B1. Compilers (25 points) Answer all questions.

(a) (8 points)
Translate the following code segment into an abstract syntax tree and then into three address code.

```
b = a * c – (e + 2)/ 2
x = 2
w = b + e + 2
while (w < b) do
   begin
        print b
        if x < b then x = x + 2
                 else w = w + 1
        w = w + 1
        x = 3
    endwhile
call sub(e, b)
w = e + 2
print w, b, x
```

(b) (3 points)
Draw a flow graph for the code above, indicating the statements in each basic block.

(c) (8 points)
In the code above, (e + 2) is evaluated in three places. This is called a common subexpression evaluation. Is it safe to eliminate any of these subexpression evaluations? If so, which ones? Justify your answer. Show what would replace such a removed subexpression evaluation. What analysis needs to be done to ensure the safety of removal of common subexpressions? Explain how the results of the analysis are used to indicate safety.

(c) (6 points)
Describe how 2 programming language features can affect the accuracy of data flow analysis.

B2. Compilers (25 points) Answer all questions.

(a) (4 points) Define and explain the role of a register interference graph.

(b) (5 points) Describe a common heuristic used for global register allocation based on interference graphs. Explain why a heuristic technique is used.

(c) (5 points) Show the following code with the given register assignment and spilling in place.

```
Read b,c
a = b + c
b = c * 2
d = a – c
print a,b,c,d
```

Register assignment: r1: b, d
r2: c, a

(d) (11 points) Consider the following class hierarchy:

```
Class A
   Int x, y;
   Method v;
   Method n;
End A
```

```
Class B extends A
   int y, t, s;
   Method v;
   Method u;
End B
```

```
Class C extends B
   int t, p,q;
   Method n;
   Method m;
End C;
```

i. (5 points) Draw the layout of an object for each of the classes above.

ii. (6 points) Draw the dispatch tables for each of the classes above.

B3. Compilers (25 points) Answer all questions.

(a) (7 points)
Both reference counting and compaction are memory management strategies that handle data objects of different sizes. Explain the advantages and disadvantages that each of these two methods have with respect to each other.

(b) (9 points)
Describe two ways to implement scoping rules in the symbol table so that it can contain several variables with the same name but different scope at the same time.

(c) (9 points)
Describe two ways to implement scoping rules in the run-time function stack so that functions can access non-local variables efficiently. (The non-local variables of a function are the variables that appear in the function definition but that are not declared in the function definition.)

B4. Compilers (25 points) Answer all questions.

(a) (6 points)
Let G be the grammar

$S \rightarrow A\, f\, B$
$A \rightarrow a\, A$
$A \rightarrow b\, B$
$A \rightarrow \epsilon$
$B \rightarrow B\, c$
$B \rightarrow \epsilon$

Compute the **FIRST** and **FOLLOW** sets for all the nonterminals (S, A, B) in the grammar.

(b) (6 points)
Compute **FIRST**($A\ B\ S$) for grammar G above.

(c) (6 points)
Suppose that you have already computed the **FIRST** and **FOLLOW** sets for all the nonterminals in a grammar and the **FIRST** sets for the right-hand sides of all the rules in the grammar. How can you use this information to decide whether it is an **LL(1)** grammar?

(d) (7 points)
Construct an **SLR** parse table for the following grammar; show as much of your intermediate work as you can.

(1) $E \rightarrow i\, B$
(2) $B \rightarrow -\, E$
(3) $B \rightarrow \epsilon$