Name ______________________ Student No. ______________________

*No aids allowed. Answer all questions on test paper. Use backs of sheets for scratch work.*

Total Marks: 100

1. Answer each of the following questions about program development:

[5] (a) List the six desirable program characteristics.

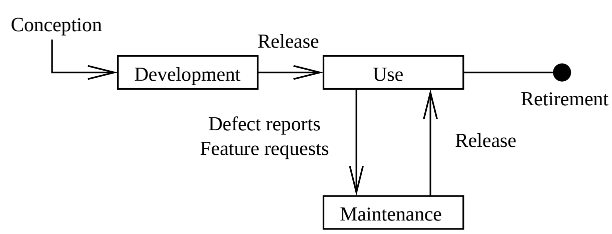
[5] (b) List the four basic activities of any software development effort.

[5] (c) Explain the Software Life Cycle.

**Solution:** (a) Integrity, Clarity, Simplicity, Efficiency, Modularity, Generality.

(b) Establish the requirements, create a design, implement the code, test the implementation.

(c)



2. Consider the following short program given in pseudo-code:

```
boolean palindrome(array A, int n)
        i ← 1
        while(i ≤ ⌊n/2⌋)
              if (A[i] ≠ A[n − i + 1])
                 return FALSE
              i ← i + 1
        end while
        return TRUE
```

Let the precondition be: $n \geq 1$, $n \in \mathbb{N}$, $A[1 \ldots n]$ is an array of characters.

Let the postcondition be: the program returns TRUE iff $A$ is a palindrome.

[10] (a) Your job is to give a *loop invariant* for which the postcondition follows from the precondition (*assuming that the program terminates*), and prove that the loop invariant holds.

[5] (b) Give an informal argument as to why the program terminates using the following fact: "Every *decreasing* sequence of natural numbers is finite." (Hint: Consider $d_i = \lfloor \frac{n}{2} \rfloor - i$.)

**Solution:** (a) Let the loop invariant be: "after the $k$-th iteration (*if it exists*), $i = k+1$, and $\forall j, 1 \leq j \leq k$, $A[j] = A[n-j+1]$." We now prove that it holds by induction on the number of iterations. BC: after zero iterations, there are no $j$'s such that $1 \leq j \leq 0$, so the statement is vacuously true. IS: we know that after $k$ iterations, $A[j] = A[n-j+1]$ for all $1 \leq j \leq k$; after one more iteration we know that $A[k+1] = A[n-(k+1)+1]$, so the statement follows for all $1 \leq j \leq k+1$.

(b) Consider $d_i = \lfloor \frac{n}{2} \rfloor - i$. By the precondition, we know that $n \geq 1$, and it is a natural number. The sequence $d_1, d_2, d_3, \ldots$ is a decreasing sequence of natural numbers (bec. $i \leq \lfloor \frac{n}{2} \rfloor$), so by the fact it is finite, and so the loop terminates.

3. Recall the *Simple Knapsack Problem*, where on input: $d, w_1, \ldots, w_d, C \in \mathbb{N}$, we want to compute $\max_S\{K(S)|K(S) \leq C\}$, where $S \subseteq \{1, \ldots, d\}$, and $K(S) = \sum_{i \in S} w_i$.

[5] (a) Define the array of subproblems $R(i, j)$.

[10] (b) Give a recurrence for computing $R(i, j)$.

[10] (c) Suppose that $R$ has been computed for all the entries. Show how to extract the actual set of weights from the values in $R$.

**Solution:** (a) $R(i, j)$ is TRUE iff there exists a subset $S$ of the first $i$ weights such that $\sum_{k \in S} w_k = j$.

(b) $R(i, j) = \text{TRUE} \iff [R(i-1, j) = \text{TRUE} \vee (j - w_i \geq 0 \wedge R(i-1, j-w_i) = \text{TRUE})]$

(c) First, we need to find the solution; so we look in the last row (i.e., row $d$) for the largest non-zero $j$. That is, the solution is $M = \max_{0 \leq j \leq C}[R(d, j) = \text{TRUE}]$. Now we check if $R(d-1, M) = \text{TRUE}$. If yes, then we know that weight $w_d$ is not necessary, so we don't include it, and continue looking at $R(d-2, M)$. If no, then because $R(d, M) = \text{TRUE}$, we know that $M - w_d \geq 0 \wedge R(d-1, M-w_d) = \text{TRUE}$. So we include $w_d$, and continue looking at $R(d-1, M-w_d)$. We stop when we reach the first row or column.

4. This question is about the PK system for propositional reasoning.

[5] (a) State the cut rule, and the two contraction rules.

[10] (b) Show that both contraction rules can be simulated with the cut rule plus some weakenings and exchanges.

[10] (c) Does the sequent $(P \vee Q) \wedge \neg P \rightarrow (P \wedge P) \wedge (Q \vee Q)$ have a PK proof? If yes, give it, if no, explain why.

**Solution:** (a)

$$\frac{\Gamma \rightarrow \Delta, \alpha \quad \alpha, \Gamma \rightarrow \Delta}{\Gamma \rightarrow \Delta} \qquad \frac{\alpha, \alpha, \Gamma \rightarrow \Delta}{\alpha, \Gamma \rightarrow \Delta} \qquad \frac{\Gamma \rightarrow \Delta, \alpha, \alpha}{\Gamma \rightarrow \Delta, \alpha}$$

(b)

$$\frac{\Gamma \rightarrow \Delta, \alpha, \alpha \qquad \dfrac{\alpha \rightarrow \alpha}{\alpha, \Gamma \rightarrow \Delta, \alpha}}{\Gamma \rightarrow \Delta, \alpha}$$

where the double line indicates several weakenings and exchanges. The left contraction rule can be simulated in a similar fashion.

(c) The sequent is not valid: consider the truth assignment $\tau(P) = F$ and $\tau(Q) = T$. Then $\tau$ does note satisfy it. Hence, by the *Soundness Principle for PK*, it does not have a proof.

[20] 5. Give a procedure for telling well-formed propositional formulas from badly formed propositional formulas. An example of a well-formed propositional formula is $P \wedge \neg P$; an example of a badly formed one is $((P\neg$. Make sure that your algorithm is correct (in particular, that it always terminates).

**Solution:**

Give a Context-Free grammar for the set of well-formed propositional formulas, translate it to Chomsky Normal Form, and apply the CYK algorithm.