Lab 1 – Compiler design (BCSE307P)

Submitted by: **Sakshi Biyani**

Registration no: **22BLC1385**

Aim: The aim of this program is to validate C++ identifiers by checking them against language rules. It provides feedback on identifier validity, helping users adhere to naming conventions and reduce syntax errors in their code.

Identifier Validation Criteria

When classifying an identifier as valid or invalid, the following rules must be adhered to ensure consistency and correctness:

1. **Starting Character**:
   The first character of the identifier must be an underscore (_), an uppercase letter (`A-Z`), or a lowercase letter (`a-z`).
2. **Subsequent Characters**:
   After the first character, the identifier may contain uppercase letters (`A-Z`), lowercase letters (`a-z`), digits (`0-9`), but must not include any whitespace or special characters.
3. **No Repeating Underscores**:
   The identifier must not contain consecutive underscore characters. This ensures clarity and avoids potential confusion.
4. **Reserved Keywords**:
   The identifier must not be a reserved keyword in the language. Reserved keywords have predefined meanings and uses, which could lead to conflicts if used as identifiers.
5. **Case Sensitivity**:
   Identifiers are case-sensitive. For example, `Variable` and `variable` would be treated as distinct identifiers.
6. **Unique from Function and Class Names**:
   The identifier must not be the same as any existing function or class name. This helps avoid naming conflicts within the codebase.
7. **English Characters Only**:
   The identifier must exclusively contain English alphabet characters and cannot include characters from other languages. This promotes consistency and compatibility.
8. **No Numeric Hyphenation**:
   Numeric characters should not be immediately followed by hyphens (-) or underscores (_). This avoids confusion and ensures readability.
9. **Length Constraints**:
   The identifier must be at least 1 character long and no longer than 20 characters. This ensures that identifiers are neither too short to be meaningful nor too long to be cumbersome.

Examples

Valid Identifiers:

1. userAge (valid)
2. _tempValue (valid)
3. itemCount2023 (valid)
4. average_score (valid)

Invalid Identifiers:

1. 9thElement (starts with digit)
2. my__var (repeating underscores)
3. if (reserved keyword)
4. sqrt (existing function name)
5. data%value (invalid character %)
6. var#name (invalid character #)
7. thisIdentifierIsWayTooLong (exceeds length limit)
8. 变量 (contains non-English characters)

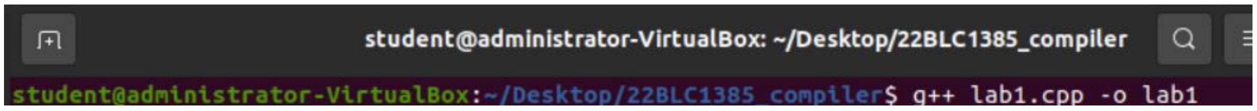
CODE:

```
#include <iostream>     // For input/output operations

#include <string>       // For using string class

#include <set>          // For set data structure

#include <regex>        // For regex validation

#include <cctype>       // For character-based checks

#include <algorithm>    // Additional utilities (if needed)

#include <vector>       // For potential list handling

using namespace std;



// Reserved keywords in C++

set<string> reserved_keywords = {

    "auto", "break", "case", "char", "const", "continue", "default", "do",

    "double", "else", "enum", "extern", "float", "for", "goto", "if",
    "int", "long", "register", "return", "short", "signed", "sizeof",
    "static", "struct", "switch", "typedef", "union", "unsigned", "void",
    "volatile", "while"
};

// Predefined function/class names
set<string> existing_function_class_names = {
    "main", "print", "calculate"
};

// Validates the identifier
bool isValidIdentifier(const string &identifier) {
    // Check length
    if (identifier.length() < 1 || identifier.length() > 20) {
        cout << "Invalid: Length is " << identifier.length() << endl;
        return false;
    } else {
        cout << "Length check passed." << endl;
    }

    // Check first character
    if (!isalpha(identifier[0]) && identifier[0] != '_') {
        cout << "Invalid: First character is not a letter or underscore." << endl;
        return false;
    } else {
```

```
    cout << "First character check passed." << endl;
}

// Check character rules
for (size_t i = 1; i < identifier.length(); ++i) {
    if (!isalnum(identifier[i]) && identifier[i] != '_') {
        cout << "Invalid: Contains special character or whitespace." << endl;
        return false;
    }
    if (identifier[i] == '_' && identifier[i - 1] == '_') {
        cout << "Invalid: Contains repeating underscores." << endl;
        return false;
    }
    if (isdigit(identifier[i - 1]) && (identifier[i] == '_' || identifier[i] == '-')) {
        cout << "Invalid: Contains numeric hyphenation." << endl;
        return false;
    }
}
cout << "No repeating underscores and numeric hyphenation check passed." << endl;

// Check reserved keywords
if (reserved_keywords.find(identifier) != reserved_keywords.end()) {
    cout << "Invalid: Identifier is a reserved keyword." << endl;
    return false;
} else {
        cout << "Reserved keyword check passed." << endl;
    }

    // Check predefined names
    if (existing_function_class_names.find(identifier) != existing_function_class_names.end()) {
        cout << "Invalid: Identifier is an existing function or class name." << endl;
        return false;
    } else {
        cout << "Function/class name check passed." << endl;
    }

    // Regex validation
    regex english_chars("^[A-Za-z0-9_]*$");
    if (!regex_match(identifier, english_chars)) {
        cout << "Invalid: Contains non-English characters." << endl;
        return false;
    } else {
        cout << "English characters check passed." << endl;
    }

    return true; // Valid identifier
}

int main() {
    string identifier;
    // Get identifier from user
    cout << "Enter an identifier: ";
    cin >> identifier;

    // Validate and display result
    cout << "Checking: " << identifier << endl;
    if (isValidIdentifier(identifier)) {
        cout << identifier << " is a valid identifier." << endl;
    } else {
        cout << identifier << " is an invalid identifier." << endl;
    }
    return 0; // Exit program
}
```

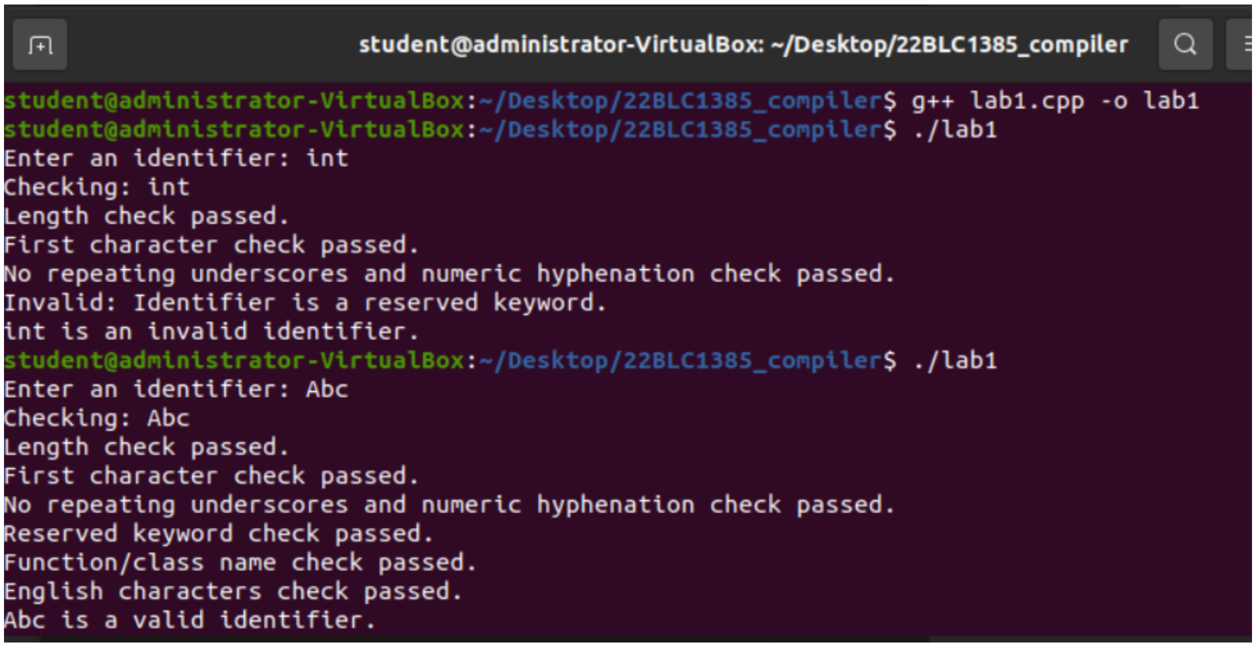
OUTPUT:

Compile the file



```
student@administrator-VirtualBox: ~/Desktop/22BLC1385_compiler
student@administrator-VirtualBox:~/Desktop/22BLC1385_compiler$ g++ lab1.cpp -o lab1
```

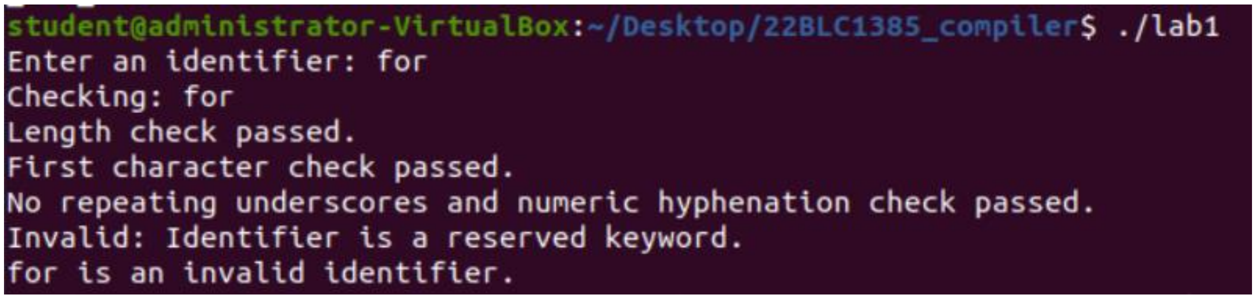
Run the file and test for inputs:



```
student@administrator-VirtualBox: ~/Desktop/22BLC1385_compiler
student@administrator-VirtualBox:~/Desktop/22BLC1385_compiler$ g++ lab1.cpp -o lab1
student@administrator-VirtualBox:~/Desktop/22BLC1385_compiler$ ./lab1
Enter an identifier: int
Checking: int
Length check passed.
First character check passed.
No repeating underscores and numeric hyphenation check passed.
Invalid: Identifier is a reserved keyword.
int is an invalid identifier.
student@administrator-VirtualBox:~/Desktop/22BLC1385_compiler$ ./lab1
Enter an identifier: Abc
Checking: Abc
Length check passed.
First character check passed.
No repeating underscores and numeric hyphenation check passed.
Reserved keyword check passed.
Function/class name check passed.
English characters check passed.
Abc is a valid identifier.
```

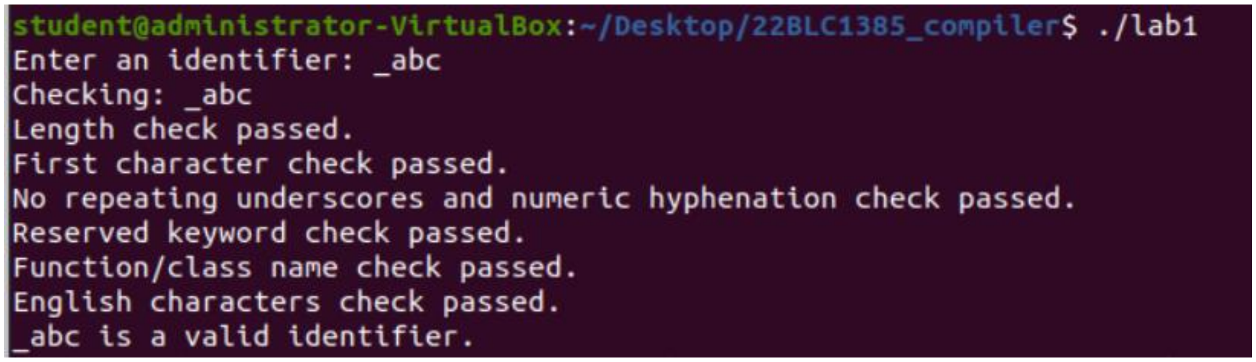
int is an invalid identifier because it is a reserved keyword in C++.

Abc is a valid identifier because it satisfies all the rules.



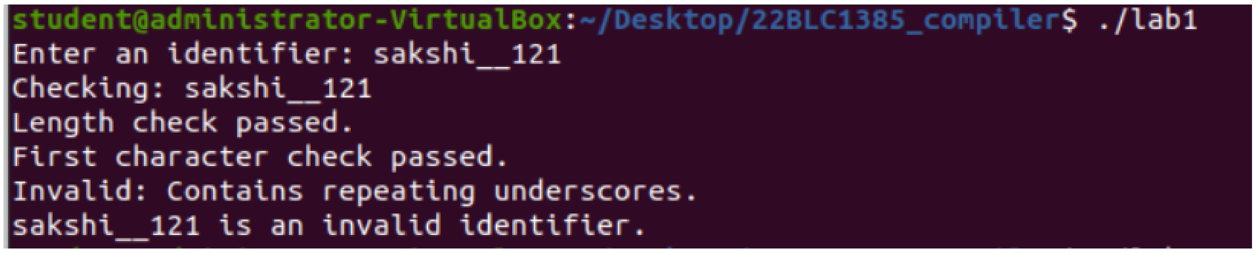
```
student@administrator-VirtualBox:~/Desktop/22BLC1385_compiler$ ./lab1
Enter an identifier: for
Checking: for
Length check passed.
First character check passed.
No repeating underscores and numeric hyphenation check passed.
Invalid: Identifier is a reserved keyword.
for is an invalid identifier.
```

for is an invalid identifier because it is a reserved keyword in C++.



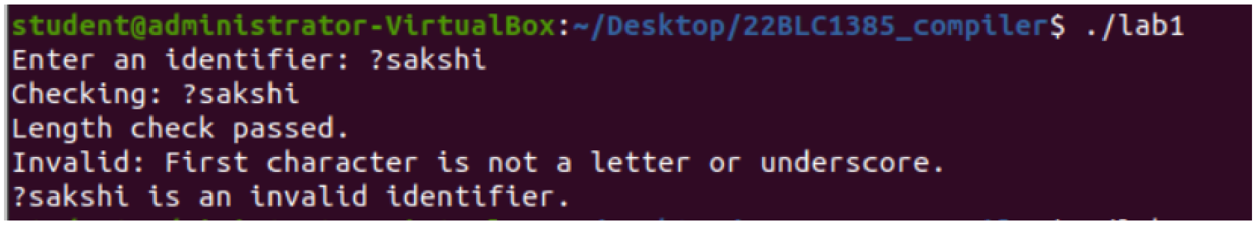
```
student@administrator-VirtualBox:~/Desktop/22BLC1385_compiler$ ./lab1
Enter an identifier: _abc
Checking: _abc
Length check passed.
First character check passed.
No repeating underscores and numeric hyphenation check passed.
Reserved keyword check passed.
Function/class name check passed.
English characters check passed.
_abc is a valid identifier.
```

_abc is a valid identifier because it starts with an underscore, contains no invalid characters, and doesn't violate any rules.



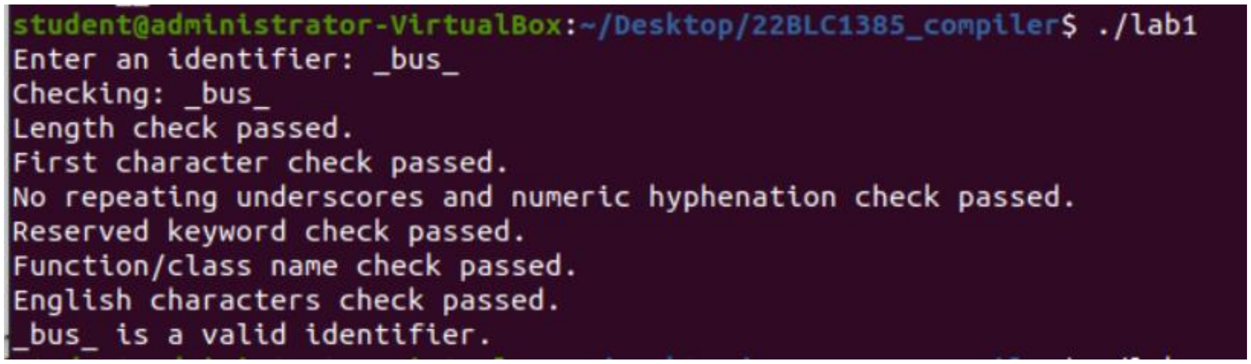
```
student@administrator-VirtualBox:~/Desktop/22BLC1385_compiler$ ./lab1
Enter an identifier: sakshi__121
Checking: sakshi__121
Length check passed.
First character check passed.
Invalid: Contains repeating underscores.
sakshi__121 is an invalid identifier.
```

sakshi__121 is an invalid identifier because it contains consecutive underscores (__), which are not allowed.



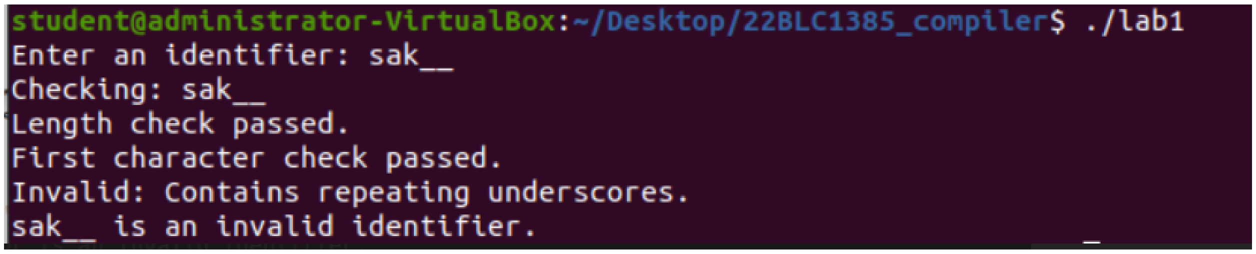
```
student@administrator-VirtualBox:~/Desktop/22BLC1385_compiler$ ./lab1
Enter an identifier: ?sakshi
Checking: ?sakshi
Length check passed.
Invalid: First character is not a letter or underscore.
?sakshi is an invalid identifier.
```

?sakshi is an invalid identifier because the first character is ?, which is not a letter or underscore.



```
student@administrator-VirtualBox:~/Desktop/22BLC1385_compiler$ ./lab1
Enter an identifier: _bus_
Checking: _bus_
Length check passed.
First character check passed.
No repeating underscores and numeric hyphenation check passed.
Reserved keyword check passed.
Function/class name check passed.
English characters check passed.
_bus_ is a valid identifier.
```

- _bus_ starts with an underscore, which is valid.
- It has no repeating underscores or invalid numeric combinations.
- It is not a reserved keyword or predefined name.
- It contains only valid characters.



```
student@administrator-VirtualBox:~/Desktop/22BLC1385_compiler$ ./lab1
Enter an identifier: sak__
Checking: sak__
Length check passed.
First character check passed.
Invalid: Contains repeating underscores.
sak__ is an invalid identifier.
```

- sak__ passes the length and first character checks.
- It fails because it contains consecutive underscores (__), which are invalid according to the rules**.**

Conclusion

- The program provides detailed feedback for each step of the validation process, making it easier for users to understand why an identifier is valid or invalid.
- It ensures compliance with common C++ identifier rules and includes checks for both syntax and semantics.
- This makes the program a practical tool for beginners learning programming concepts or for developers wanting to ensure identifier validity in their code.

By combining step-by-step validation with clear output messages, the program serves as an educational and functional tool for working with identifiers in C++.