**Name: Sparsh Karna**

**Reg Number: 23BDS1172**

**Lab Excercise: 1**

**Aim:**

The aim of this program is to check the validity of an identifier name.

**Procedure:**

1. To check the validity of the identifire name, we have to check for the characters in the name and if the follow the rules.
2. Pre-defined Rules:

**Rule 1:** The identifier name should start with either an alphabet or an underscore(‘\_’).

**Rule 2:** The rest of the name should not contain any special characters other than underscore

**Rule 3:** The identifier should not be from any reserved tokens(eg. Int float etc)

1. Own-Rules:

**Rule 1:** The name should not end with an underscore.

**Rule 2:** Numbers are only allowed at the last of the identifier.

**Rule 3:** If the identifier is starting with a letter it should be capitalised.

**Rule 4:** The length of the identifier should be atleast 3 characters or just an undercore.

**Rule 5:** The identifier must contain te character a.

1. To implement this we can either use Regular Expressions(RegEx) or iterate over the string and check letter by letter.
2. This implementation shows the later method.

**Algorithm:**

1. Take the size of the identifier from the user as an input and store it in a variable.
2. Input the identifier string from the user.
3. Start iterating over the string.

**For pre-defined rules:**

1. Check if the name is a reserved token or not using an array of reserved words.
2. Check the first character if the it is not ‘\_’ or an alphabet, terminate the program and return false.
3. If not then iterate over the string and check each character, if it is not alphanumeric value or a not an underscore terminate the program and return false.
4. If the iteration is over and no rules are violated return true.

**For own rules:**

1. Initialise a variable “flag” to track the exitense of ‘a’ and initialise it as 0.
2. Check the length of the identifier and if it is less than 3 or not 1 with ‘\_’, terminate the program.
3. Check if the name is a reserved token or not using an array of reserved words.
4. Check the first character if the it is not ‘\_’ or an alphabet and if the alphabet is not capitalised, terminate the program and return false.
5. If not then iterate over the string excluding the last cahracter and check each character, if it is not alphabetic value or a not an underscore terminate the program and return false.
6. While iteration if the element is ‘a’ change the value of the above defined variable “flag” to 1.
7. Check the last character of the identifier, if is not a alphanumeric value terminate the program.
8. After the iteration check the value of the flag, if it is 1 then return true else return false.

**Program:**

1. **Pre-Defined Rules:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int isValidIdentifier(char identifier[], int n){

if((!isalpha(identifier[0])) && identifier[0] != '\_') {

printf("%c \n", identifier[0]);

return 0;

};

for(int i = 1; i < n; i++){

if(!(isalnum(identifier[i])) && identifier[i] != '\_'){

printf("%c \n", identifier[i]);

return 0;

};

}

return 1;

}

int main(){

int n;

scanf("%d", &n);

char identifier[n];

scanf("%s", identifier);

int isValid = isValidIdentifier(identifier, n);

if(isValid){

printf("%s is a valid indetifier\n", identifier);

}

else{

printf("%s is not a valid identifier\n", identifier);

}

}

1. **My Own Rules:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int isValidIdentifier(char identifier[], int n)

{

if ((!isalpha(identifier[0])) && identifier[0] != '\_')

{

printf("%c \n", identifier[0]);

return 0;

};

for (int i = 1; i < n; i++)

{

if (!(isalnum(identifier[i])) && identifier[i] != '\_')

{

printf("%c \n", identifier[i]);

return 0;

};

}

if (identifier[n - 1] == '\_')

{

return 0;

}

int hasNumber = 0;

for (int i = 0; i < n; i++)

{

if (isdigit(identifier[i]))

{

hasNumber = 1;

for (int j = i; j < n; j++)

{

if (!isdigit(identifier[j]))

{

return 0;

}

}

break;

}

}

if (isalpha(identifier[0]) && !isupper(identifier[0]))

{

return 0;

}

if (n < 3 && identifier[0] != '\_')

{

return 0;

}

if (n == 1 && identifier[0] != '\_')

{

return 0;

}

int hasA = 0;

for (int i = 0; i < n; i++)

{

if (identifier[i] == 'a' || identifier[i] == 'A')

{

hasA = 1;

break;

}

}

if (!hasA)

{

return 0;

}

if (isReservedWord(identifier))

{

return 0;

}

return 1;

}

int isReservedWord(char identifier[])

{

char reserved[][20] = {"int", "float", "double", "char", "void", "if", "else", "for", "while", "do", "switch", "case", "default", "break", "continue", "return", "struct", "union", "enum", "typedef", "const", "static", "extern", "auto", "register", "signed", "unsigned", "short", "long", "sizeof", "goto"};

int count = 31;

for (int i = 0; i < count; i++)

{

if (strcmp(identifier, reserved[i]) == 0)

{

return 1;

}

}

return 0;

}

int main()

{

int n;

scanf("%d", &n);

char identifier[n];

scanf("%s", identifier);

int len = strlen(identifier);

int isValid = isValidIdentifier(identifier, len);

if (isValid)

{

printf("%s is a valid indetifier\n", identifier);

}

else

{

printf("%s is not a valid identifier\n", identifier);

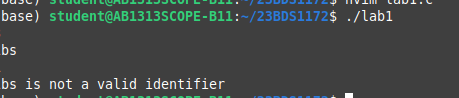
}

}

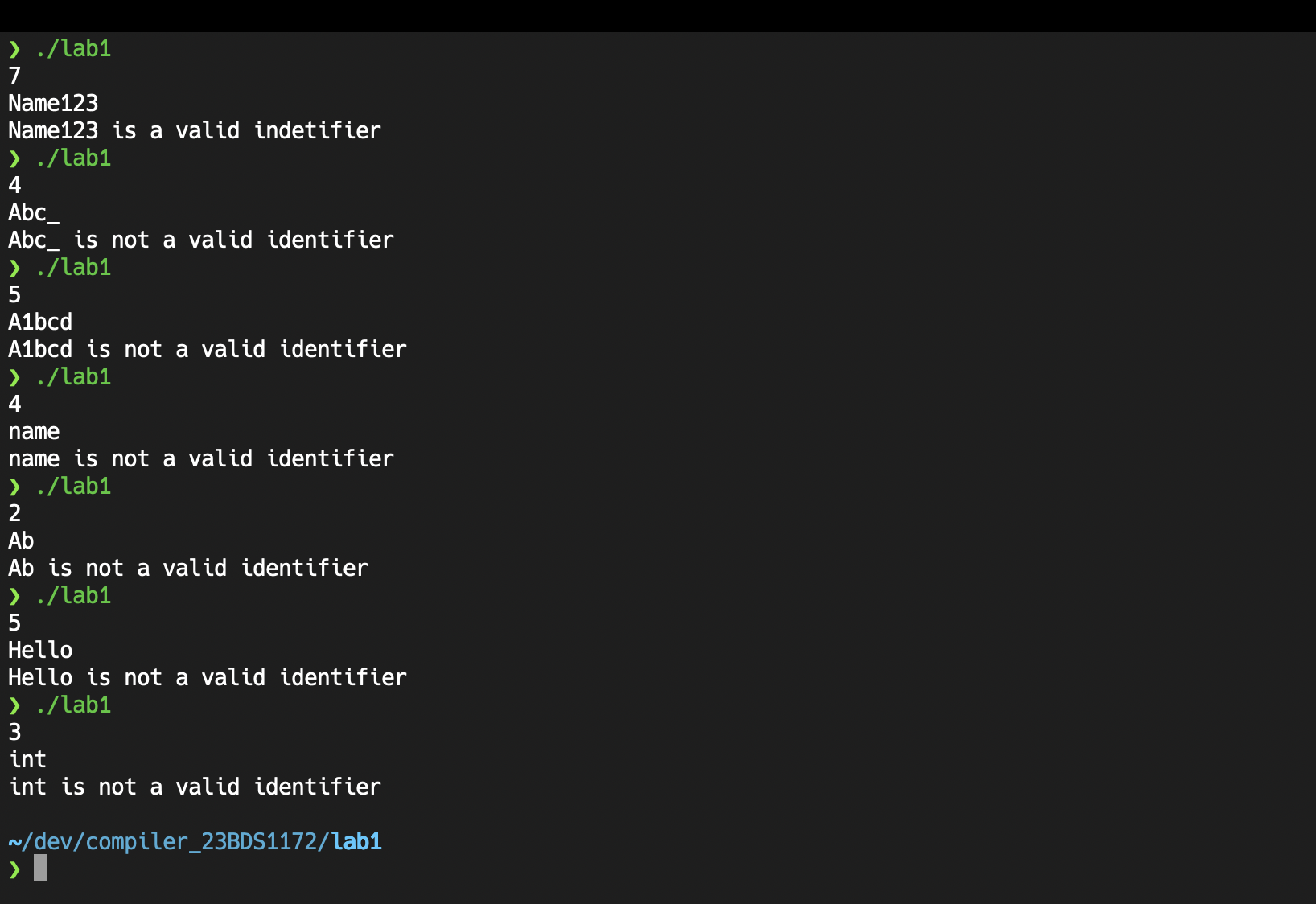
**Output:**

**Pre-Defined Rules:**

****

****

**My Own Rules:**

****

**Result:**

All specified validation rules were successfully implemented and tested, providing a robust identifier validation system suitable for lexical analysis applications.