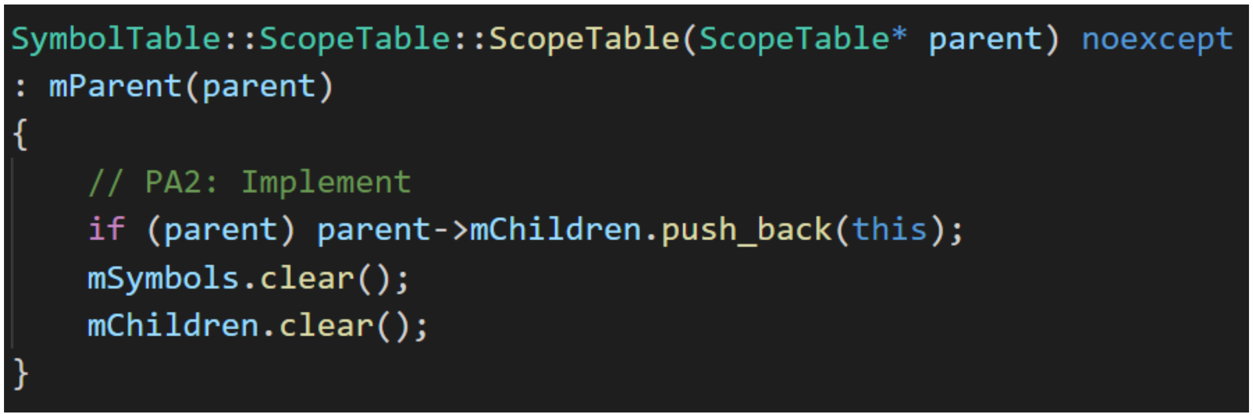
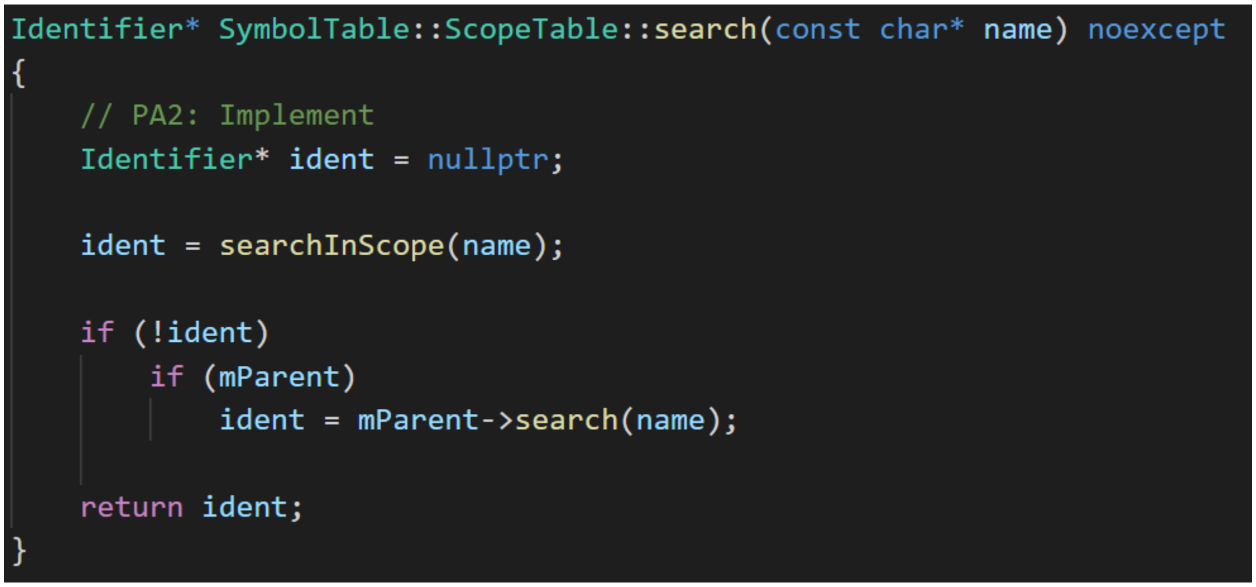
The following are my code snippets and comments for each parse-function I implemented. Since there are a lot of modifications based on original code, I just list a few functions that make me struggle while debugging.

- **Symbols.cpp/ScopeTable()**
  The function is the constructor of **ScopeTable** and create elements in it. Before figuring out to add current table its parent's children list, I was confused by my abnormal printed symbol table.



```
SymbolTable::ScopeTable::ScopeTable(ScopeTable* parent) noexcept
: mParent(parent)
{
    // PA2: Implement
    if (parent) parent->mChildren.push_back(this);
    mSymbols.clear();
    mChildren.clear();
}
```

- **Symbols.cpp/search(name)**
  The function searches for the corresponding identifier through traveling the linked scope tables. I did not realize I can use "mParent" variable to DFS the achieve the goal in the first time.



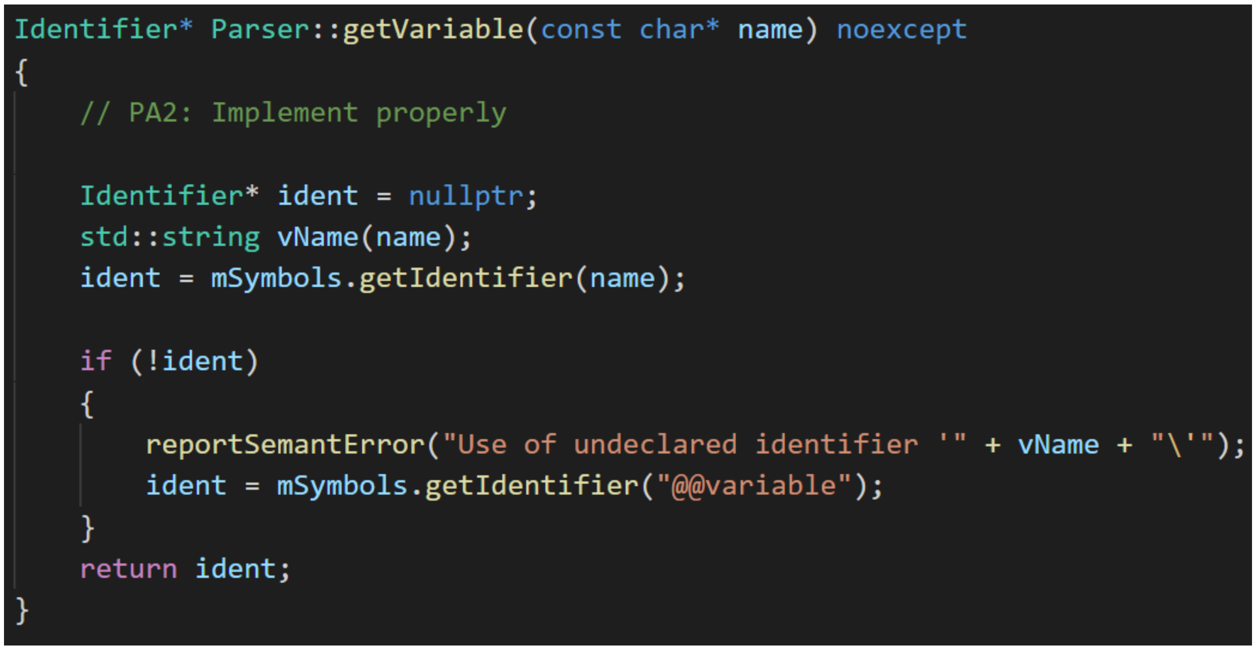
```
Identifier* SymbolTable::ScopeTable::search(const char* name) noexcept
{
    // PA2: Implement
    Identifier* ident = nullptr;

    ident = searchInScope(name);

    if (!ident)
        if (mParent)
            ident = mParent->search(name);

    return ident;
}
```

- **Parse.cpp/getVariable(name)**
  This function attempts to get the corresponding identifier first. If the returned pointer is a null pointer, it reports a semantic error and returns the dummy variable instead. I misunderstood the meaning of "return the identifier for the dummy @@variable", resulting in using **createIdentifier()** instead.

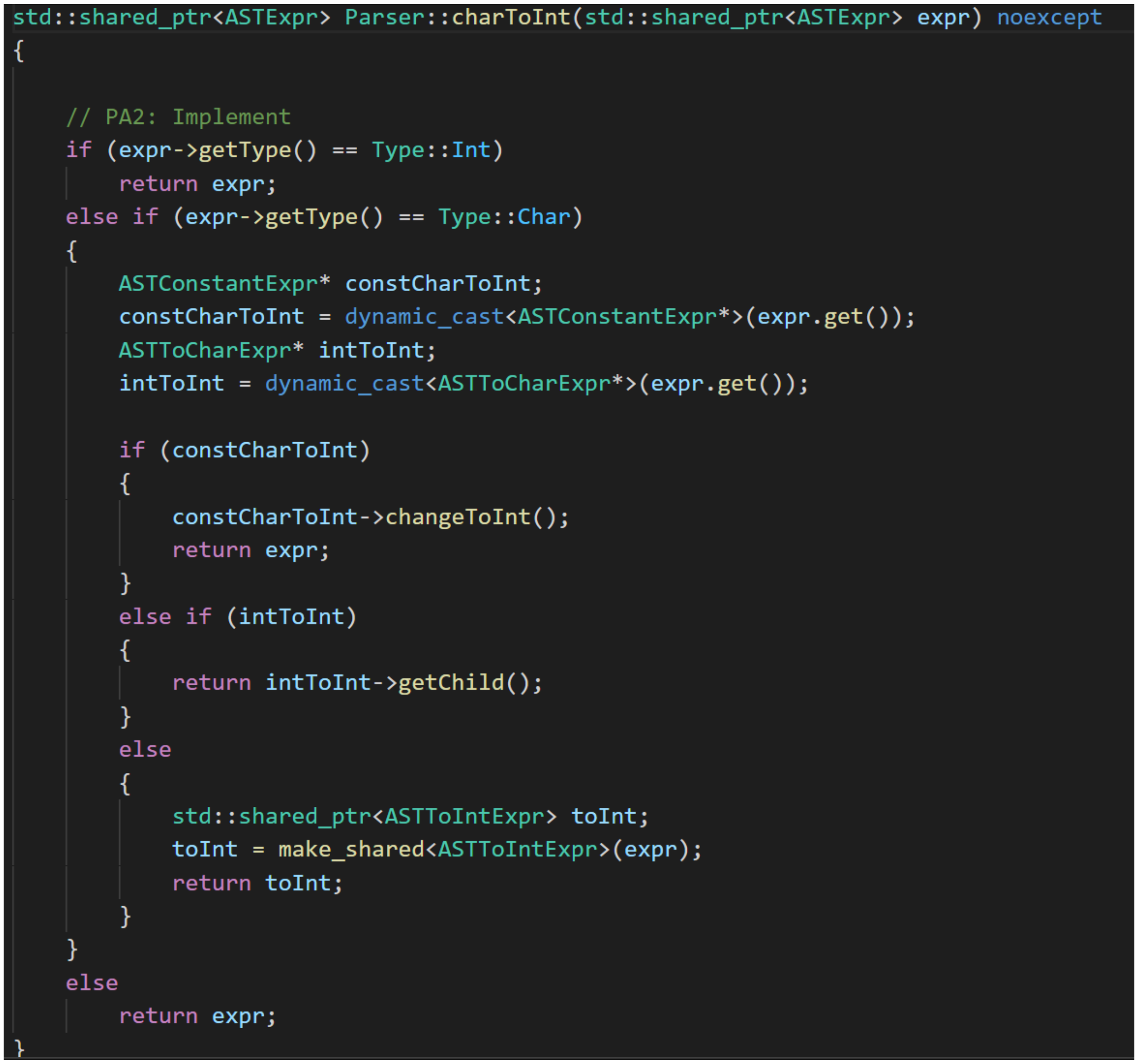


```
Identifier* Parser::getVariable(const char* name) noexcept
{
    // PA2: Implement properly

    Identifier* ident = nullptr;
    std::string vName(name);
    ident = mSymbols.getIdentifier(name);

    if (!ident)
    {
        reportSemantError("Use of undeclared identifier '" + vName + "\'");
        ident = mSymbols.getIdentifier("@@variable");
    }
    return ident;
}
```

- **Parse.cpp/charToInt(), intToChar()**
  These two functions attempt to convert expressions' types. It is my first time to use **dynamic_cast**, which causes me more efforts to learn how to use it. I just post one of these two functions because they are very similar.



```
std::shared_ptr<ASTExpr> Parser::charToInt(std::shared_ptr<ASTExpr> expr) noexcept
{

    // PA2: Implement
    if (expr->getType() == Type::Int)
        return expr;
    else if (expr->getType() == Type::Char)
    {
        ASTConstantExpr* constCharToInt;
        constCharToInt = dynamic_cast<ASTConstantExpr*>(expr.get());
        ASTToCharExpr* intToInt;
        intToInt = dynamic_cast<ASTToCharExpr*>(expr.get());

        if (constCharToInt)
        {
            constCharToInt->changeToInt();
            return expr;
        }
        else if (intToInt)
        {
            return intToInt->getChild();
        }
        else
        {
            std::shared_ptr<ASTToIntExpr> toInt;
            toInt = make_shared<ASTToIntExpr>(expr);
            return toInt;
        }
    }
    else
        return expr;
}
```