Name: _____ **Number:** _____

Instructor: G. Darling

UNIVERSITY OF VICTORIA

EXAMINATIONS APRIL 2000

SOFTWARE ENGINEERING 265, S01

Page 1:	
Page 2:	
Page 3:	
Page 4:	
Page 5:	
Page 6:	
Page 7:	
Page 8:	
Page 9:	
Page 10:	
Pages 11 & 12:	
TOTAL:	

INSTRUCTIONS:

This is a closed book exam. Students are allowed to bring 2 sheets of 8.5 inch by 11 inch paper (double sided) containing any notes or other reference material. No calculators or other electronic devices are permitted. All answers are to be written on this paper.

This exam consists of **200 MARKS** and **12 PAGES** (plus this cover page) Please count the number of pages in this exam paper before beginning, and immediately report any discrepancy to the invigilator. Please note that none of the questions are optional.

You will have **3 HOURS** to write this exam. Please do not begin until you are instructed to do so. Budget your time based upon the marks allocated for each question.

1. **(40 marks)** For each part below, give a single UNIX (csh) command line that will:

   i. display information about the C standard library function called **sscanf** (note – this information is located in section 3S of the online documentation).

   ii. display information about the current access permission settings for the current directory (but no other files or directories).

   iii. create an **alias** that will alter the behaviour of the **cd** command so that it will do what **cd** normally does, but will also change the system prompt to be the current directory path, followed by a colon and a space.

   iv. pause (i.e., "stop", not "kill") the execution of the current foreground process (allowing you to regain control of the shell so you could type other commands).

   v. place the most recently paused (i.e., "stopped") process into the background (so it can continue to run while you do other things in the foreground).

   vi. repeat the most recent command that began with the letter "g".

   vii. change the access permission setting for the current directory so that you will be able to do anything with this directory (i.e., you will be able to list the files in the directory, add or delete files, and use items that are located within this directory) but all other users will only be able to access items within this directory if they know the names of the items (i.e., others will neither be able to list the files in this directory nor add/delete files).

   viii. merge all of the files in the current directory with names that have the form **LSnn.txt**, where **n** represents a digit (e.g., files **LS01.txt** and **LS16.txt** are examples of files that should be included in this merge) into a single file called **Classlist.txt**. Make sure that the resulting file is sorted into ascending order and contains no duplicate lines. Note that if any files with names like **LSxy.txt** (where **xy** is not a pair of digits) are located in this directory, they should not be included in this merge.

2. **(25 marks)** In answering this question, assume that the following C code exists within the same source file as the other code given below (and assume that all of the appropriate header files have been included).

```
static int lastGlump;
extern int square( int n ) {   return( n * n );   }
extern void cube( int n ) {
   n = ( n * n * n );
}
extern void dbl( int *pn ) {   *pn *= 2;   }
extern int *lastOne( int *pn ) {
   *pn = lastGlump;
   return( pn );
}
static char *glump( char *s, int n ) {
   int i; char *out; lastGlump = n;
   out = (char*) malloc( strlen( s ) * n + 1 );
   strcpy( out, s );
   for( i=1; i<n; i++ ) strcat( out, s );
   return( out );
}
```

For parts (i) through (v) below, give the output these code segments would produce, or just write "**COMPILE ERROR**", "**LINK ERROR**", or "**PROBABLE FATAL ERROR**" if any of these code segments (or the functions they call) would cause either a compile time error, a linking error or probably would cause a fatal runtime error (respectively). Note some non-fatal logic errors may also be present, but please just show the output that would result in those cases.

i.
```
int a = 2, b = 3, c = 4;
a=square(a); cube(b); dbl(&c);
printf("%d,%d,%d ",a,b,c);
```
Output: _____

ii.
```
int n = 1;
dbl((int*)n);
printf("%d",n);
```
Output: _____

iii.
```
int n = 5;
cube(&n);
printf("%d",n);
```
Output: _____

iv.
```
char *s1 = glump( "iq", 3 );
char *s2 = glump( "qi", 1 );
printf("%s|%s",s1,s2);
```
Output: _____

v.
```
char *s = glump(glump(glump("x",1),2),3);
int *p = lastOne(
   (int*)malloc(sizeof(int)) );
printf("%s|%d",s,*p);
```
Output: _____

3. **(20 marks)** Every one of the functions provided in question #2 has one or more problems. In this question you are to address some of those problems.

   i. (10mk) One of the functions is completely useless (in the sense that calling it never has any effect). Rewrite that function in the space below by changing its parameter type(s) in order to make it useful. Do not change its return type. Make any other changes to the function you feel are necessary (don't worry about changing the caller). If appropriate, please also add code to reduce the likelihood that any parameter error could cause unanticipated failure of this function.

ii. (10mk) It is possible for each of the functions listed below to fail due to logic errors or fatal runtime errors (e.g., due to pointer problems, or for other reasons). Add code to each of these methods below to verify the validity of their parameters (using assertions) and to perform any other safety checks you feel are advisable to reduce the likelihood of unanticipated failures. (If there is not enough room, add the code at the side, and use arrows to indicate where the code should be placed).

```
extern void dbl( int *pn ) {

   *pn *= 2;

}
extern int *lastOne( int *pn ) {

   *pn = lastGlump;

   return( pn );

}
static char *glump( char *s, int n ) {

   int i; char *out;

   lastGlump = n;

   out = (char*) malloc( strlen( s ) * n + 1 );

   strcpy( out, s );

   for( i=1; i<n; i++ )

      strcat( out, s );

   return( out );

}
```

4. **(4 marks)** In the space below, define a C preprocessor macro called **new** that takes one parameter that is a type name. Your macro is to invoke malloc to return a pointer of the appropriate type containing the address of a block of memory large enough for a single item of the type passed, or NULL if malloc returns NULL. That is, the following code:

   MyType *p = new(MyType);

   Should store the (appropriately cast) address of a MyType item (or NULL) into p.

5. **(6 marks)** In the space below, write a globally accessible C function called **intArrayAlloc** that takes a single int parameter, **n**, and returns the address of a dynamically allocated array of **n** ints, or returns NULL if that is not possible (i.e., return NULL if the method fails for any reason).

6. **(10 marks)** Write a C function called **swap** with file scope (i.e., accessible within its source file only) that swaps C strings of any length. Your **swap** method must be designed to be called as follows:

```
char *s1 = (char*) malloc( 124 );
char *s2 = (char*) malloc( 78 );
strcpy( s1, "Bart" );
strcpy( s2, "Homer" );
swap( s1, s2 );
printf( "%s, %s", s1, s2 );
```

   The code segment above should produce this output:

```
Homer, Bart
```

   Give the code for your **swap** function below, or explain why that is not possible.

7. **(20 marks)** Write a complete C filter program that reads lines from the standard input, and writes only those lines that are palindromes to the standard output unaltered. A palindrome is defined as a single line of characters such that the first character is the same as the last character, the second character is the same as the second-to-last character, and so on. White space characters (space, tab or linefeed) and the comma and period characters are not considered part of the line, and case is not significant for the purpose of determining whether or not it is a palindrome.

   For example, given the input data below:

```
Nothing interesting here.
A man, a plan, a canal, Panama.
Evil I did dwell, lewd did I live.
Pay no attention to this line either.
```

   the output should be:

```
A man, a plan, a canal, Panama.
Evil I did dwell, lewd did I live.
```

8. **(7 marks)** Give the C++ class definition (such as might appear in a typical class header file) for a base class called **Animal**. No function implementations are to be present within the class definition. This class is to have the following member variables which are to be accessible only within member functions of this class or its descendants:

    name // the name of this type of animal
    weight // the (integer) weight of this particular animal in pounds

    This class is to have a constructor and a destructor. The constructor is to take parameters with the same names as its member variables. The class is also to provide a public "void feed(void)" method. This feed method is to be "pure virtual" (that is, make it such that this method must be overridden by every descendant class).

9. **(8 marks)** In the space below, give the code for the constructor and destructor methods for the Animal class of question 8 as they would appear in a typical separate C++ source file. The constructor is to stores its parameters appropriately into the corresponding member variables, allocating memory if appropriate. It must also output the message "New X." (where "X" is the Animal object's name) to the standard output. The class is to have a destructor that is to output the message "Delete X."(where "X" is the Animal object's name) to the standard output.. The destructor is then to free any memory allocated by your constructor.

10. **(25 marks)** Given the Animal class from question 8 and the classes defined below:

```
class HousePet : public Animal {
   public:
      HousePet( char *name, int weight ) {
         // ...
      }
      void feed( void ) {
         cout << "Food was given to the "
            << name << "\n";
      };
};

class WildAnimal : public Animal {
   public:
      WildAnimal( char *name, int weight ) {
         // ...
      }
      void feed( void ) {
         cout << "The " << name << " found food.\n";
      };
};
```

please answer the questions that follow.

In parts (i) through (v), give the output the code segments below would produce, or just write "**COMPILE ERROR**", "**LINK ERROR**", or "**FATAL ERROR**" if any of these code segments (or the functions they call) would cause either a compile time error, a linking error or fatal runtime error respectively. Note, some non-fatal logic errors may also be present, but please just show the resulting output in those cases.

i.
```
Animal *p;
p = new Animal( "cow", 1000 );
delete( p );
```

Output:

ii.
```
HousePet h( "dog", 80 );
h.feed();
```

Output:

iii.
```
WildAnimal *p;
p = new WildAnimal ( "cheetah", 300 );
p->feed();
delete( p );
```

Output:

iv.
```
HousePet *p = NULL;
if( p == NULL ){
   p = new HousePet( "cat", 10 );
   WildAnimal q( "lynx", 75 );
   p->feed();
   q.feed();
}
```

Output:

v.
```
Animal *a[2];
a[0] = new HousePet( "fish", 1 );
a[1] = new WildAnimal( "tiger", 1 );
Animal **p = a;
(*++p)->feed();
```

Output:

11. **(15 marks)** What is the output of the Perl program shown below?

```
#!/public/bin/perl -w
@strings = ( 'abacab', 'abracadabra', 'razamataz' );
@patterns = (
   'a', 'a.*a', 'ab.*ab',
   '[abrz]{2}', '([abrz]{2}).*\1'
);
sub doMatch {
   ( $str, $pat ) = @_;
   if( $str =~ /$pat/ ) {
        print( "    [$`|$&|$']\n" );
   } else {
        print( "    no match\n" );
   }
}
sub doOneString {
   ( $str ) = @_;
   print( "String: '$str'\n" );
   foreach $pat ( @patterns ) {
        &doMatch( $str, $pat );
   }
}
foreach $str ( @strings ) {
   &doOneString( $str );
```

12. **(20 marks)** In the space below, write a complete Perl program that reads HTML source code from the standard input, and writes to the standard output a list of all of the opening and closing HTML tag names it finds in the file. You may assume that individual HTML tags in the input do not span lines, and that the input files contain correct HTML syntax. Also please just ignore the possibility that tags may appear within quotes or comments. HTML tags (as you know from the course assignments) have the form **<TAGNAME attributes>** or **</TAGNAME>**, where the attributes are optional and the **TAGNAME** or **/TAGNAME** may be preceded or followed with whitespace.

    For example, given this input:

```
<HTML><HEAD><title>Example</Title></HEAD>
<BODY bgcolor="#FFFFFF">
Blah, <br> blah, <br> blah. </BODY></HTML>
```

    Your Perl program should output:

```
HTML
HEAD
title
/Title
/HEAD
BODY
br
br
/BODY
/HTML
```

(This is a blank page – in case you need more room to answer the last question).

END