21.2.1.2.2. data_set_obj::data_set_obj(string_class name_string)
21.2.1.2.3. data_set_obj::data_set_obj(string_class name_string, data_set data_list)
21.2.1.2.4. data_set_obj::data_set_obj(data_set_obj &original)
21.2.1.2.5. data_set_obj& data_set_obj::operator=(data_set_obj &original)
21.2.1.2.6. string_class data_set_obj::get_index()
21.2.1.2.7. status data_set_obj::set_index(const string_class setting)
21.2.1.2.8. void data_set_obj::set_to_first_index()
21.2.1.2.9. void data_set_obj::set_to_next_index(const string_class ref)
21.2.1.2.10. compare data_set_obj::compare_index(const string_class index1, const string_class index2)
21.2.1.2.11. data_set data_set_obj::get_data()
21.2.1.2.12. status data_set_obj::set_data(data_set setting)
21.2.1.2.13. compare data_set_obj::compare_data(const data_set name1, const data_set name2)
21.2.1.2.14. void data_set_obj::set_pointer_to(data_set_obj *p)
21.2.1.2.15. data_set_obj* data_set_obj::get_pointer()

21.2.1.2.15. data_set_obj* data_set_obj::get_pointer()
21.2.1.2.16. void data_set_obj::print_node()
See pseudo-code for equivalent member functions of calc_object.

21.2.1.2.17. status data_set_obj::get_element_assoc(complex &result, const int index, const string_class field)
21.2.1.2.18. status data_set_obj::set_element_assoc(const complex new_value, const int index, const string_class field)
21.2.1.2.19. status data_set_obj::get_element(complex &result, const int index, const int field_index)
21.2.1.2.20. status data_set_obj::set_element(const complex new_value, const int index, const int field_index)
21.2.1.2.21. int data_set_obj::get_length()
21.2.1.2.22. int data_set_obj::get_width()
21.2.1.2.23. void data_set_obj::get_fields(string_class *storage)
21.2.1.2.24. status data_set_obj::clear_data_array()
See Pseudo-code for equivalent functions in data_set class.

21.2.1.2.25. status data_set_obj::load_real_input(set_input &input, string_class field)
Dynamically allocate string_class array 'fields' of size determined by 'width' of data_set 'data'
Call data.get_fields(fields) to convert 'data' ulist of string_class to array of string_class
Get integer index of 'field' in data_set 'data'.
IF 'field' not found in data_set 'data'
RETURN ERROR
Extract individual data members of 'input' , storing in index_low, index_high, start_value & increment
IF index_low OR index_high out of range of data_set array
RETURN ERROR
FOR LOOP : iterate through all elements from index_low to index_high
IF setting iterator element returns ERROR
RETURN ERROR
SET start_value to start_value+incrementer
RETURN SUCCESS

21.2.1.3. Friend Member Function Pseudo-code
21.2.1.3.1. ostream& operator<<(ostream& output_stream, data_set_obj a)
See pseudo-code for equivalent member function of calc_object class.

21.2.1.4. Static Member Function Pseudo-code
None

22. Code File: "set_input_object.c++"

22.1. Non-Class Function Pseudo-code

none

22.2. Class Definitions

22.2.1. Class "set_input_object"

22.2.1.1. Private Member Function Pseudo-code
None

22.2.1.2. Public Member Function Pseudo-code
22.2.1.2.1. set_input_object::set_input_object()
22.2.1.2.2. set_input_object::set_input_object(string_class name_string)
22.2.1.2.3. set_input_object::set_input_object(string_class name_string, set_input data)
22.2.1.2.4. set_input_object::set_input_object(set_input_object &original)
22.2.1.2.5. set_input_object& set_input_object::operator=(set_input_object &original)
22.2.1.2.6. string_class set_input_object::get_index()
22.2.1.2.7. status set_input_object::set_index(const string_class setting)
22.2.1.2.8. void set_input_object::set_to_first_index()
22.2.1.2.9. void set_input_object::set_to_next_index(const string_class ref)
22.2.1.2.10. compare set_input_object::compare_index(const string_class index1, const string_class index2)
22.2.1.2.11. set_input set_input_object::get_data()
22.2.1.2.12. status set_input_object::set_data(set_input setting)

22.2.1.2.12. status set_input_object::set_data(set_input setting)
22.2.1.2.13. compare set_input_object::compare_data(const set_input name1, const set_input name2)
22.2.1.2.14. void set_input_object::set_pointer_to(set_input_object *p)
22.2.1.2.15. set_input_object* set_input_object::get_pointer()
22.2.1.2.16. void set_input_object::print_node()
See pseudo-code for equivalent member functions of calc_object.

22.2.1.3. Friend Member Function Pseudo-code
22.2.1.3.1. ostream& operator<<(ostream& output_stream, set_input_object a)
See pseudo-code for equivalent member function of calc_object

22.2.1.4. Static Member Function Pseudo-code
None

23. Code File: “IO_map_object.c++”

23.1. Non-Class Function Pseudo-code

none

23.2. Class Definitions

23.2.1. Class “IO_map_object”

23.2.1.1. Private Member Function Pseudo-code
None

23.2.1.2. Public Member Function Pseudo-code
23.2.1.2.1. IO_map_object::IO_map_object()
23.2.1.2.2. IO_map_object::IO_map_object(string_class map_name)
23.2.1.2.3. IO_map_object::IO_map_object(string_class map_name, IO_map new_map)
23.2.1.2.4. IO_map_object::IO_map_object(IO_map_object &original)
23.2.1.2.5. IO_map_object& IO_map_object::operator=(IO_map_object &original)
23.2.1.2.6. string_class IO_map_object::get_index()
23.2.1.2.7. status IO_map_object::set_index(const string_class setting)
23.2.1.2.8. void IO_map_object::set_to_first_index()
23.2.1.2.9. void IO_map_object::set_to_next_index(const string_class ref)
23.2.1.2.10. compare IO_map_object::compare_index(const string_class index1, const string_class index2)
23.2.1.2.11. IO_map IO_map_object::get_data()
23.2.1.2.12. status IO_map_object::set_data(IO_map setting)
23.2.1.2.13. compare IO_map_object::compare_data(const IO_map name1, const IO_map name2)
23.2.1.2.14. void IO_map_object::set_pointer_to(IO_map_object *p)
23.2.1.2.15. IO_map_object* IO_map_object::get_pointer()
23.2.1.2.16. void IO_map_object::print_node()
See pseudo-code for equivalent member functions of calcobject.

23.2.1.2.17. void IO_map_object::get_input_fields(string_class *stored_input_fields)
23.2.1.2.18. int IO_map_object::get_number_of_input_fields()
23.2.1.2.19. void IO_map_object::get_output_fields(string_class *stored_output_fields)
23.2.1.2.20. int IO_map_object::get_number_of_output_fields()
All call equivalent member functions of IO_map - thus identical operation.

23.2.1.3. Friend Member Function Pseudo-code
23.2.1.3.1. ostream& operator<<(ostream& output_stream, IO_map_object a)
See pseudo-code for equivalent member functions of calcobject.

23.2.1.4. Static Member Function Pseudo-code
None

24. Code File: “graph_spec_obj.c++”

24.1. Non-Class Function Pseudo-code

none

24.2. Class Definitions

24.2.1. Class “graph_spec_obj”

24.2.1.1. Private Member Function Pseudo-code

None

24.2.1.2. Public Member Function Pseudo-code

24.2.1.2.1. graph_spec_obj::graph_spec_obj()
24.2.1.2.2. graph_spec_obj::graph_spec_obj(string_class graph_name)
24.2.1.2.3. graph_spec_obj::graph_spec_obj(string_class graph_name, graph_spec new_graph_spec)
24.2.1.2.4. graph_spec_obj::graph_spec_obj(graph_spec_obj &original)
24.2.1.2.5. graph_spec_obj& graph_spec_obj::operator=(graph_spec_obj &original)
24.2.1.2.6. string_class graph_spec_obj::get_index()
24.2.1.2.7. status graph_spec_obj::set_index(const string_class setting)
24.2.1.2.8. void graph_spec_obj::set_to_first_index()
24.2.1.2.9. void graph_spec_obj::set_to_next_index(const string_class ref)
24.2.1.2.10. compare graph_spec_obj::compare_index(const string_class index1,const string_class index2)
24.2.1.2.11. graph_spec graph_spec_obj::get_data()
24.2.1.2.12. status graph_spec_obj::set_data(graph_spec setting)
24.2.1.2.13. compare graph_spec_obj::compare_data(const graph_spec name1, const graph_spec name2)
24.2.1.2.14. void graph_spec_obj::set_pointer_to(graph_spec_obj *p)
24.2.1.2.15. graph_spec_obj* graph_spec_obj::get_pointer()
24.2.1.2.16. void graph_spec_obj::print_node()

See pseudo-code for equivalent member functions of calc_object class.

24.2.1.3. Friend Member Function Pseudo-code

24.2.1.3.1. ostream& operator<<(ostream& output_stream, graph_spec_obj a)

See pseudo-code for equivalent member function of calc_object class.

24.2.1.4. Static Member Function Pseudo-code

None

25. Code File: “name_object.c++”

25.1. Non-Class Function Pseudo-code

none

25.2. Class Definitions

25.2.1. Class “name_object”

25.2.1.1. Private Member Function Pseudo-code

None

25.2.1.2. Public Member Function Pseudo-code

25.2.1.2.1. name_object::name_object()

SET next_name_object to NULL, SET name to UNDEFINED_INDEX

25.2.1.2.2. name_object::name_object(const string_class string, const complex_container complex_field)

SET name to string, SET data to complex_field, SET next_name_object to NULL

25.2.1.2.3. name_object::name_object(const char *string, const complex_container complex_field)

SET name to string, SET data to complex_field, SET next_name_object to NULL

25.2.1.2.4. name_object::name_object(const name_object &original)

Copy all data members from original to ‘this’.

25.2.1.2.5. name_object& name_object::operator=(name_object &source)

25.2.1.2.5. name_object& name_object::operator=(name_object &source)
Copy all data members from source to 'this'.

25.2.1.2.6. string_class name_object::get_index()
RETURN name

25.2.1.2.7. status name_object::set_index(const string_class setting)
SET name to setting
RETURN SUCCESS

25.2.1.2.8. void name_object::set_to_first_index()
SET name to "a"

25.2.1.2.9. void name_object::set_to_next_index(const string_class ref)
SET name to ref appended by "a"

25.2.1.2.10. compare name_object::compare_index(const string_class index1, const string_class index2)
```
IF index1>index2
    RETURN LARGER
ELSE
    IF index1 = index2
            RETURN EQUAL
    ELSE
            RETURN SMALLER
```

25.2.1.2.11. complex_container name_object::get_data()
RETURN data

25.2.1.2.12. status name_object::set_data(const complex_container setting)
SET data to setting
RETURN SUCCESS

25.2.1.2.13. compare name_object::compare_data(const complex_container name1, const complex_container name2)
Call: compare_containers (name1, name2) and RETURN result

25.2.1.2.14. void name_object::set_pointer_to(name_object *p)
SET next_name_object to p

25.2.1.2.15. name_object* name_object::get_pointer()
RETURN next_name_object

25.2.1.2.16. void name_object::print_node()
cout << name << "-> " << data;

25.2.1.2.17. complex name_object::get_complex()
RETURN data.complex

25.2.1.2.18. void name_object::set_complex(const complex new_complex)
SET data.complex to new_complex

25.2.1.2.19. macro_type name_object::get_indicator()
RETURN data.indicator

25.2.1.2.20. void name_object::set_indicator(const macro_type new_indicator)
SET data.indicator to new_indicator

25.2.1.2.21. string_class name_object::get_equation()
RETURN data.equation

25.2.1.2.22. void name_object::set_equation(const string_class new_equation)
SET data.equation to new_equation

25.2.1.3. Friend Member Function Pseudo-code

25.2.1.3.1. ostream& operator<<(ostream& output_stream, const name_object a)

output_stream << name << "-> " << data;

25.2.1.4. Static Member Function Pseudo-code

None

25.2.1.5. Static Data Members

25.2.1.5.1. int string_object::undefined_index=""

Defines value index takes when object is created but no index is specified.

26. Code File: "calcobject.c++"

26.1. Non-Class Function Pseudo-code

none

26.2. Class Definitions

26.2.1. Class "calc_object"

26.2.1.1. Private Member Function Pseudo-code

None

26.2.1.2. Public Member Function Pseudo-code

26.2.1.2.1. calc_object::calc_object()
26.2.1.2.2. calc_object::calc_object(const string_class name)
26.2.1.2.3. calc_object::calc_object(const calc_object &original)
26.2.1.2.4. calc_object& calc_object::operator=(calc_object &source)
26.2.1.2.5. string_class calc_object::get_index()
26.2.1.2.6. status calc_object::set_index(const string_class setting)
26.2.1.2.7. void calc_object::set_to_first_index()
26.2.1.2.8. void calc_object::set_to_next_index(const string_class ref)
26.2.1.2.9. compare calc_object::compare_index(const string_class index1, const string_class index2)
26.2.1.2.10. calculator calc_object::get_data()
26.2.1.2.11. status calc_object::set_data(const calculator setting)

See pseudo-code for equivalent member functions of name_object class.

26.2.1.2.12. compare calc_object::compare_data(const calculator calc1, const calculator calc2)

RETURN SMALLER

26.2.1.2.13. void calc_object::set_pointer_to(calc_object *p)

SET next_calc_object to p

26.2.1.2.14. calc_object* calc_object::get_pointer()

RETURN next_name_object

26.2.1.2.15. void calc_object::print_node()

Output calculator name, data.get_var_list() and data.get_equation_list()

26.2.1.2.16. complex calc_object::evaluate(string_class input_string)
26.2.1.2.17. string_class calc_object::flush_errors()
26.2.1.2.18. ostream& calc_object::peek_errors(ostream& output_stream)
26.2.1.2.19. int calc_object::get_number_of_errors()
26.2.1.2.20. void calc_object::all_clear()
26.2.1.2.21. status calc_object::clear_single_memory(string_class name)
26.2.1.2.22. void calc_object::auto_verify_off()
26.2.1.2.23. status calc_object::auto_verify_on()
26.2.1.2.24. void calc_object::set_validator(validator *checker)

See Pseudo-code for equivalent functions in calculator class.

26.2.1.3. Friend Member Function Pseudo-code

26.2.1.3.1. ostream& operator<<(ostream& output_stream, calc_object a)

Use Overloaded output operators to output name and calculator info.

26.2.1.4. Static Member Function Pseudo-code

None

27. Code File: "stringobject.c++"

27. Code File: “stringobject.c++”

27.1. Non-Class Function Pseudo-code

none

27.2. Class Definitions

27.2.1. Class “string_object”

27.2.1.1. Private Member Function Pseudo-code

None

27.2.1.2. Public Member Function Pseudo-code

27.2.1.2.1. string_object::string_object()

SET next_string_object to NULL, SET index to UNDEFINED_INDEX

27.2.1.2.2. string_object::string_object(const string_class original)

SET next_string_object to NULL, SET index to UNDEFINED_INDEX, SET string to original

27.2.1.2.3. string_object::string_object(const string_object &original)

Copy all data members from original to ‘this’.

27.2.1.2.4. string_object& string_object::operator=(string_object &source)

Copy all data members from source to ‘this’.
RETURN *this

27.2.1.2.5. int string_object::get_index()

RETURN index

27.2.1.2.6. status string_object::set_index(const int setting)

SET index to setting if setting>=0
RETURN SUCCESS if setting>=0
RETURN ERROR if setting<0

27.2.1.2.7. void string_object::set_to_first_index()

SET index to 1

27.2.1.2.8. status string_object::set_to_next_index(const int ref)

SET index to ref+1
RETURN SUCCESS

27.2.1.2.9. compare string_object::compare_index(const int index1, const int index2)

```
IF index1>index2
    RETURN LARGER
ELSE
    IF index1==index2
            RETURN EQUAL
    ELSE
            RETURN SMALLER
```

27.2.1.2.10. string_class string_object::get_data()

RETURN string

27.2.1.2.11. status string_object::set_data(const string_class set_string)

SET string to set_string
RETURN SUCCESS

27.2.1.2.12. compare string_object::compare_data(const string_class data1, const string_class data2)

```
IF data1>data2
    RETURN LARGER
ELSE
    IF data1==data2
            RETURN EQUAL
    ELSE
            RETURN SMALLER
```

27.2.1.2.13. void string_object::set_pointer_to(string_object *p)

SET next_string_object to p

27.2.1.2.14. string_object* string_object::get_pointer()

RETURN next_string_object

27.2.1.2.15. void string_object::print_node()

cout << index << ' ' << string

27.2.1.3. Friend Member Function Pseudo-code

27.2.1.3.1. ostream& operator<<(ostream& output_stream, const string_object a)

output_stream << a.index << ' ' << a.string

27.2.1.4. Static Member Function Pseudo-code

None

27.2.1.5. Static Data Members

27.2.1.5.1. int string_object::undefined_index=0

Defines value index takes when object is created but no index is specified.

28. Code File: “iadditionalmath.c++”

28.1. Non-Class Function Pseudo-code

28.1.1. double log2(const double value)
return log(value)/log(2)

28.1.2. double logx(const double value, const double x)
return log(value)/log(base)

28.1.3. double asinh(const double x)
return log(x+sqrt(1+pow(x,2)))

28.1.4. double acosh(const double x)
return log(x+sqrt(pow(x,2)-1))

28.1.5. double atanh(const double x)
return 0.5*log((1+x)/(1-x))

28.1.6. int factorial(const int x)
iteratively compute factorial and return result

29. Code File: “complex functions.c++”

29.1. Non-Class Function Pseudo-code

29.1.1. double magnitude(const complex a)
RETURN sqrt(a.re^2 + a.im^2)

29.1.2. double arg(const complex a)
Use C library function atan2 to calculate tan^-1(a.im/a.re): result in range -π -> +π
Normalise to 0->2π range and RETURN result

29.1.3. complex sqrt_comp(const complex a)
Use complex power operator to raise ‘a’ to power one half, RETURN result

29.1.4. complex cbrt(const complex a)
Use complex power operator to raise ‘a’ to power one third, RETURN result

29.1.5. complex polar_rect(const double mag, const double arg)
Return complex number : re=mag*cos(arg)
im=mag*sin(arg)

30. Code File: “complex.c++”

30.1. Non-Class Function Pseudo-code

none

30.2. Class Definitions

30.2.1. Class “complex”

30.2.1.1. Private Member Function Pseudo-code
none

30.2.1.2. Public Member Function Pseudo-code
30.2.1.2.1. complex::complex()
30.2.1.2.2. complex::complex(const double real, const double imag)
30.2.1.2.3. void complex::set(const double real, const double imag)
Initialise both re and im data members with appropriate values, as in header file documentation.

30.2.1.2.4. complex& complex::operator+=(const complex a)
30.2.1.2.5. complex& complex::operator-=(const complex a)
30.2.1.2.6. complex complex::operator-()
Perform simple disjoint addition/subtraction of re/im components.

30.2.1.2.7. complex& complex::operator*=(const complex a)

30.2.1.2.7. complex& complex::operator*=(const complex a)
real result = a.re*re - a.im*im
imag result = a.im*re + a.re*im

30.2.1.2.8. complex& complex::operator/=(const complex a)
Multiply *this and 'a' by complex conjugate of 'a', evaluate real divison of (*this*(conj a))/(a*(conj a))
real result = (re*a.re+im*a.im)/d;
imag result = (im*a.re - re*a.im)/d;
where d =a.re*a.re-a.im*a.im

30.2.1.3. Friend Member Function Pseudo-code
30.2.1.3.1. ostream& operator<<(ostream& output_stream, const complex a)
30.2.1.3.2. istream& operator>>(istream& input_stream, complex& complex_number)
use standard stream input/output operators to i/o chars

30.2.1.3.3. complex operator+(const complex a, const complex b)
30.2.1.3.4. complex operator-(const complex a, const complex b)
Perform simple disjoint addition/subtraction of re/im components.

30.2.1.3.5. complex operator*(const complex a, const complex b)
real result=a.re*b.re-a.im*b.im
imag result=a.im*b.re + a.re*b.im

30.2.1.3.6. complex operator/(const complex a, const complex b)
Multiply 'a' and ''b' by complex conjugate of 'b', and evaluate real division of (a*(conj b))/(b*(conj b))
real result=(a.re*b.re+a.im*b.im)/d
imag result=(a.im*b.re - a.re*b.im)/d
where d=b.re*b.re+b.im*b.im

30.2.1.3.7. complex operator^(const complex a, const complex exponent)
For real power operations, where 'a' and 'b' have zero imag components, use C library function pow.
For cases where a=exp(1)+0j, assume exponential-form complex number, use :
polar_rect(exp(exponent.re), exponent.im) to evaluate rectangular form
For all other cases (ie complex number to the power real number) - use DeMoivre's Theorem
polar_rect(pow(magnitude(a),exponent.re), arg(a)*exponent.re);

30.2.1.4. Static Member Function Pseudo-code
none

31. Code File: "newstring.c++"

31.1. Non-Class Function Pseudo-code

None

31.2. Class Definitions

31.2.1. Class "string_class"

31.2.1.1. Private Member Function Pseudo-code
None

31.2.1.2. Public Member Function Pseudo-code
31.2.1.2.1. string_class::string_class()
31.2.1.2.2. string_class::string_class(const char *source)
31.2.1.2.3. string_class::string_class(const string_class &original)
letters char array is allocated from the Application heap
string data is copied to the new location using the C library function strcpy
string_length is set to the number of bytes allocated for the string (this includes the NULL terminator)

31.2.1.2.4. string_class::~string_class()
Allocated memory is released back to the Application heap

31.2.1.2.5. status string_class::string_copy(char *target)
copying of char array from letters address to target is performed using strcpy

31.2.1.2.6. string_class& string_class::operator=(const string_class &source)
31.2.1.2.7. string_class& string_class::operator=(const char *source)
31.2.1.2.8. string_class& string_class::operator=(const char &source)
target letters char array is deallocated, a new allocation is made of size equal to size of source.letters
source.letters is copied to target.letters using strcpy
source string_length is stored in target string_length

31.2.1.2.9. string_class operator+(const string_class source1, const string_class source2)
31.2.1.2.10. string_class operator+(const string_class source1, const char *source2)
31.2.1.2.11. string_class operator+(const char *source1, const string_class source2)
31.2.1.2.12. string_class operator+(const string_class source1, const char input_char)
string_class to be returned has 'letters' allocated sufficient memory to hold both source1 and source2 concatenated together.
source1.letters and source2.letters are copied to return.letters using strcpy/strcat
return.string_length is set to source1.string_length+source2.string_length-1

31.2.1.2.13. char& string_class::operator[](const int index)
31.2.1.2.14. char& string_class::operator[](const int index) const
Range check is performed upon index.
If index is out of range for the size of the current char array inside string_class object NULL is returned.

31.2.1.2.15. int string_class::length()
returns number of chars stored in the string_class object (not including NULL), ie string_length-1

31.2.1.3. Friend Member Function Pseudo-code
31.2.1.3.1. ostream& operator<<(ostream& output_stream, const string_class output_string)
use output operator to output letters data member, note: no additional white space proceeds letters
RETURN output_stream

31.2.1.3.2. istream& operator>>(istream& input_stream, string_class& input_string)
Skip initial white-space until end of stream or non-space character found.
Replace last read non-space character into stream if stream not empty.
WHILE end of stream not reached AND input_buffer not full
    read in single characters from input stream and store in buffer
    Break out of loop if non alpha-numeric character read in, after putting char back into stream
ENDWHILE
IF buffer is full and not all of the input has been read in
    Output error message to cout
RETURN input_stream

31.2.1.3.3. int operator==(const string_class &string1, const string_class &string2)
31.2.1.3.4. int operator==(const string_class &string1, const char *string2)
31.2.1.3.5. int operator==(const char *string1, const string_class &string2)
31.2.1.3.6. int operator!=(const string_class &string1, const string_class &string2)
31.2.1.3.7. int operator!=(const string_class &string1, const char *string2)
31.2.1.3.8. int operator!=(const char *string1, const string_class &string2)
31.2.1.3.9. int operator>=(const string_class &string1, const string_class &string2)
31.2.1.3.10. int operator<=(const string_class &string1, const string_class &string2)
31.2.1.3.11. int operator>(const string_class &string1, const string_class &string2)
31.2.1.3.12. int operator<(const string_class &string1, const string_class &string2)
Use C library function strcmp to compare two character arrays.
RETURN 1 if comparison is TRUE, else RETURN 0

31.2.1.3.13. void append_name(string_class &target, const string_class &source)

Allocate memory to hold a char array large enough for appending target to source + single space
Update target.string_length to size of temp char array
IF target is an empty string
    copy source.letters to temp char array
ELSE
    copy source.letters to temp char array
    concatenate target.letters to temp char array
concatenate single ASCII 32 space to temp char array
Deallocate target.letters char array
SET target.letters to point to temp char array.

31.2.1.3.14. status search_string(const string_class &source, const string_class &target)

Use C library function strstr to search for source string within target string
RETURN ERROR if target not found, else RETURN SUCCESS.

31.2.1.4. Static Member Function Pseudo-code

none