HW 1

Syntax

Grammars

For each of the following three grammars:

1. Briefly describe the language defined by the grammar.
2. State whether the language is ambiguous, and if so, submit a single string and two corresponding parse trees.

1

$S \rightarrow 0\ S\ 1 \mid 0\ 1$

✓ **Answer**

This essentially means that S can be 01, or 0 followed by any S followed by 1.

This practically means that S is any n amount of 0s followed by n amount of 1s.

This language is not ambiguous. All pairs must be parsed from outside in, there are no other ways to parse this tree, or inner S's will fail to parse.

2

$S \rightarrow a\ S\ b\ S \mid b\ S\ a\ S \mid \epsilon$

✓ **Answer**

This means that S can be nothing, or specific combinations of a and b such that there is an equal amount of a's and b's.

This language is ambiguous.
$abab$ for example may be parsed $(a\epsilon b(a\epsilon b\epsilon))$ or $(a(b\epsilon a\epsilon)b\epsilon)$
Where $()$ represent a recursive node in the parse tree.

3

$S \rightarrow S + S \mid S * S \mid (S) \mid \text{id}$

✓ **Answer**

This is basic arithmetic syntax for the $+$ and $*$ operators. It matches any expression containing $+$, $*$, balanced pairs of parenthesis $()$, and numbers.

This grammar is ambiguous because there is no order of operations.
$1 + 2 * 3$ could be parsed as $\langle\langle 1 + 2\rangle * 3\rangle$ or as $\langle 1 + \langle 2 * 3\rangle\rangle$
Where $\langle\rangle$ represent a recursive node in the parse tree.

Syntax-Directed Translation

Construct a syntax-directed translation scheme that translates arithmetic expressions from infix notation to prefix notation in which an operator appears before its operands; e.g., $-xy$ is the prefix notation for $x - y$. Give annotated parse trees for the inputs $9 - 5 + 2$ and $9 - 5 * 2$.

✓ **Answer**

```
digit -> \d+
expr -> digit { print(digit) }
    | { print("(") } (expr) { print(")" }
    | { print("+") } expr + expr
    | { print("-") } expr - expr
    | { print("*") } expr * expr
    | { print("/") } expr / expr
```

```
expr \─── { print("+") }
     ├─── expr \─── { print("-") }
     |         ├─── expr \─── digit ──── 9
     |         |         ├─── { print(9) }
     |         ├─── [-]
     |         ├─── expr \─── digit ──── 5
     |                   ├─── { print(5) }
     ├─── [+]
     ├─── expr \─── digit ──── 2
               ├─── { print(2) }

-> + - 9 5 2
```

```
expr \─── { print("-") }
     ├─── expr \─── digit ──── 9
     |         ├─── { print(9) }
     ├─── [-]
```

```
      ├── expr \── { print("*") }
                ├── expr \── digit ─── 5
                |         ├── { print(5) }
                ├── [*]
                ├── expr \── digit ─── 2
                          ├── { print(2) }

→ - 9 * 5 2
```

Static Scope

Static Scope

For the block-structured C code, indicate the values assigned to w, x, y, and z.

a

```
int w, x, y, z;
int i = 4; int j = 5;
{
    int j = 7;
    i = 6;
    w = i + j;
}
x = i + j;
{
    int i = 8;
    y = i + j;
}
z = i + j;
```

✓ Answer

```
int w, x, y, z;          // i, j,  w,  x,  y,  z
int i = 4; int j = 5;    // 4, 5, 00, 00, 00, 00
{
    int j = 7;           // 4, 7, 00, 00, 00, 00
    i = 6;               // 6, 7, 00, 00, 00, 00
    w = i + j;           // 6, 7, 13, 00, 00, 00
}
x = i + j;               // 6, 5, 13, 11, 00, 00
```

```
{
    int i = 8;           // 8, 5, 13, 11, 00, 00
    y = i + j;           // 8, 5, 13, 11, 13, 00
}
z = i + j;               // 6, 5, 13, 11, 13, 11
```

$w, x, y, z = 13, 11, 13, 11$

b

```
int w, x, y, z;
int i = 3; int j = 4;
{
    int i = 5;
    w = i + j;
}
x = i + j;
{
    int j = 6;
    i = 7;
    y = i + j;
}
z = i + j;
```

✓ **Answer**

```
int w, x, y, z;          // i, j,  w,  x,  y,  z
int i = 4; int j = 5;    // 4, 5, 00, 00, 00, 00
{
    int j = 7;           // 4, 7, 00, 00, 00, 00
    i = 6;               // 6, 7, 00, 00, 00, 00
    w = i + j;           // 6, 7, 13, 00, 00, 00
}
x = i + j;               // 6, 5, 13, 11, 00, 00
{
    int i = 8;           // 8, 5, 13, 11, 00, 00
    y = i + j;           // 8, 5, 13, 11, 13, 00
}
z = i + j;               // 6, 5, 13, 11, 13, 11
```

$w, x, y, z = 9, 7, 13, 11$

Symbol Table

Implement a hierarchal environment implemented as a *chained symbol table* (Figure 2.36 and 2.37). The table key is a string representing an id, and the value is unused for the time being. Create a function to print the contents of a chained symbol table.

✓ Answer

```
from typing import Generic, TypeVar, Self

T = TypeVar("T")

def print_level(str: str, level: int = 0) -> None:
    print("    "*level + str)
    return

class CST(Generic[T]):
    def __init__(self, parent: Self | None = None) -> None:
        """Creates a base CST with no values"""
        self.table: dict[str, T] = {}
        self.parent: Self | None = parent
        self.children: list[CST[T]] = []

    def put(self, id: str, val: T) -> Self:
        """Insert an id into the current table."""
        self.table[id] = val
        return self

    def set(self, id: str, val: T) -> Self | None:
        """Set an already created id to a specific value."""
        env = self
        while env is not None:
            if id in env.table:
                env.table[id] = val
                return env
            env = env.parent
        return None

    def get(self, id: str) -> T | None:
        """Search for nearest definitions of id in currennt and
enclosing environments."""
```

```python
        env = self
        while env is not None:
            if id in env.table:
                return env.table[id]
            env = env.parent
        return None

    def make_child(self) -> Self:
        """Create a child for nested environments"""
        child = self.__class__(self)
        self.children.append(child)
        return child

    def print(self, level: int = 0):
        """Print all entries in the current environment."""
        print_level("Symbol Table:", level)
        for id in self.table:
            print_level(f"{id}: {self.table[id]}", level)
        print_level("--- End of Table ---", level)

    def print_down(self, level: int = 0):
        """Print this environment and all children."""
        self.print(level)
        for child in self.children:
            child.print_down(level + 1)

    def print_up(self) -> None:
        """Print this environment and all parents"""
        self.print()
        if self.parent:
            self.parent.print_up()
```