COMPILER DESIGN LAB ASSIGNMENT-4

Name: AYUSH MONDAL

Class: BCSE-III

Section: A2

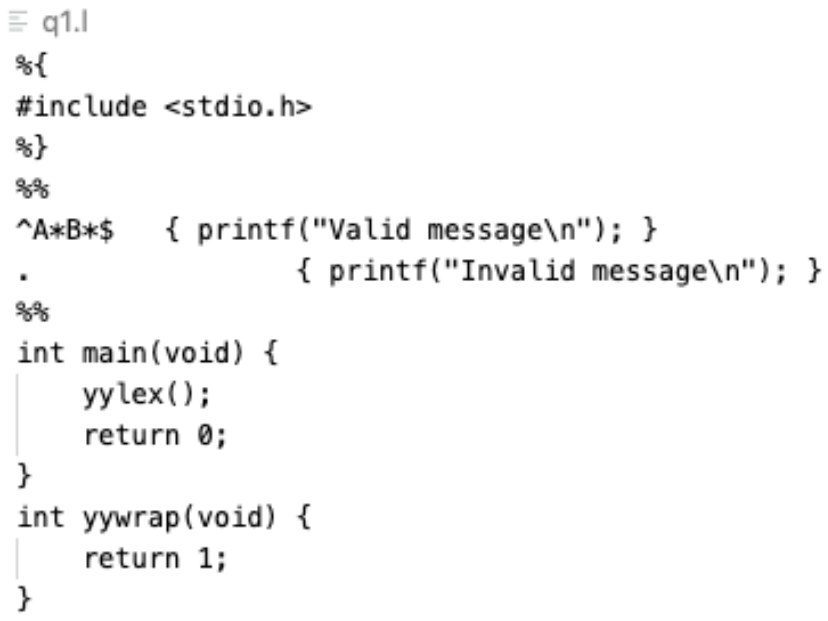
Roll Number: 002210501066

Assignment Number: 4

QUESTION 1:

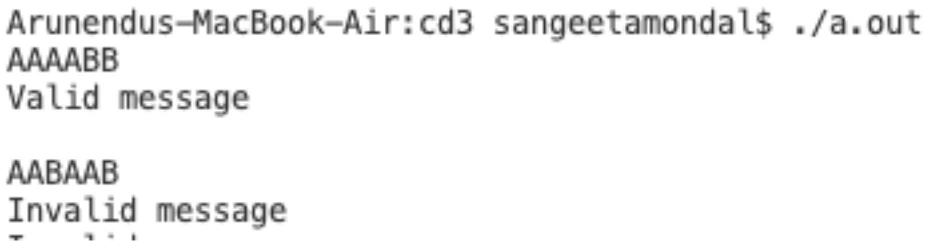
Design a grammar to recognise a string of the form AA...ABB...B, i.e. any number of As followed by any number of Bs. Use LEX or YACC to recognise it. Which one is a better option?

USING LEX:



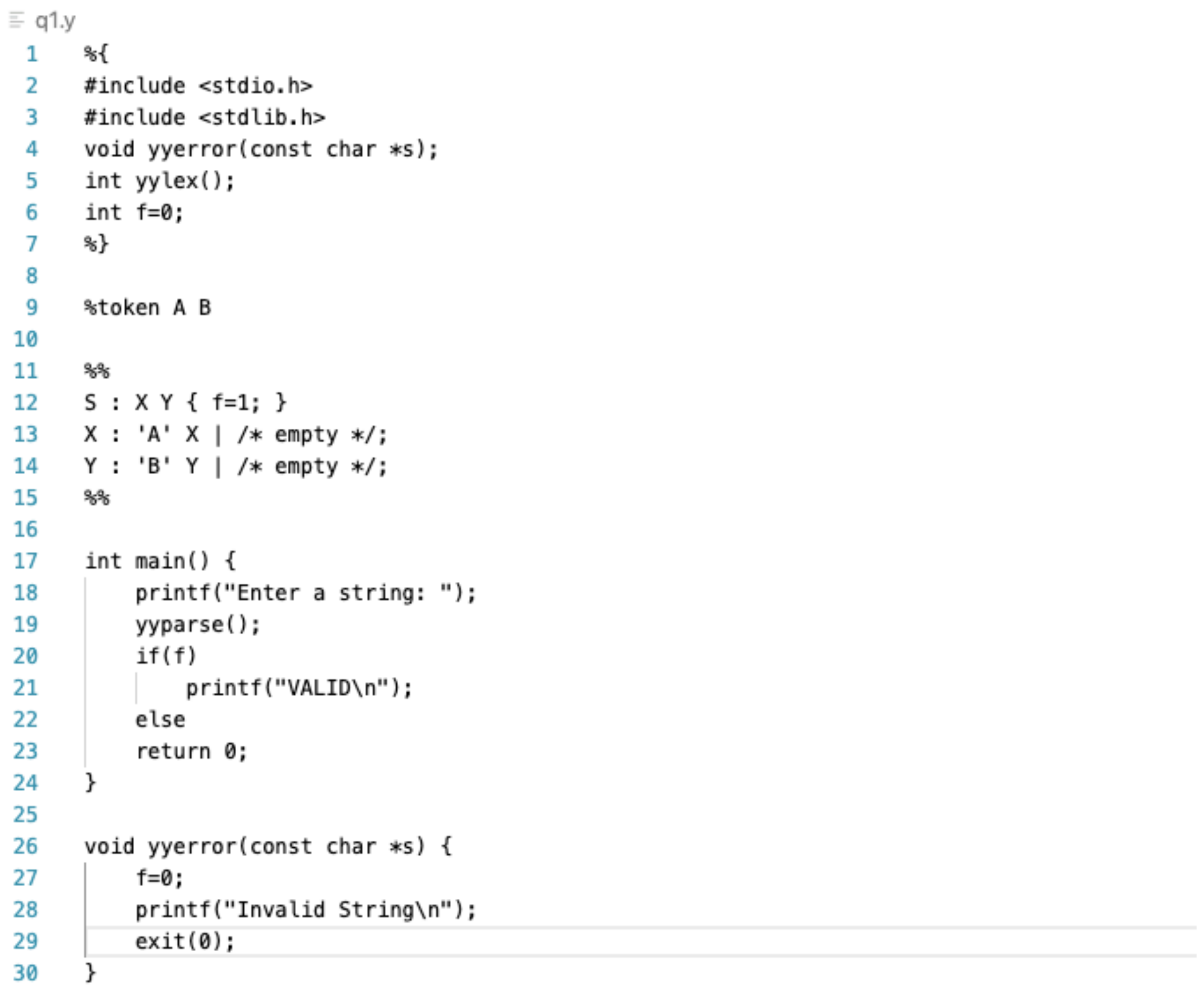
```
q1.l
%{
#include <stdio.h>
%}
%%
^A*B*$   { printf("Valid message\n"); }
.                { printf("Invalid message\n"); }
%%
int main(void) {
    yylex();
    return 0;
}
int yywrap(void) {
    return 1;
}
```

OUTPUT:

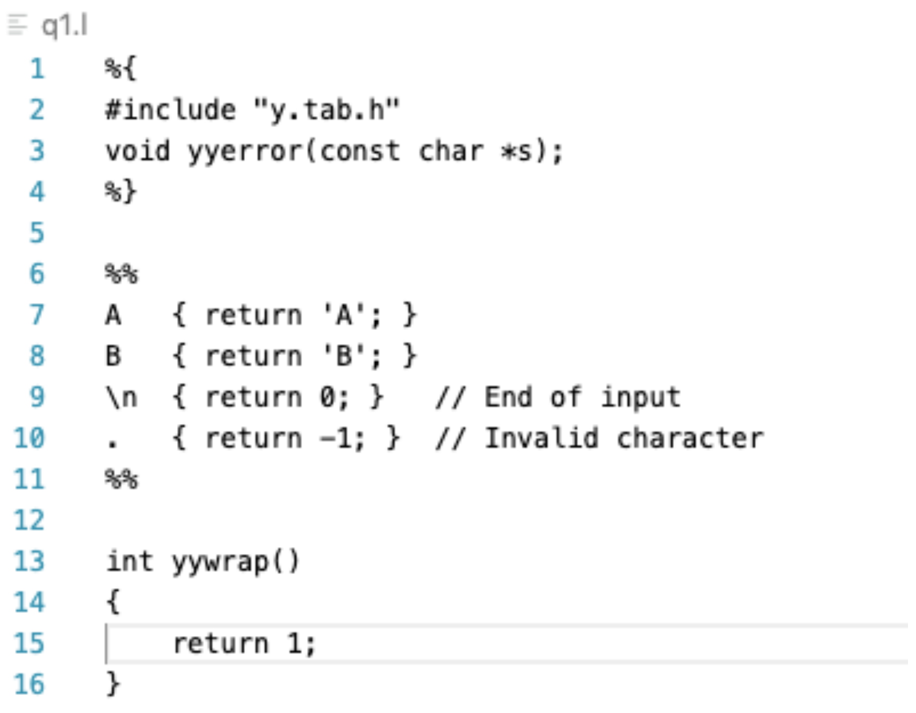


```
Arunendus-MacBook-Air:cd3 sangeetamondal$ ./a.out
AAAABB
Valid message

AABAAB
Invalid message
```

USING YACC:

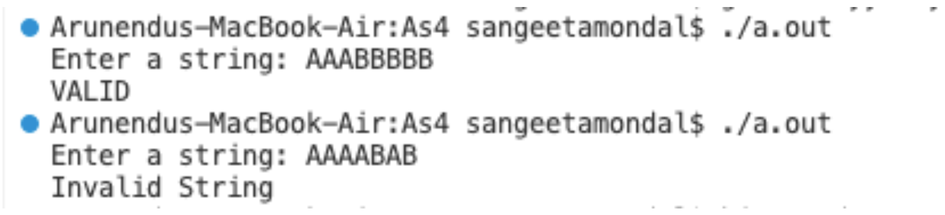


```
%{
#include <stdio.h>
#include <stdlib.h>
void yyerror(const char *s);
int yylex();
int f=0;
%}

%token A B

%%
S : X Y { f=1; }
X : 'A' X | /* empty */;
Y : 'B' Y | /* empty */;
%%

int main() {
    printf("Enter a string: ");
    yyparse();
    if(f)
        printf("VALID\n");
    else
    return 0;
}

void yyerror(const char *s) {
    f=0;
    printf("Invalid String\n");
    exit(0);
}
```



```
%{
#include "y.tab.h"
void yyerror(const char *s);
%}

%%
A    { return 'A'; }
B    { return 'B'; }
\n   { return 0; }    // End of input
.    { return -1; }   // Invalid character
%%

int yywrap()
{
    return 1;
}
```

OUTPUT:


```
● Arunendus-MacBook-Air:As4 sangeetamondal$ ./a.out
  Enter a string: AAABBBBB
  VALID
● Arunendus-MacBook-Air:As4 sangeetamondal$ ./a.out
  Enter a string: AAAABAB
  Invalid String
```

Which One is a Better Option: LEX or YACC?

When we need to recognize strings of the regex form A*B*, YACC is a better choice than LEX. Here's why:

1. **Grammar Handling**
    - **LEX** works well for simple pattern matching with regular expressions but does not enforce rules about the order of symbols.
    - **YACC** uses grammar rules, which naturally makes sure all A's come before B's.
2. **Clear Division of Tasks**
    - With **LEX**, you would have to manually count the A's and B's, which adds extra work.
    - **YACC** handles this through its grammar rules, making the solution simpler and cleaner.
3. **Easy to Extend and Maintain**
    - If you need to add more rules later (like allowing C's after B's), **YACC** makes it easier to change the grammar.
    - Changing a LEX solution could be harder because you would need to adjust the manual counting logic.

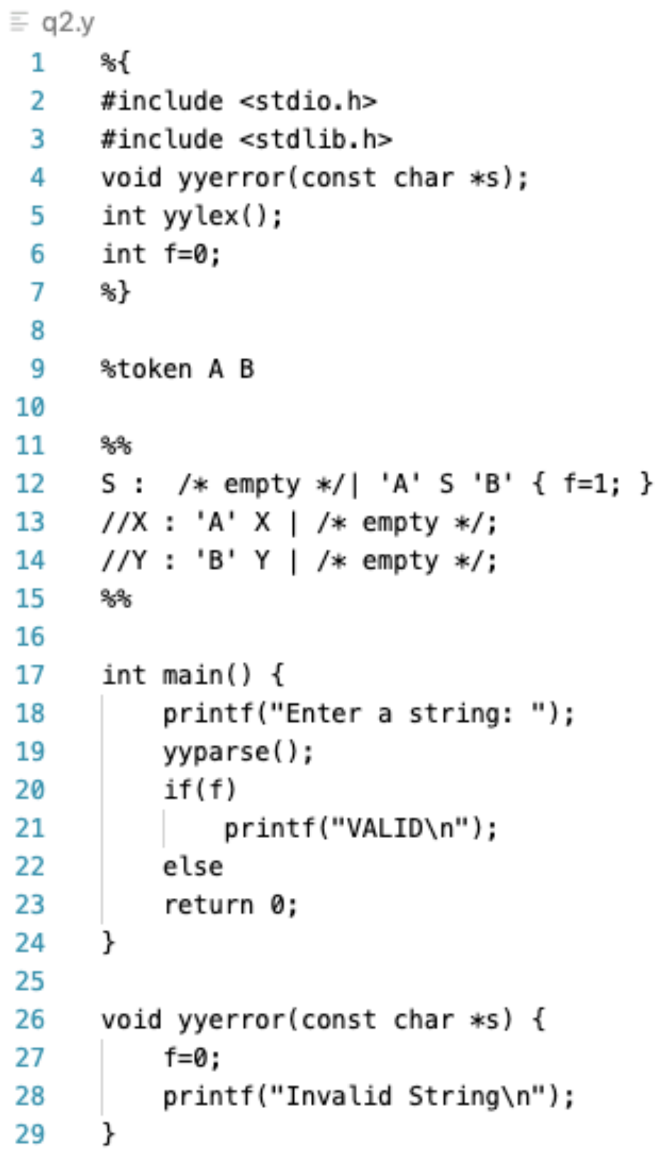
When to Use LEX Instead:

- If you only need to check if a string contains A's and B's without worrying about their order.
- If you need a very fast, character-by-character check, LEX might be the better option.

In Summary:
For checking that a string has some A's followed by some B's, YACC is simpler and cleaner because it naturally follows grammar rules, while LEX is best for simple pattern matching tasks.

QUESTION 2:

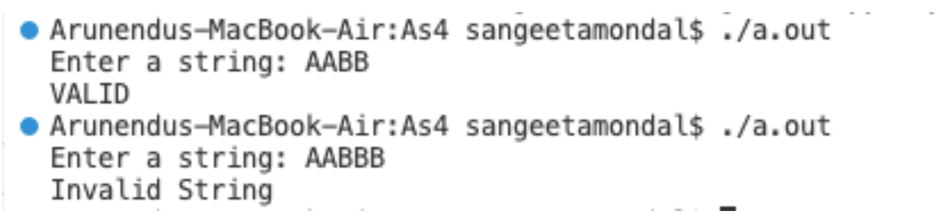
Change your grammar to recognise strings with equal numbers of As and Bs.



q2.y

```
%{
#include <stdio.h>
#include <stdlib.h>
void yyerror(const char *s);
int yylex();
int f=0;
%}

%token A B

%%
S :  /* empty */| 'A' S 'B' { f=1; }
//X : 'A' X | /* empty */;
//Y : 'B' Y | /* empty */;
%%

int main() {
    printf("Enter a string: ");
    yyparse();
    if(f)
        printf("VALID\n");
    else
    return 0;
}

void yyerror(const char *s) {
    f=0;
    printf("Invalid String\n");
}
```

OUTPUT:



```
● Arunendus-MacBook-Air:As4 sangeetamondal$ ./a.out
  Enter a string: AABB
  VALID
● Arunendus-MacBook-Air:As4 sangeetamondal$ ./a.out
  Enter a string: AABBB
  Invalid String
```