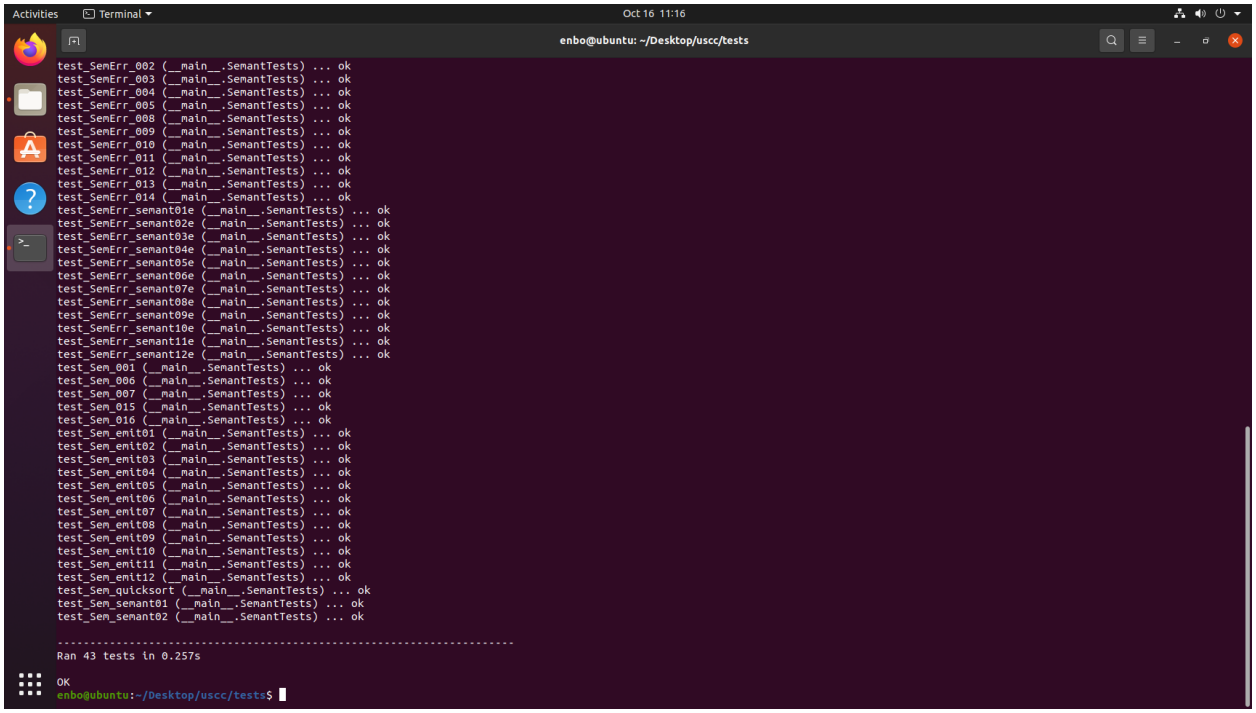
CSE 504 Compiler Design PA-02 Report
Name: Enbo Yu
ID: 113094714
Date: 16 Oct

I. Test Result:



Ran 43 tests in 0.257s

OK

II. Class Details

ASTExpr.cpp

-finalizeOp:

There are four sub-class which are helped by the function of finalizeOP. These functions just need to set the type of the node to integer, and return true if both lhs and rhs are integers

```cpp
bool ASTLogicalAnd::finalizeOp() noexcept
{
    if (mLHS->getType() == Type::Int && mRHS->getType() == Type::Int)
    {
        this->mType = Type::Int;
        return true;
    }
    return false;
}

bool ASTLogicalOr::finalizeOp() noexcept
{
    if (mLHS->getType() == Type::Int && mRHS->getType() == Type::Int)
    {
        this->mType = Type::Int;
        return true;
    }
    return false;
}

bool ASTBinaryCmpOp::finalizeOp() noexcept
{
    if ((mLHS->getType() == Type::Int || mLHS->getType() == Type::Char) &&
        (mRHS->getType() == Type::Int || mRHS->getType() == Type::Char))
    {
        this->mType = Type::Int;
        return true;
    }
    return false;
}

bool ASTBinaryMathOp::finalizeOp() noexcept
{
    if ((mLHS->getType() == Type::Int || mLHS->getType() == Type::Char) &&
        (mRHS->getType() == Type::Int || mRHS->getType() == Type::Char))
    {
        this->mType = Type::Int;
        return true;
    }
    return false;
}
```

Symbols.cpp

-ScopeTable:

By the function of ScopeTable, we can create the elements in the constructor of ScopeTable. After printing the symbol table, the current table elements will be added in its parents' children lists.

-**search:**

By the function of Search, we can search for the analogous or equivalent identifier and go through the following linked scope tables.

```cpp
// Exits the current scope and moves the current scope back to
// the previous scope table.
void SymbolTable::exitScope()
{
    mCurrScope = mCurrScope->getParent();
}

SymbolTable::ScopeTable::ScopeTable(ScopeTable* parent) noexcept
: mParent(parent)
{
    if (parent)
        parent->mChildren.push_back(this);
}

SymbolTable::ScopeTable::~ScopeTable() noexcept
{
    // mParent->mChildren.remove(this);

    for (auto& it : mSymbols)
        delete it.second;

    for (auto& it : mChildren)
        delete it;

    mSymbols.clear();
    mChildren.clear();
}

// Adds the requested identifier to the table
void SymbolTable::ScopeTable::addIdentifier(Identifier* ident)
{
    mSymbols.insert(std::make_pair(ident->getName(), ident));
}

```

```cpp
// Searches this scope for an identifier with
// the requested name. Returns nullptr if not found.
Identifier* SymbolTable::ScopeTable::searchInScope(const char* name) noexcept
{
    auto result  = mSymbols.find(name);
    if (result != mSymbols.end())
        return result->second;
    return nullptr;
}

// Searches this scope first, and if not found searches
// through parent scopes. Returns nullptr if not found.
Identifier* SymbolTable::ScopeTable::search(const char* name) noexcept
{
    Identifier *ident = nullptr;
    ScopeTable *scope = this;
    while (scope != nullptr) {
        ident = scope->searchInScope(name);
        if (ident)
            break;
        scope = scope->mParent;
    }
    return ident;
}

```

Parse.cpp

-getVariable:

By the function of getVariable, we can check the following input identifier at first. When we get a null pointer, this function will return the error that Use of undeclared identifier… Also, the dummy variable will be returned as a substitute. Which means that this function should first try to get the identifier from the symbol table, and return the identifier if it's found.

-charToInt() and intToChar:

We can use these two functions to transform the type of input, like char to int or int to char. For example, if the expression passed to charToInt is already an integer, we can just return the expression directly. Otherwise, if it receives an expression of type char

```
Identifier* Parser::getVariable(const char* name) noexcept
{
    Identifier *ident = mSymbols.getIdentifier(name);
    if (ident == nullptr)
    {
        std::string msg = "Use of undeclared identifier '";
        msg += name;
        msg += "\'";
        reportSemantError(msg);
        ident = mSymbols.getIdentifier("@@variable");
    }
    return ident;
}

const char* Parser::getTypeText(Type type) const noexcept
{
    switch (type)
    {
        case Type::Char:
            return "char";
        case Type::Int:
            return "int";
        case Type::Void:
            return "void";
        case Type::CharArray:
            return "char[]";
        case Type::IntArray:
            return "int[]";
        case Type::Function:
            return "function";
    }
}
```

```cpp
std::shared_ptr<ASTExpr> Parser::charToInt(std::shared_ptr<ASTExpr> expr) noexcept
{
    std::shared_ptr<ASTExpr> retVal = expr;

    if (expr->getType() == Type::Char)
    {
        ASTConstantExpr *constExpr = dynamic_cast<ASTConstantExpr*>(expr.get());
        if (constExpr)
        {
            constExpr->changeToInt();
        }
        else
        {
            ASTToIntExpr* toCharExpr = dynamic_cast<ASTToIntExpr*>(expr.get());
            if (toCharExpr)
                return toCharExpr->getChild();
            return make_shared<ASTToIntExpr>(expr);
        }
    }
    return retVal;
}

```

```cpp
std::shared_ptr<ASTExpr> Parser::intToChar(std::shared_ptr<ASTExpr> expr) noexcept
{
    std::shared_ptr<ASTExpr> retVal = expr;

    if (expr->getType() == Type::Int)
    {
        ASTConstantExpr* constExpr = dynamic_cast<ASTConstantExpr*>(expr.get());
        if (constExpr)
        {
            constExpr->changeToChar();
        }
        else
        {
            ASTToCharExpr* toIntExpr = dynamic_cast<ASTToCharExpr*>(expr.get());
            if (toIntExpr)
                return toIntExpr->getChild();
            return make_shared<ASTToCharExpr>(expr);
        }
    }

    return retVal;
}

```