Instructions: This is an open book, open note, 110 minute exam. You are not allowed to use your homework assignments or your project. There are 5 questions worth a total of 100 points. You must answer all of the questions. Put your name on all pieces of paper that you turn in. Show all of your work, write legibly and good luck.

1) (30 points) You are to add a new language element to ASU and enhance your compiler appropriately. The language should support one-dimensional arrays of elements of any base type. An array **a** is declared as:

```
var a : array [10] of integer;
```

The kth element of the array is accessed as:

```
:= ... a[k] ...
```

Describe in detail the changes you would make to your compiler to:

a) Enhance the scanner to handle the new lexical elements.

b) Enhance the parser to handle the new syntactic elements.

c) Enhance the translation schemes to handle the new semantic elements, including defining any new AST nodes necessary.

d) Enhance the code generator to handle the new AST nodes.

You can assume that arrays are indexed from 1 through the maximum index. If you need any more assumptions just state them explicitly and proceed.

2) (20 points) The definition of a programming language maps the text of a program onto a computation. It does this by way of a variety of rules about what constitutes a *well-formed* and *legal* program.

a) Describe at least four types of rules for the ASU language. (multiple instances of the same kind of rule are unacceptable)

b) Why aren't all of these rules described in a single formalism?

c) What components of a compiler handle the processing of those rules?

d) Why aren't all of these rules processed by a single part of a compiler?

In answering these questions be as detailed as possible and give examples where appropriate, e.g., of a kind of rule that is too "complex" for one kind of processing but for which another kind of processing is suitable.

3) (15 points) Consider the following program:

```
program problem3;
var x, y : integer;

function one : integer;
  var l : integer;
begin
  l := 0;
  return l
end;

procedure up(a : integer);
  var b : integer;
begin
  b := x + a;
{**** Point A ****}
  y := b
end;

begin
  x := 1;
  up(one);
  if y = 2 then
    printPass
  else
    printFail
end.
```

Based on the internal representations discussed in class:

a) Draw a picture of the symbol table at **Point A** in the parse.

b) Draw a picture of the AST for this program.

c) Draw a picture of the CFG for the main program (you don't need to do it for the sub-programs).

Be sure to include attributes and values for nodes/edges where appropriate.

4) (20 points) Given the following LL(1) grammar:

<S>	→	a <A> <B>
	|	<A> <B>
<A>	→	b <S> <A>
	|	ϵ
<B>	→	c
	|	d

a) Compute FIRST and FOLLOW for S, A and B

b) Construct the predictive parse table for a non-recursive top-down parser.

c) How can we determine if the grammar is ambiguous? Is it?

5) (15 points) Define the following concepts and relate them to your project compiler.

a) What is a compiler phase?
What is a compiler pass?

b) How many of phases and passes are in your project compiler?
Describe their inputs/outputs?

c) Could you re-design your compiler to reduce the number of passes?
If so, how?
If not, why not?