1. (20) Run-Time Storage

(a) Run-time storage for compiled programs is organized into static, stack and heap sections. Where does each of the following belong?

i. numeric constants

ii. global variables

iii. variables local to a function

iv. explicitly dynamically allocated strings (using malloc or new)
(b) Sketch the run-time stack for the given code when function \texttt{p} is called on line (18). Give identifier names but not sizes and show the data, or access, link (for scope search) and the control links (return address) on the stack.

\begin{verbatim}
(1) #include <stdio.h>

(2) int x;

(3) void p(void)
(4) {
(5)    double r=2;
(6)    printf("%g\n",r);
(7)    printf("%d\n", x);
(8) }

(9) void r(void)
(10) {
(11)    x = 1;
(12)    p();
(13) }

(14) void q(void)
(15) {
(16)    double x = 3;
(17)    r();
(18)    p();
(19) }

(20) main()
(21) {
(22)    p();
(23)    q();
(24)    return 0;
(25) }
\end{verbatim}
2. (20) Attribute Grammars.

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>Semantic Rules</th>
</tr>
</thead>
</table>
| 1. `<bin - num> → <bin - dig>_1 . <bin - dig>_2` | `val(<bin - num>) = val(<bin - dig>_1)`  
|                       | `+ val(<bin - dig>_2)/2 * len(<bin - dig>_2)`     |
| 2. `<bin - dig>_1 → <bin - dig>_2 <bit>`           | `val(<bin - dig>_1) = 2 * val(<bin - dig>_2)`  
|                       | `+ val(<bit>)`                                      |
|                       | `len(<bin - dig>_1) = len(<bin - dig>_2) + 1`      |
| 3. `<bin - dig> → <bit>`                            | `val(<bin - dig>) = val(<bit>)`                    |
| 4. `<bit> → 0`                                             | `val(<bit>) = 0`                                    |
| `<bit> → 1`                                             | `val(<bit>) = 1`                                    |

(a) Identify the attributes in the grammar.

(b) For each grammar symbol that has an attribute, and tell if its attributes are inherited or synthesized.

(c) Using the given attribute grammar, construct and decorate a parse tree for the string

\[10.001\]
3. (20) ADT'S.

(a) A partial specification for an abstract data type is given below. Write a complete set of axioms for the data type.

**type** symboltable(name,value)

**operations:**
- **create**: → symboltable creates an empty symbol table
- **search**: symboltable × name → value returns a copy of the value associated with name
- **insert**: symboltable × name × value → symboltable inserts a (name,value) pair into symbol table
- **remove**: symboltable × name → symboltable deletes pair with given name from symbol table
- **isin**: symboltable × name → Boolean returns true if name is in the table and false otherwise
- **isempty**: symboltable → Boolean returns true if the table is empty and false otherwise

**variables:** s: symboltable; x,y: name; v: value;
(b) ADT 2 Consider the lattice point ADT as given below.

**Functions:**
- create: \( \text{int} \times \text{int} \rightarrow \text{lp} \)
- getx: \( \text{lp} \rightarrow \text{int} \)
- gety: \( \text{lp} \rightarrow \text{int} \)
- add: \( \text{lp} \times \text{lp} \rightarrow \text{lp} \)
- subtract: \( \text{lp} \times \text{lp} \rightarrow \text{lp} \)
- multiply: \( \text{lp} \times \text{lp} \rightarrow \text{lp} \)
- negate: \( \text{lp} \rightarrow \text{lp} \)

**Variables:**
- \( z_1, z_2: \text{complex} \)

**Axioms:**
1. \( \text{getx}(\text{create}(i,j)) = i \)
2. \( \text{gety}(\text{create}(i,j)) = j \)
3. \( \text{getx}(\text{add}(z_1, z_2)) = \text{getx}(z_1) + \text{getx}(z_2) \)
4. \( \text{gety}(\text{add}(z_1, z_2)) = \text{gety}(z_1) + \text{gety}(z_2) \)
5. \( \text{getx}(\text{subtract}(z_1, z_2)) = \text{getx}(z_1) - \text{getx}(z_2) \)
6. \( \text{gety}(\text{subtract}(z_1, z_2)) = \text{gety}(z_1) - \text{gety}(z_2) \)
7. \( \text{getx}(\text{multiply}(z_1, z_2)) = \text{getx}(z_1) * \text{getx}(z_2) - \text{gety}(z_1) * \text{gety}(z_2) \)
8. \( \text{gety}(\text{multiply}(z_1, z_2)) = \text{getx}(z_1) * \text{gety}(z_2) + \text{gety}(z_1) * \text{getx}(z_2) \)
9. \( \text{getx}(\text{negate}(z_1)) = -\text{getx}(z_1) \)
10. \( \text{gety}(\text{negate}(z_1)) = -\text{gety}(z_1) \)

Addition and multiplication of integers is commutative. That is, if \( a \) and \( b \) are integers, then \( a + b = b + a \) and \( a * b = b * a \). Prove that addition of lattice points is commutative.
4. (15) Formal Semantics. Assuming the sample language and semantics developed in class, show the Env function before the start of execution and after each step of execution for each of the following pieces of code.

(a) Assignment

(1) \( n := (1+3)*2 \)

(b) Loop

(1) \( n := 2; \)

(2) \( \text{while } n \text{ do } n := n - 1 \text{ od} \)
5. (25) Describe the high-level design for a new programming language. Justify your design decisions. Your answer should include the following elements, and additional topics are encouraged:

(a) Target application domain, such as general-purpose, scientific, systems, artificial intelligence, management information systems, or web-centered

(b) Implementation method such as interpreter or compiler

(c) Programming paradigm, such as imperative, logic, functional or object-oriented

(d) Type system, including built-in data types, mechanisms for user-defined types, and type conversion

(e) Scope rules

(f) Parameter passing mechanisms