

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page. The exam has 5 questions with equal weight. Some questions have multiple parts. Point totals are given in the margin.

Name: _____ **Answer Key** _____

1. _____ **20** _____

2. _____ **20** _____

3. _____ **20** _____

4. _____ **20** _____

5. _____ **20** _____

Total _____ **100** _____

No notes or books may be used on the exam. If you have any questions, please raise your hand and I will try to answer them.

1. Assume standard operator precedences and associativities in the following questions:

5 (a) Translate the following infix expression to prefix notation:

$$a + b \times c - d$$

Answer: $- + a \times b c d$

5 (b) Translate the following postfix expression to infix notation:

$$x y + w \times z -$$

Answer: $(x + y) \times w - z$

5 (c) Translate the following prefix expression to infix notation:

$$\times i - j k$$

Answer: $i \times (j - k)$

5 (d) Why do all operators at the same precedence level always have the same associativity?

Answer: If they did not, then the grammar would be ambiguous. It would not be possible to determine the correct order of evaluation in an expression such as $a + b - c$.

- 10 2. (a) Pascal allows functions to be passed as parameters but not returned as values. In contrast, C allows pointers to functions to be both passed as parameters and returned as values. Explain if this is just an arbitrary limitation in Pascal or if there is a justifiable reason for this restriction.

Consider the following illegal Pascal program in your answer:

```

program illegal (input, output);
  type retval = ^procedure (integer);
  var x : retval;

  function f : retval;
    var y : integer;

    procedure p (z : integer);
      begin
        y := z;
      end;

  begin
    f := &p;                                {address of procedure p is returned}
  end;

begin
  x := f;                                    {x is assigned the return value of f}
  x^ (3);                                    {call to procedure p through x}
end.

```

Hint: What happens when p is finally executed?

Answer: The variable y is nonlocal to procedure p and is located in the stack frame for f. By returning p we are able to call it from the main program without the stack frame for f existing. In general, Pascal does not allow functions and procedures to be returned as values since the function or procedure returned may make use of variables that exist only in the stack frames of enclosing functions. These frames may not exist when the returned function or procedure is called, resulting in an error. Therefore, the restriction is justified.

10

- (b) In Pascal, the type of a formal parameter must be specified as the name of a declared type such as `integer` or `nodeptr` and not as an arbitrary type constructor such as an array or pointer. In contrast, C allows the type of a formal parameter to be specified as an arbitrary type constructor. Explain if this is just an arbitrary limitation in Pascal or if there is a justifiable reason for this restriction.

Consider the following useless C function definition in your answer:

```
void useless (struct { int x, y; } pair) {  
    pair.x = 0;  
    pair.y = 0;  
}
```

Hint: What kind of type equivalence does C use?

Answer: C uses name equivalence for structures. Since the structure definition given has no name, this function can never be called, making it useless. (Even if the structure was given a name, it would be local to the function, allowing only recursive calls.) By forcing the type of a formal parameter to be the name of a declared type, Pascal prevents this type of error. Therefore, the restriction is justified.

3. Consider the following program:

```

program parameters (input, output);
  var a : array [1..10] of integer;
  var x : integer;

  procedure swap (i, j : integer);
    var temp : integer;
  begin
    temp := i;
    i := j;
    j := temp;
  end;

begin
  x := 3;
  a [3] := 1;
  a [1] := 7;

  swap (x, a [x]);

  writeln (x);
  writeln (a [3]);
  writeln (a [1]);
end.

```

5 (a) What is the output of this program if call by value is used?

Answer:

3
1
7

5 (b) What is the output of this program if call by reference is used?

Answer:

1
3
7

5 (c) What is the output of this program if call by value-result is used?

Answer:

1
3
7

5 (d) What is the output of this program if call by name is used?

Answer:

1
1
3

4. Consider the following program:

```

program nonlocals (input, output);
  var x, y : integer;

  procedure A;
  begin
    writeln (x);
  end;

  procedure B;
    var x : integer;

    procedure C;
    begin
      x := 1 + y;
      A;                                { call to procedure A }
    end;

  begin
    x := 2;
    A;                                { call to procedure A }
    C;                                { call to procedure C }
  end;

begin
  x := 5;
  y := 7;
  B;                                { call to procedure B }
end.

```

5 (a) What is the output of this program if static scoping is used?

Answer:

5
5

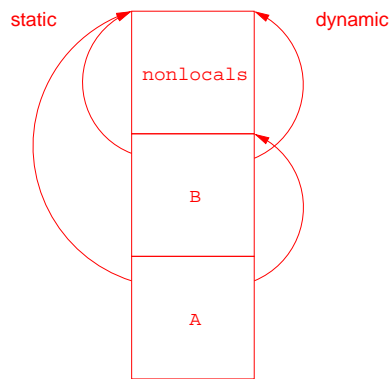
5 (b) What is the output of this program if dynamic scoping is used?

Answer:

2
8

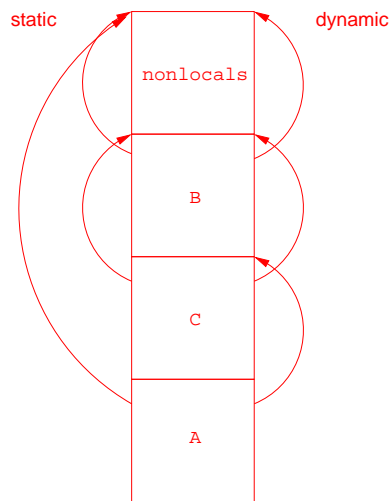
- 5 (c) Draw the stack frames showing the static and dynamic links at the first call to `writeln`.

Answer:



- 5 (d) Draw the stack frames showing the static and dynamic links at the second call to `writeln`.

Answer:



5. Briefly answer the following questions:

5 (a) Why are limited types not possible in a language that uses structural equivalence of types?

Answer: Limited types rely on the structure of a type being hidden from the users of that type.

5 (b) Why is a limited type not a first class object?

Answer: Its value has to be assigned explicitly and it obeys the same rules for equality and assignment as pointer types.

5 (c) Are objects first class in C++? Briefly explain your answer.

Answer: Yes, they can be assigned to, passed as parameters, returned as values, used as parts of other types, created dynamically, and can define their own behavior for operations.

5 (d) In terms of first class objects, in what way do imperative languages do a poor job at abstracting away the details of the machine?

Answer: First class objects in imperative languages are those things that can fit into a single machine entity such as a pointer, integer, or real. Types such as arrays, records, and functions are not first class objects because they do not typically fit in a single machine entity.