This exam permits one sheet of notes: NO OTHER BOOKS, NOTES OR EQUIPMENT (besides manual writing instruments and non-data storage watches) are permitted. The duration is 1 hour. Syntax of C++ code counts. Clarity counts also: Obscure, incomprehensible or unnecessarily complicated programs or explanations will be graded as wrong.

PART 1 (10 points)

Fill in each conditional below with C++ code so the if body prints its message precisely when the message is true. You can use extra variables if you give full, correct declarations of them; but extra variables are not necessary.

```cpp
char *pch;  unsigned int X,Y;

1. if(                     !pch or pch==0 or pch==NULL )
   {
       cout << "pch contains the NULL pointer." << endl;
   }

2. (For full 4 point credit, do a test to avoid a crash below.)

   if(               (pch != 0) && (*pch == '\0') )
   {
       cout << "pch points to the empty string." << endl;
   }

3. if(              X < Y              )
   {
       cout << "X < Y under unsigned binary int interpretation" << endl;
   }

4. if(  static_cast<signed>( X ) < static_cast<signed>( Y )  )
   {
       cout << "Under signed int interpretation of their bits, X < Y" << endl;
   }
```
PART 2 (22 points)

This problem is to construct the state transition and action table for a scanner that processes input composed of unsigned (decimal) integer and floating point numerals. An example of correct input is

\[
1 \quad 124 \quad .190 \quad 3.14159 \quad 0 \quad 19.
\]

More specifically,

1. Correct input consists of zero or more “numerals” delimited by WS characters. Zero or more WS characters can appear at the beginning or after the last numeral.

2. Assume the input is terminated by a character of type END.

3. A “numeral” is either an “integer” or a “float.”

4. An “integer” consists of one or more digits (char. class DIG).

5. A “float” consists of one or more DIGs, and either before the DIGs, within them, or just afterward is exactly one DOT.

6. At the first character of each numeral, do the action mark_act

7. At the first character after a float, do the action float_act

8. At the first character after an integer, do the action int_act

9. If the input has an error, go to the ERROR state and stop. At the end of legal input, go to the DONE state and stop.

10. Assume only the characters of the given the 4 classes appear in the input.

(1) (20 points) On the next page, fill in the initializers for the transition table rows. Remember that both a new state and an action must be specified for each transition.

The states have been named to describe the progress of the scan.
enum ch_type { WS, DIG, DOT, END };  
enum state_type { START, DIGS_ONLY, DOT_ONLY, DIGS_AND_DOT, ERROR, DONE };  
typedef void (*action_type) (struct fsa *);
#define NO_ACT static_cast<action_type>( 0 )
extern action_type mark_act, int_act, float_act;
struct transition {ch_type symbol; state_type next; action_type action;};

transition start_row[] = {  
// symbol  next state  action  
// (you fill in) (you fill in)  
{ WS, START, NO_ACT },
{ DIG, DIGS_ONLY, mark_act },
{ DOT, DOT_ONLY, mark_act },
{ END, DONE, (dont care) };

transition digs_only_row[] = {  
{ WS, START, int_act },
{ DIG, DIGS_ONLY, NO_ACT },
{ DOT, DIGS_AND_DOT, NO_ACT },
{ END, DONE, int_act };

transition dot_only_row[] = {  
{ WS, ERROR, (dont care) },
{ DIG, DIGS_AND_DOT, NO_ACT },
{ DOT, ERROR, (dont care) },
{ END, ERROR, (dont care) };

transition digs_and_dot_row[] = {  
{ WS, START, float_act },
{ DIG, SIGS_AND_DOT, NO_ACT },
{ DOT, ERROR, (dont care) },
{ END, DONE, float_act };

transition Table[] = {start_row,digs_only_row,dot_only_row,digs_and_dot_row};

(2) (2 points) Within the fsa scanner driver loop, let ptran be the pointer to the relevant transition and pf be the pointer to the struct fsa. Write a C++ statement that calls the action function if a non-null action function pointer is in the action field.

    if(ptran->action != 0)
      (ptran->action)( pf );  or  (*ptran->action)( pf );
PART 3 (22 points)

1. (2 points) What happens when you give the run command to the debugger if you had not given a break command first?
   The program runs through without stopping except for input or an error (“crash”).

2. (2 points) Declare a C++ function of your choice with at least one argument in the way that means the definition will come from somewhere else.

   ```cpp
   extern int somefun( char *pch ); or
   extern int somefun( char *a );
   ```

3. (2 points) Write a C++ definition of the same function you declared above.

   ```cpp
   int somefun( char *pch )
   {
     return strlen( pch );
   }
   ```

Suppose main.cc
(1) included the declaration but not the definition of your function above,
(2) had the main() function definition,
(3) main() called your function, and
(4) main.cc (and all included files) had no syntax errors.

What happens when each of the commands below is given? Describe the kind and name of file produced and/or the nature of any error messages.

4. (2 points) g++ -g -c main.cc
main.cc is preprocessed and then compiled. An object file named main.o is produced.

5. (2 points) g++ -g -O main main.cc
The linker is run. It fails and reports the error that somefun (or its mangled name) is an undefined symbol.
6. (4 points) Briefly give two advantages of a revision control system over just editing program files. (An answer like “It’s required by management.” doesn’t count, even if it is true.)

It helps maintain a log of the development, implementation decisions, known bugs or their fixes, implemented features, etc.

A file can be replaced with an earlier version or an earlier version can be printed for reference when the current version gets “messed up” or is accidently deleted. A file can be locked for editing by one team member so another’s edits will not interfere with it.

Revision numbers, and even logs can be embedded as strings into object and executable files so the versions of each module used to build the software can be read from these strings.

7. (4 points) (1) When are automatic storage extent variables deallocated, and (2) why should a C/C++ function never return a pointer to one of its own automatic storage extent variables?

(1) Automatic storage extent variables are deallocated when the function activation or block in which they are allocated returns or exits.

(2) This pointer would be returned just when the function deallocates the storage it points to. Thus the pointer always points to deallocated space (it will always be “dangling”) and should never be used.

If such a pointer is ever dereferenced, the result will be unpredictable, but the dereferencing itself will not cause a “crash” because the pointer points into the stack region of virtual memory.

8. (4 points) When a C function is compiled, a symbol identical to the function’s name appears as a label in the assembly language output and as a defined symbol in the object file.

When a C++ function is compiled, its symbol is a “mangled” function name. The function name is “mangled” by appending extra characters after the original function name. What information do these extra characters express?

The types of the return value and each of the arguments. This plus the function name (and class) is called the function’s signature.
PART 4 (19 points)
(1) (13 points) Write complete definition of function copy1String(char **rag, int i) so when it called from main() like

#include "EnvStuff.h"

main(int argc, char *argv[], char *env[])
{
  int I; char *pch;
  // ...
  pch = copy1String( env, I );
  if( !pch )
  { // ...
    delete pch;
  }
  // ...
}

it returns a dynamically allocated copy of the environment string with index I if I is a valid index, and returns the null pointer if not. Hint1: Testing validity here requires a loop. Hint2: Three steps are needed to make the copy with dynamic (free-store) storage extant.

The definition (including the body) of copy1String must be written below in a separate C++ file named EnvStuff.cc

// EnvStuff.cc
// Implementation of the EnvStuff module
#include "EnvStuff.h"
char * copy1String( char **env, int I )
{
  int i = 0;
  while ( (env[i] != 0) && (i < I) )
  {
    i++;
    if ( (i != I) || (env[i] == 0) )
      return 0;
  }

  int size = strlen( env[i] );
  if( size==0 ) return 0;

  size = size + 1;

  char *pcopy = new char[ size ];

  strcpy( pcopy, env[i] );

  return pcopy;
}
(2) (2 points)

// EnvStuff.h
// Interface of the EnvStuff module
// This module contains only copy1String for now..

extern char * copy1String( char **env, int I );

(3) (4 points) Explain why, to determine if I is a valid index, the above program must do more than just test (I >= 0) and, if true, test env[I].

The only way to determine the length of the array of environment string pointers is to find the first null pointer in it. Therefore, all array entries up to env[I] must be tested for being non-null in order to determine if I is an index beyond the end of the array.
PART 5 (15 points)
A MIPS R4000 computer memory contains the following instructions/data (expressed in hexadecimal words) beginning at address 0. For each of the 3 instructions, (1) show it in binary, (2) show how it is decoded (into fields), as affected by the instruction type, and (3) briefly describe the execution of the 3 instructions in sequence, including (4) the new numeric values put in registers. (Hints: Circle the instruction names on the next page as you decode them. The results are independent of the undefined initial register values.)

0x2401BBAA
0x8C020004 <- contents of the word at address 4
0x00411824

0x2401BBAA == 0010 0100 0000 0001 1011 1011 1010 1010
001001 00000 00001 1011 1011 1010 1010
opcode rs=0 rt=1 imm = 0xBBA
row1,col1
of opcode table shows the instruction is ADDIU
reg[ 1 ] = reg[ 0 ] + sign-extend (0xBBA)
result: reg[ 1 ] now holds 0xFFFFBBAA

0x8C020004 == 1000 1100 0000 0010 0000 0000 0000 0100
100011 00000 00010 00001 00000 00000 0100
row4,col3
LW  rs=0 rt=2 imm = 0x0004
reg[ 2 ] = Mem[ reg[ 0 ] + sign-extend (0x0004) ]
        = Mem[ 4 ]
result: reg[ 2 ] now holds 0x8C020004 (see above)

0x00411824 == 0000 0000 0100 0001 0001 1000 0010 0100
000000 00010 00001 00011 00000 010000
SPECIAL rs=2 rt=1 rd=3 sha=0 funct in row4,col4 of SPECIAL table
AND instruction
        = 0x8C020004 & 0xFFFFBBAA
        = 0x8C020000
result: reg[ 3 ] now holds 0x8C020000
(copy of instruction formats table from the MIPS R4000 User Manual, with rs of I-Type format labelled “source”, rt labeled “destination”, and immediate labelled with “is sign extended when used in ADDIU, LW, SW, ADDI, etc."

rs and rt in R-Type format labelled “sources” and rd is labelled “destination”.)

To maximize partial credit CIRCLE the instruction names for the opcodes (and function code) you decoded.
PART 6 (12 points)
Describe algorithms to implement the simplified stopwatch facility specified below. The descriptions
of how the commands will be used and what they do are given below. You write how to implement
them. Do not include error detection or other behavior not specified in the problem.

The algorithm descriptions must include the variable(s) used plus the operations done to im-
plement each command. You may use (clear and precise) English, pseudo-code or C++ . Use the
return value of times(...) for the current real time, in ticks since system startup.

Start: Starts the stopwatch. It will be given only once and will always be the first command.
Read: Report the elapsed time from when the stopwatch was started until now. Read commands
will be given only after the start command.

Variables: StartTime
Start: StartTime = times(...);
Read: Report times(...) - StartTime

Now describe algorithms to implement a “time out” stopwatch.

Start: Same as above.
Read: Same as above except the time report excludes the time during “time out” periods.
Time out: Begin a time out period. “Time out” commands will not be given during “time out”
periods.
Resume: End the current “time out” period (resume time accumulation). “Resume” commands
will not be given when time is accumulating.

(alternative 1:)

Variables:
StartTime, AccumTime, Timing
Start:
StartTime = times(); AccumTime = 0;
Timing = true;
Read:
if (Timing) then report
AccumTime + times() - StartTime;
else report AccumTime;
Time out:
AccumTime = AccumTime + times() -
StartTime;
Timing = false;
Resume:
StartTime = times();
Timing = true;

(alternative 2)

Variables:
StartTime, AccumTimeOut, LastTimeout, Timing
Start:
StartTime = times(); AccumTimeOut = 0;
Timing = true;
Read:
if (Timing) then
report times() - StartTime - AccumTimeOut;
else report LastTimeout - StartTime - AccumTimeOut;
Time Out:
LastTimeout = times();
Timing = false;
Resume:
AccumTimeOut = AccumTimeOut + LastTimeout - times();
Timing = true;