ECE650 Assignment 1: Object Oriented Programming, Turing Completeness, Control Flow and Call Stack, Regular Expressions, Context-Free Grammars, Parsers

Question 1: True/False questions (40 points)

Please specify if the following assertions are True or False. You will get points only if you provide appropriate justification. Otherwise you will get 0 points. Each question is worth 2 points.

1. The universal language is context-free.

   T

   The universal language can be expressed as Σ*, which is a context-free language, as Σ being the alphabet with finite number of elements, it's closed under Kleene star, so Σ* is also context-free.

2. It is impossible for a string within a context-free language to have more than one leftmost derivations.

   F

   It is possible for a language with ambiguous grammar to have more than one leftmost derivations.

3. The union of a context-free language and a regular language is a context-free language.

   T

   It's what the regular closure property stated. Also, as regular language is also CF, then the union of 2 CFL is obviously a CFL.

4. It is known that context-free languages are not closed under intersection. Since every regular language is context-free, it follows that the intersection of a regular language and a context-free one is also not context-free.

F

Also by the regular closure property. The regular language and context-free language are closed under intersection.

5. The following grammar represents the language of all strings over the alphabet $\{a, b\}$ with equal number of a's followed by equal number of b's

$$S \rightarrow aSb \mid ab$$

T

Since for every derivation, there must have one 'a' and one 'b' with the 'b' following the 'a'.

6. Let a, b, c, d be terminals and S, A, B be non-terminals where S is the start symbol. Is the following grammar in the appropriate form for context-free grammars:

$$S \rightarrow AcB$$

$$Ac \rightarrow ab$$

$$B \rightarrow b$$

F

The reason is that it has Ac→ab at line 2. But only non-terminals S,A,B can be used for derivation (Only S, A, B can on the left side of →).

7. The following grammar:

$$S \rightarrow aS \mid aSbS$$

generates the set S of all strings s, over the alphabet $\{a, b\}$ such that each prefix of s has at least as many a's as b's.

T

Each prefix of s has at least as many a's as b's means that num(a) >= num(b). And this is true S can only be derived as aS or aSbS. When it's derived to aS, in the prefix, the number of a is greater than number of b. And when it's derived to aSbS, there's as many a's as b's in the prefix since a appear earlier than b.

8. The following grammar is ambiguous

$$S \rightarrow SbF \mid F$$

$$F \rightarrow b \mid \epsilon$$

T

Use example string 'b'. 1) S→SbF→FbF→ϵbF→bF→bϵ→b.

And 2) S→F→b

These 2 have different left-most derivation and can form 2 different derivation tree. Hence it's ambiguous.

9. Every non-context-free language L is infinite, i.e. has infinitely many distinct strings as elements of the language L.

T

Say we have a language L and it's finite. In this case, it has finitely many of elements and is definitely regular. Hence if a language is finite, then it must be regular. And if a language is infinite, then it's only possible that it's regular. But however, with a non-context-free language, it absolutely has to be infinite, if it's not, then it's regular. But a language can't be regular and non-context-free at the same time, contradiction.

10. The intersection of two non-CFL languages cannot be a CFL. (Hint: You can use non-CFL languages like $\{ a^n b^n c^n \}$ as part of your answer.) If your answer is TRUE, then provide a proof. The proof doesn't have to be formal, but should capture your reasoning clearly. If FALSE, provide a counterexample.

F

Counter Example: Let L1 = $\{a^n b^n c^n\}$, L2 = $\{d^n e^n f^n\}$. Both these 2 languages are not context free, however, their intersection, $L1 \cap L2 = \emptyset$, is empty, which is context free.

11. If a language L is regular, then $(\overline{L^*})^*$ is regular.

T

As regular languages are closed under complement and Kleene star. So if L is regular, L* is regular, and that makes $\overline{L*}$ regular. And by using another Kleene star, $(\overline{L*})^*$is also regular as it's closed under Kleene star.

12. Every finite language is the language of some regular expression.

T

For any finite language, it contains a finite number of elements/strings. And the union/Kleene star operations on those strings can be used to express the language.

13. The following is an identity, where *r, s* are regular expressions.

$$(r + s)^* = r^* + s^*$$

F

$(r+s)^* = \{\lambda,r,rr,\ldots\} \cup \{\lambda,s,ss\}$

$r^*+s^* = \{\lambda,r,s,rr,rs,\ldots\}$

And $(r+s)^*$ doesn't contain the string 'rs'. Hence they are not the same.

14. The following is an identity, where *r, s* are regular expressions.

$$(sr)^*(rs)^* = (srrs)^*$$

F

$(sr)^*=\{\lambda,sr,srsr,\ldots\}$ $(rs)^*=\{\lambda,rs,rsrs,\ldots\}$

$(sr)^*(rs)^*=\{\lambda,sr,srrs,srrsrs,srsr,srsrrs,srsrrsrs\}$

$(srrs)^*=\{\lambda,srrs,srrssrrs,srrssrrssrrs,\ldots\}$

However, $(srrs)^*$ doesn't contain $\{sr,srrsrs,srsrrs,\ldots\}$. Hence they are not the same.

15. $\emptyset^*$ and $\emptyset$ represent the same language.

F

$\emptyset^*$ contains λ while $\emptyset$ doesn't.

16. Give a context-free grammar for generating the language described by the regular expression $(10^*1^* + 0)$.

S → 1S | 0S | ∈

17. Give a context-free grammar for generating the set of odd-length palindromes over the alphabet $\{a, b, c\}$.

S → aSa | bSb | cSc | a | b | c

18. Consider the context-free grammar over the alphabet $\{a, b\}$:

$$S \rightarrow aAA$$

$$A \rightarrow bS \mid a$$

Write down 3 different strings of length less than 10 that can be derived from this grammar.

1. aaa
2. abaaaa
3. aabaaa

19. Consider the context-free grammar over the alphabet $\{a, b\}$:

$$S \rightarrow aSbSbS \mid bSaSbS \mid bSbSaS \mid \epsilon$$

describe in english the set of strings it accepts.

It accepts the set of strings such that the number of b's in the string is twice as many as the number of a's in the string.

20. In class we studied that if L is a CFL and R is a regular language, then $L \cap R$ is a CFL. This property of CFLs is called regular closure. We also studied that the language $G = \{a^n b^n \mid n \geq 0\}$ is context-free. Use the regular closure property of CFLs to show that the following language L is a CFL:

$\{a^n b^n \mid n$ is divisble by 2$\}$

We know that L1 = $\{a^n b^n \mid n \geq 0\}$ is context-free, and L2 = $\{a\}$ is regular, L3 = {b} is regular. Since regular languages are closed under *, we have $L2^* = \{a\}^*$ *and* $L3^* = \{b\}^*$ being regular. Also, as regular languages are closed under concatenation, we have $L4 = L2^* \cdot L3^* = \{a\}^* \cdot \{b\}^*$ being regular. In this case, due to the regular closure property, we have $L1 \cap L4$ being context-free. Hence, $L1 \cap L4 = \{a^n b^n\} \cap \{\{a\}^* \cdot \{b\}^*\} = \{a^n b^n \mid n \textit{ is divisible by } 2\}$ is context-free.

Question 2: Object-Oriented (OO) Concepts (30 points)

For each of the sub-questions below, provide a concise, correct, and complete answer. Each sub-question is worth a maximum of 10 points.

1. **Basic Object Oriented Design and Data Encapsulation:** After learning about basic OO design and data encapsulation, Ian realized that his code below in Listing 1 violates the principle of data encapsulation. He then went ahead and made a very minor change to the getter function get_a() to obtain the code in listing 2:

```
class Base {
   private:
     int a;
   public:
      Base(){...}
      int* get-a() {
           ...
         return &a;
      }
};
```

Listing 1: original code

```
class Base {
   private:
     int a;
   public:
      Base(){...}
      int& get_a() {
           ...
         return a;
      }
};
```

Listing 2: modified code

Is Ian's modified code in Listing 2 fundamentally different from the original code in listing 1, when viewed from the point of view of violation of the principle of data encapsulation?

In both listings, the variable int a has been made private, and this could restrict unauthorized access to the data. However, in Listing 1, the getter function returned the address of a, and by having the address, the value of 'a' may be changed. Which is a violation of data encapsulation. And as for Listing 2, the getter function returns a reference to the int variable, it's the reference to the piece of memory, and the value of 'a' could still be changed in this way, hence it's still a violation of data encapsulation.

2. **Templates in C++:** In class we studied function and class templates. Chunxiao recently wrote the following pieces of code, both using function templates. He was very excited when the first piece of code worked correctly, but the second one gave a compile error. From our discussion in class, he realized that he had forgotten to overload the < operator for the ADT class and hence he was getting a compile error. Therefore, he went ahead and fixed that issue by overloading the < operator. However, he continued to get a compile error. Can you find the issue for him and tell him how he can fix it?

```
1  template <typename T>
2  T min(T a, T b) { return a < b++ ? a : b;}
3
4  int main(){
5      int x;
6      x = min<int>(10, 3);
7      return x;
8  }
```

Listing 3: code using templates

```
1  template <typename T>
2  T min(T a, T b) { return a < b++ ? a : b;}
3
4  class ADT {// Abstract DataType
5   private:
6      int value;
7   public:
8      ADT(int v) : value(v) {}
9      friend bool operator< (const ADT& t1, const ADT& t2) {
10         return t1.v < t2.v;
11         }
12 };
13 int main(){
14     ADT x(10), y(4);
15     y = min<ADT>(10, x);
16     return y;
17 }
```

For line 10, as object of ADT class, it has no member named v, only one int member named value. Hence, it should be changed to *return t1.value < t2.value+1*; And the +1 corresponds to the ++ after b in line 2

Listing 4: code using templates and an ADT

This is my modification of the code in order for it to compile and run.

```
4
5   template <typename T>
6   T mins(T a, T b){
7       //cout << "test" << endl;
8       return a < b ? a : b;
9   }
10
11  class ADT {
12      private:
13          int value;
14      public:
15          ADT(int v): value(v){}
16          friend bool operator< (const ADT& t1, const ADT& t2){
17              return t1.value < t2.value+1;//t2.value is read-only, so can't use ++ here as it will just modify the value, which is not permitted
18          }
19          int getter(){
20              return value;
21          }
22  };
23
24  int main(){
25      //cout << "Hello" << endl;
26      ADT x(10), y(4);
27      y = mins<ADT>(15, x);
28      cout << y.getter()+1;
29      return y.getter()+1;
30  }
```

The reason the ++is removed here is in the comment at line 17.

Since the value variable is private, there should be a getter function to obtain its value.

As the < is overridden, I think it's better to use y = mins<ADT>(y, x) rather than this code here, as it's overridden to compare 2 ADT objects.

The +1 in the return line is also corresponding to the ++ in line 2 of Listing 2.

3. **Semantics of Inheritance and Constructors in C++:** Mary hastily wrote the following code (Listing 5) and got a very unexpected compile error. She quickly realized her error and went ahead and fixed to obtain the code Listing 6 below. However, now she is not able to instantiate objects of the base class. What is going on?

```
#include <iostream>
class base {
   private:
      base() {
         std::cout << "Hey: I am the base constructor()\n";
      }
};
class derived: public base {
   public:
      derived() {
         std::cout << "Listen, I may be derived, but I am better than you\n";
      }
};
int main() {
   derived d;
}
```

Listing 5: original code

```
#include <iostream>
class base {
   protected:
      base() {
         std::cout << "Hey: I am the base constructor()\n";
      }
};
class derived: public base {
   public:
      derived() {
         std::cout << "Listen, I may be derived, but I am better than you\n";
      }
};
...
```

Listing 6: modified code

So she changed the private accessor to protected. In order to instantiate objects, the constructor needs to be accessible. The “derived d” works as the protected member of the base class is accessible via object of derived class. However, the if want to instantiate object of base class by writing “base b” is not possible as it’s not accessible. Another way of saying it is that the base() constructor is protected, and that means only the base class and any class derived from base class can call and use the base() constructor. Hence it’s not permitted to call it in main() and it could only be accessed through an object of the derived class.

If want to instantiate object of base class by writing “base b”, she can change the protected in base class to public. However, this could be a violation of data encapsulation.

Question 3: Programming Language Concepts (30 points)

Each of the following is worth 5 points.

1. **Just-in-Time Compilers:** Describe the purpose of Just-in-Time (JIT) compilers, i.e., in what context are they used and why? JIT compilers often use their run time heap to store and transform code. Why would they make such a design choice of storing and transforming code in a place where data is typically stored? What is the difference between data and code anyway?

   The normal, traditional compilers do the compilation process before the program is ran, however, Just-in-Time compilers do the compilation process after the program is ran and it compiles the code on the fly. JVM and CLR uses JIT compilers. The purpose of using JIT compilers is the efficiency of this would overcome the non-JIT compilers, and it could reduce the burden of the CPU. The difference between data and code is that data could be extracted from the memory and be used directly, while the codes stored must be compiled first in order for the computer to understand it.

2. **Dynamically-typed vs. statically-typed languages:** Define the terms dynamically-typed and statically-typed languages. What is one advantage of dynamically-typed languages over statically-typed ones? Similarly, what is one advantage of statically-typed languages over dynamically-typed ones?

   Dynamically-typed languages do the type-checking process at run time, and the type of variables are associated with run time values. While Statically-typed languages do the type-checking process at compile time, and the type of variables are known at compile time.

   For Dynamically-typed languages, they are easier to make changes to, and it's simpler than statically-typed languages, there are less type annotations needed.
   And an advantage of statically-typed languages would be that the compiler does the checking and many trivial bugs and mistakes could be found at an early stage.

Each of the following is worth 10 points.

3. **Control-Flow and Call Stack:** Describe how nested function call semantics are implemented using the help of call stacks. As described in class, local variables (whose values cannot be determined to be constants by the compiler) corresponding to a function F are stored on the stack frame for F . Suppose you implemented a compiler to store local variables on the heap. Why would this be a bad design choice?

   The call stack is stored as part of a process' memory. And every time a function A is called by a called B, a frame is pushed into the call stack, it contains registers, stack pointer, and other data relevant to operation of the function B as well as the return address, so that when A has finished executing, the execution of code can return/jump to the instruction right after the call to A.
   The use of call stacks makes it simple to keep track of the stack, and it enables the computer to run the code in a top-to-bottom fashion sequentially. If implement a compiler to store local variables on the heap, there's no enforced patter to the assignment of blocks from the heap, the block can be allocated and freed at any time. In this case, it would be harder to keep track of which part of the heap is freed/occupied at a given time.

4. **Turing Completeness:** Say you are given a language description that consists of basic arithmetic operations, jump instructions, and memory load/store. Is such a language Turing-complete? Does there exist a Turing-complete language that does not explicitly support basic arithmetic and jump operations, but can simulate it nonetheless?

   Yes, as it matches the required actions that a Turing-complete language needs, which are jump instructions that could implement loops and/or recursions, the ability to read/write from/to memory, and the ability to do arithmetic operations. And any program written in this language can be converted into an equivalent Turing Machine and vice versa.

   Yes, the language of Turing Machine, although it doesn't support basic arithmetic and jump operations, it could simulate the basic arithmetic, memory load/store, and jump operations.