Compilers

Date: Thursday 27th May 2010
Time: 09.45 – 11.45

Please answer any THREE Questions from the FIVE questions provided

This is a CLOSED book examination

The use of electronic calculators is NOT permitted.
1. a) What differences would you expect in (i) compilers designed for a compiler that generates code for the embedded processor of a mobile phone and (ii) for a compiler used in an introductory programming course in a high school. (3 marks)

b) Consider the following C program:

```c
1. int main()
2. {
3.  char *p;
4.  register int x;
5.  scanf("%d", &x); scanf("%d", &y);
6.  if (x > y) then { x=y; }
7.  y=x+1_007;
8.  y+=(double)(p);
9.  printf("%d\n", y);
10.}
```

When compiling this program, a certain C compiler produces the following error messages:

- Line 6: syntax error at '@' token
- Line 6: 'y' undeclared (first use in this function)
- Line 7: 'then' undeclared (first use in this function)
- Line 7: parse error before '{' token
- Line 8: underscore in number
- Line 9: pointer value used where a floating point value was expected
- Line 10: warning: unknown escape sequence '\n'

Indicate which compilation phase is most likely to generate each of the above errors. Justify your answer. (7 marks)

c) Briefly describe the output of each major compilation phase for the assignment statement \( x = y * z + 3 + 2 \), where \( x, y, z \) are real numbers. State any assumptions you make. (6 marks)

d) Explain why the register keyword in C, which tells the compiler to store a variable in a CPU register (for example, see line 4 above), might have been a good idea when the language was first developed, but is probably not nowadays. (4 marks)
2. a) Write a regular expression that recognises the same string as the following C-like code:

```c
ch=getchar();
if (ch=='a') {
    ch=getchar();
    while (ch=='a' || ch=='b') {
        ch=getchar();
        do {
            ch=getchar();
            } while (ch=='c');
    }  
    if (ch=='d') accept();
} else {
    if (ch=='e') accept();
}
```

(4 marks)

b) Consider the following regular expression

```regex
( 0 | ε) ( 0 | 1 )* 0
```

i) Construct an NFA for the regular expression above using Thomson’s construction. (4 marks)

ii) Convert the NFA to a DFA using the subset construction algorithm. Feel free to use a shortened version of the NFA for this conversion, which does not include unimportant $\varepsilon$-transitions. (8 marks)

iii) Use the algorithm for state minimisation to construct a minimised DFA. (4 marks)
3. Consider the following context-free grammar for variable declarations in a Pascal-like language:

```
Goal → Var_decl
Var_Decl → var Decl_List
Decl_List → Decl ; Decl_List
Decl_List → Decl ;
Decl → Id_List : Id_Type
Id_List → Id_List , identifier_name
Id_List → identifier_name
Id_Type → integer
Id_Type → real
Id_Type → boolean
Id_Type → char
```

where the terminal symbol “identifier_name” consists of 1, 2, or 3 letters.

a) Derive a leftmost derivation for the string

```
var x,y:integer; z:boolean;
```

and show the corresponding parse tree. (6 marks)

b) Transform this grammar so that it can be used to construct a top-down predictive parser with one symbol of lookahead. (6 marks)

c) Briefly describe how a bottom-up shift-reduce parser works. Show, in full detail, the steps that a shift-reduce parser would follow to parse the string

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

Explain what a shift/reduce conflict is. [NB: you should consider the original grammar, not the grammar derived in b) above]. (8 marks)
4. a) Convert the following, C-like code fragment into stack machine code. State your assumptions.

\[
\text{if (x > 0) \{ x = y + 1; \}}
\]

(3 marks)

b) The following shows two different ways to generate code for the assignment statement

\[
x = a \times b + c \times d + e \times f
\]

Explain the difference between the two approaches highlighting the advantages of each approach.

\[
\begin{align*}
\text{load r1, @a} & \quad \text{load r1, @a} \\
\text{load r2, @b} & \quad \text{load r2, @b} \\
\text{mult r1, r1, r2} & \quad \text{load r3, @c} \\
\text{load r3, @d} & \quad \text{load r4, @d} \\
\text{mult r2, r2, r3} & \quad \text{load r5, @e} \\
\text{add r1, r1, r2} & \quad \text{mult r2, r2, r3} \\
\text{add r1, r1, r5} & \quad \text{add r1, r1, r3}
\end{align*}
\]

(3 marks)

c) Sketch an algorithm that you would incorporate in a compiler to generate code that implements the multiplication of an integer variable with an integer constant as a sequence of shifts and additions. (5 marks)

d) Discuss two compiler optimisations, whose impact on program performance would be significantly higher when compiling Java than when compiling C. (4 marks)

e) A certain C compiler performs the following optimisation:

\[
\begin{align*}
\text{/* before optimisation */} & \quad \text{/*after optimisation */} \\
\text{for (i=1; i<1000; i++)} & \quad \text{for (i=1; i<1000; i++)} \\
\text{for (j=1; j<1000; j++)} & \quad \text{for (j=1; j<1000; j++)} \\
\text{a[j][i]+=1;} & \quad \text{a[i][j]+=1;}
\end{align*}
\]

Why is this an optimisation? What would stop the compiler from applying this optimisation if the assignment statement of the original code was

\[
a[j][i]+=a[j-1][i+1] 
\]

(5 marks)
5.  a) Consider the following basic block:

1. load r1, @x
2. load r2, @y
3. add r3, r1, r2
4. mult r4, r1, r2
5. add r5, r3, r4
6. add r6, r5, r3
7. sub r7, r6, r2
8. mult r8, r4, r1

i) Show the result of using the top-down algorithm to allocate registers. (4 marks)

ii) Show the result of using the bottom-up (Best's) algorithm to allocate registers. (4 marks)

[NB: You should assume that your target processor has 4 physical registers. Also, in both cases, you should briefly describe how the algorithm works and show how the basic block looks after you apply each algorithm.]

b) i) Suggest an efficient list scheduling algorithm for instruction scheduling on a hypothetical processor with multiple functional units, if the cost of executing an instruction (delay latency of the instruction) is different on each functional unit. (6 marks)

ii) Apply your algorithm above to the basic block below and show the schedule.

1. load r1, @x
2. load r2, [r1+4]
3. add r3, r3, 1
4. mult r6, r6, 17
5. store r6
6. div r5, r5, 29
7. add r4, r2, r5
8. mult r5, r2, r4
9. store r4

You should assume three functional units. The cost of executing each instruction (delay latency of each instruction) on each functional unit is given by the table below.

(Question 5 continues on the following page)
(Question 5 continues from the previous page)

<table>
<thead>
<tr>
<th></th>
<th>Functional unit 1</th>
<th>Functional unit 2</th>
<th>Functional unit 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>div</td>
<td>3</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>load</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>mult</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>store</td>
<td>7</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

(6 marks)