Formal Languages and Compilers

Lab9

User input in C

- scanf() – function to get user input

```c
#include <stdio.h>

int main()
{
    int num;
    printf("Enter a number: \n");
    scanf("%d", &num);
    int x=num*2;
    printf("The double of your number is: %d\n", x);

    char name[50];
    printf("Enter your name: \n");
    scanf("%s", name);
    printf("Hello %s.", name);

    return 0;
}
```

```
Enter a number:
7
The double of your number is: 14
Enter your name:
Alle
Hello Alle.
```

Pointers

- The memory address is the location of where the variable is stored on the computer – to access it we use &
- A pointer is a variable that stores the memory address of another variable as its value - we create one using *

```c
#include <stdio.h>

int main()
{
    int number = 7;
    int* pointer = &number;
    printf(" Reference->The memory address of the variable: %p", pointer);
    printf(" same as: %p\n", &number);
    printf(" Deference->The value of the variable: %d", *pointer);
    printf(" same as: %d", number);

    return 0;
}
```

input

```
Reference->The memory address of the variable: 0x7ffd708ef72c same as: 0x7ffd708ef72c
Deference->The value of the variable: 7 same as: 7
```

Functions

- Block of code which runs when it's called
- Parameters = variables inside functions
- Arguments = parameters passed to the function
- A function has 2 parts: declaration and definition (the block of code to be executed)
- Declare a function: returnType FunctionName(parameters){}
- void – used when a function should not return a value
- Call a function: inside main-> functionName(arguments);
- When calling a function, it must have the same number of arguments as there are parameters (arguments must be passed in the same order)

Functions - examples

```c
#include <stdio.h>

int myFunction(int x, int y) {
  return x + y;
}

void myFunction2(char l[]){
    printf(" Hello, %s\n", l);
}

void myFunction3(){
    printf(" Function with no value returned\n");
}

int myFunction4(int, int); //declare function

int main() {
  printf(" The result is: %d\n", myFunction(7, 3));
  myFunction2("Alle");
  myFunction3();
  myFunction3();
  int result = myFunction4(5, 3);
  printf(" The result is: %d\n", result);

  return 0;
}

int myFunction4(int x, int y) {//define function
  return x - y;
}
```

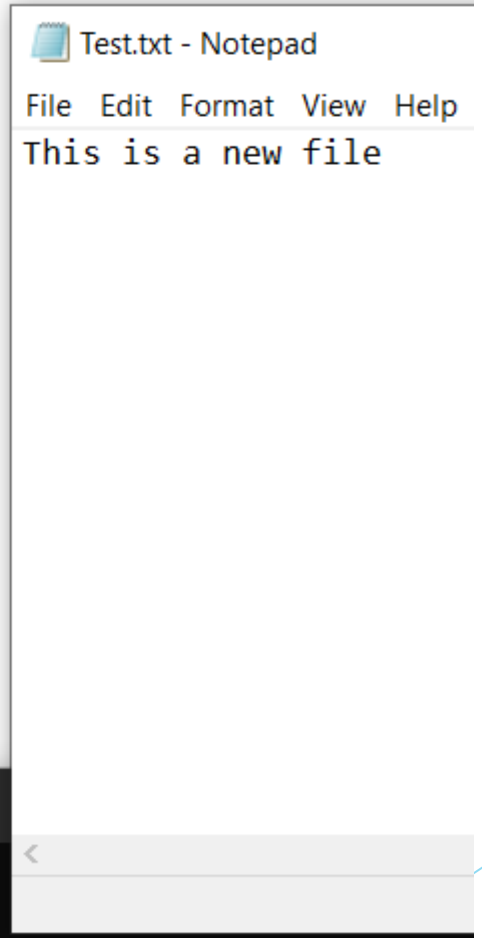
```
The result is: 10
Hello, Alle
Function with no value returned
Function with no value returned
The result is: 2
```

File handling

Function	Meaning	Attribute	Meaning
fopen	Create new file/open existing file	a	Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the last character in it. If the file doesn't exist, a new file is created.
fscanf()/ fgets()	Read from file	r	Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer which points to the first character in it.
fprintf/ fputs()	Write to file	w	Searches file. If the file exists, its contents are overwritten. If the file doesn't exist, a new file is created.
fseek() / rewind()	Move to a specific location in a file		
fclose()	Close a file		

File handling - examples

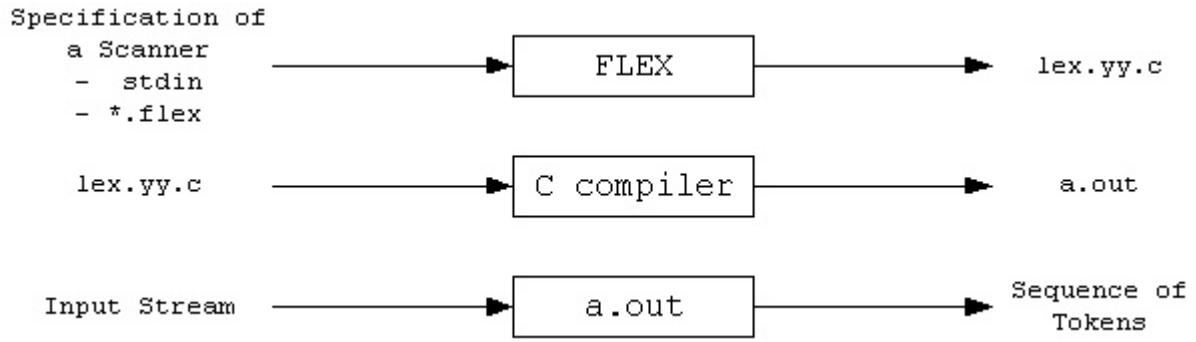
```c
# include <stdio.h>
# include <string.h>
int main( ){FILE *filePointer;
char dataToBeWritten[50]="This is a new file";
filePointer = fopen("Test.txt", "w");
if ( filePointer == NULL )
{
    printf( "Test.txt file failed to open." ) ;
}
else
{
    printf("The file is now opened.\n") ;
if ( strlen ( dataToBeWritten ) > 0 )
{
fputs(dataToBeWritten, filePointer) ;
fputs("\n", filePointer) ;
}
fclose(filePointer) ;
printf("Data successfully written in file Test.txt\n");
printf("The file is now closed.") ;
}
return 0;
}
```




```
D:\Facultate\Predat\FormalLanguagesAndCompilers\2022\C\Lab9.exe
The file is now opened.
Data successfully written in file Test.txt
The file is now closed.
Process returned 0 (0x0)   execution time : 0.068 s
Press any key to continue.
```

FLEX

- **FLEX** is a **f**ast **lex**ical analyzer generator. It performs the pattern matching on text. FLEX is a tool for generating scanners. Instead of writing a scanner from scratch, you only need to identify the vocabulary of a certain language, write a specification of patterns using regular expressions (e.g., DIGIT [0-9]), and FLEX will construct a scanner for you.



- Successor of Lex

FLEX

- Pattern matching primitives

Metacharacter	Matches
`.`	any character except newline
`\n`	newline
`*`	zero or more copies of the preceding expression
`+`	one or more copies of the preceding expression
`?`	zero or one copy of the preceding expression
`^`	beginning of line
`$`	end of line
`a|b`	`a` or `b`
`(ab)+`	one or more copies of `ab` (grouping)
`"a+b"`	literal "`a+b`" (C escapes still work)
`[]`	character class

\t : tab

- Predefined variables

Name	Function
`int yylex(void)`	call to invoke lexer, returns token
`char *yytext`	pointer to matched string
`yyleng`	length of matched string
`yylval`	value associated with token
`int yywrap(void)`	wrapup, return 1 if done, 0 if not done
`FILE *yyout`	output file
`FILE *yyin`	input file
`INITIAL`	initial start condition
`BEGIN`	condition switch start condition
`ECHO`	write matched string

FLEX

More patterns

x match the character 'x'

. any character except newline

[xyz] a "character class"; in this case, the pattern matches either an 'x', a 'y', or a 'z'

[abj-oZ] a "character class" with a range in it; matches an 'a', a 'b', any letter from 'j' through 'o', or a 'Z'

[^ A-Z] a "negated character class", i.e., any character but those in the class. In this case, any character EXCEPT an uppercase letter.

[^ A-Z\n] any character EXCEPT an uppercase letter ora newline

r* zero or more r's, where r is any regular expression

r+ one or more r's

r? or one r's (that is, "an optional r")

r{2,5} anywhere from two to five r's

r{2,} two or more r's

r{4} exactly 4 r's

{name} the expansion of the "name" definition (see above)

[+xyz]\"foo" the literal string: `[xyz]"foo"`

\X if X is an 'a', 'b', 'f', 'n', 'r', 't', or 'v', then the ANSI-C interpretation of \x. Otherwise, a literal 'X' (used to escape operators such as '*')

\123 the character with octal value 123

\x2a the character with hexadecimal value 2a

(r) match an r; parentheses are used to override precedence (see below)

rs the regular expression r followed by the regular expression s; called "concatenation"

r|s either an r or an s

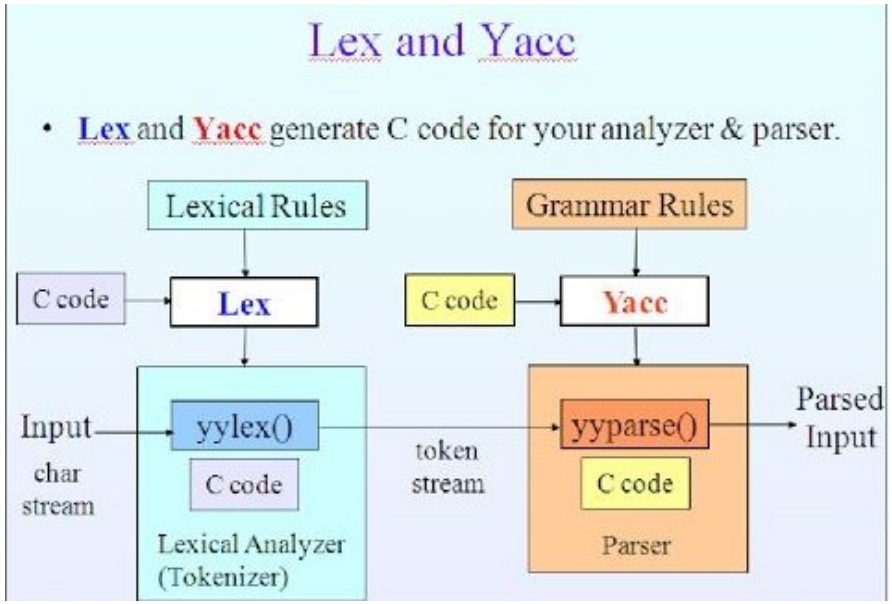
r/s an r but only if it is followed by an s. The s is not part of the matched text. This type of pattern is called as "trailing context".

^ r an r, but only at the beginning of a line

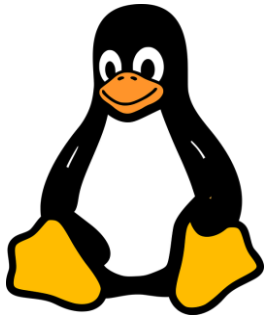
r$ an r, but only at the end of a line. Equivalent to "r/\n".

BISON

- **YACC** (Yet Another Compiler Compiler) is the standard parser generator for the Unix operating system. An open-source program, YACC generates code for the parser in the C programming language. YACC was originally designed for being complemented by Lex.
  - A parser generator is a program that takes as input a specification of a syntax and produces as output a procedure for recognizing that language.
- **Bison** is the GNU implementation/extension of **Yacc**.



Installing FLEX & BISON -Linux-



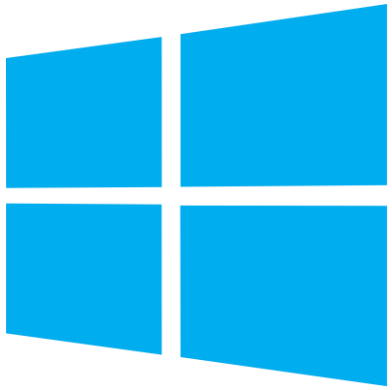
FLEX

- Step 1:Open Terminal
- Step 2: Write sudo apt-get install flex
- Step 3: You're good to go!

BISON

- Step 1:Open Terminal
- Step 2: Write sudo apt-get install bison
- Step 3: You're good to go!

Installing Flex -Windows-



- Step 1: Install Code::Blocks

  You can either search on Google for Code::Blocks and download from the official site or you can click here for direct download.

- Step 2: Install FLEX **!!The installation folder should not contain blank spaces!!**

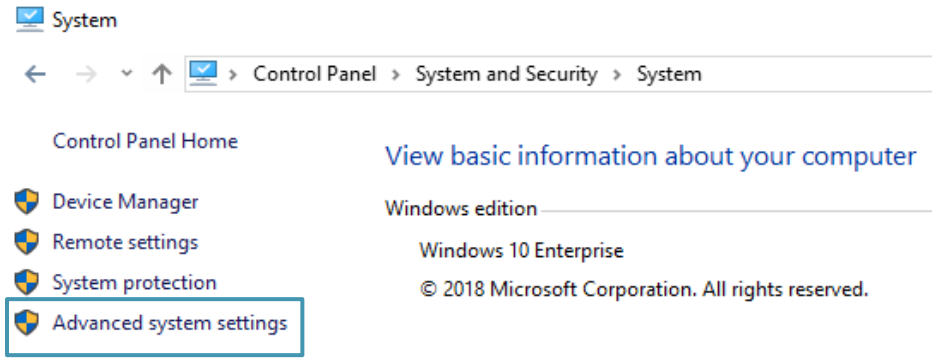
  You can either search on Google for FLEX GNUWin and download from the official site or you can click here for direct download.

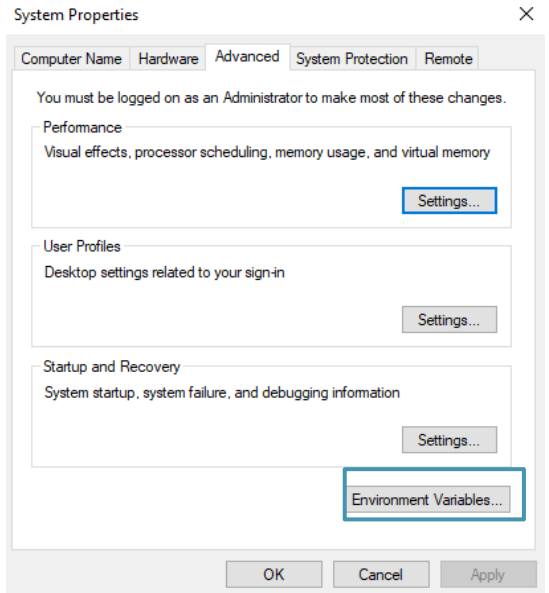
- Step 3: After installation, paths must be setup for Code::Blocks and FLEX

3.1: Path setup for Code::Blocks

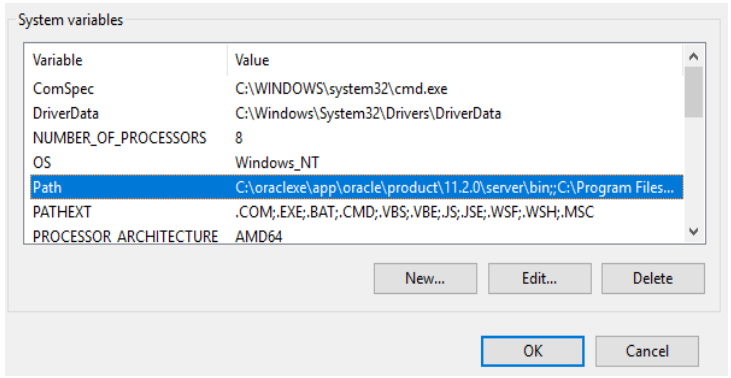
- Go to Code::Blocks installation folder → MinGW → Bin
- Copy the address of bin (it should look like this)

- Open Control Panel → System and Security → System → Advanced System Settings → Environment Variables





- **Double-Click** on Path inside System Variables → New
- Paste the copied path



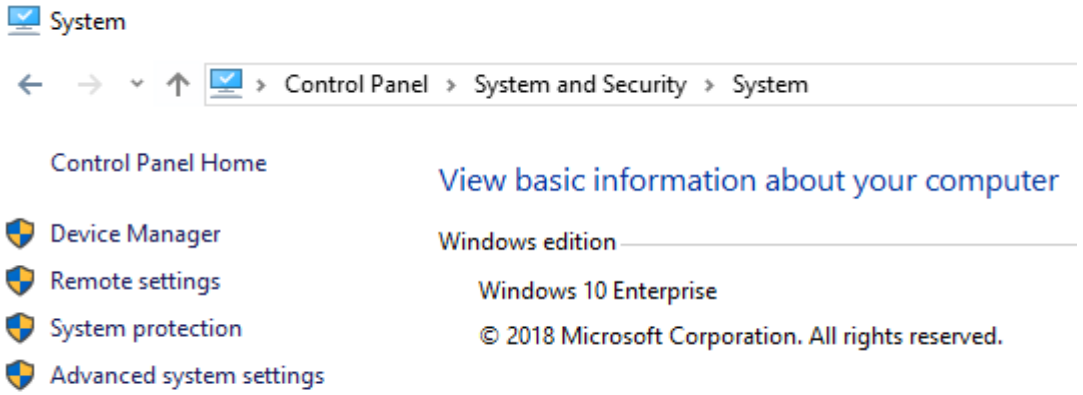
- Click Ok!

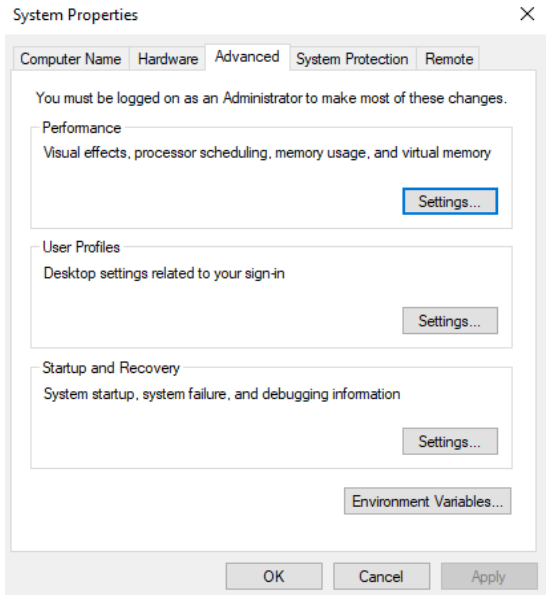
IMPORTANT! This step must be done before 3.2

3.2: Path setup for FLEX

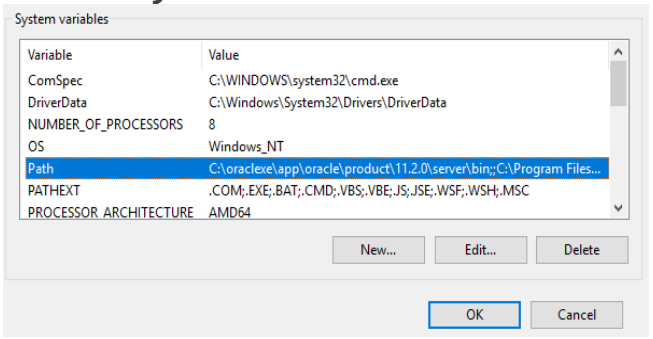
- Go to FLEX installation folder GNUWin32→ Bin
- Copy the address of bin (it should look like this)

- Open Control Panel → System and Security → System → Advanced System Settings → Environment Variables

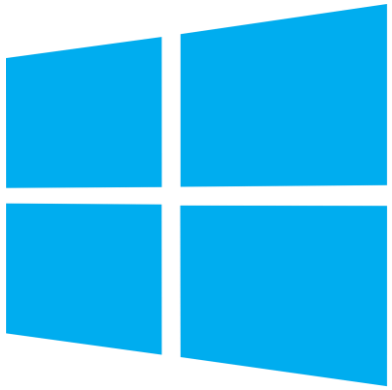




- **Double-Click** on Path inside System Variables → New
- Paste the copied path
- Click Ok!



Installing BISON -Windows-



- Step 1: For Bison to run on Windows, firstly we need to install Code::Blocks and FLEX.

  You can either search on Google for Bison and download from the official site or you can click here for direct download.

- Step 2: By default, Bison is installed in the same folder as FLEX (GNUWin). To simplify steps, it should be left as default.

- Step 3(Optional): If Bison was installed in another folder, the path must be setup like Step 3.2 from FLEX.

Flex file structure

- .l extension
- 3 sections separated by %%:
  - Definitions:
    - Declaration of Libraries, constants, ordinary C variables (e.g., %{ #include <math.h> %})
    - Flex definitions (e.g., Digit [0-9])
    - Text characters are written between “ ”
  - Rules:
    - Form: Pattern {action}
    - Actions are C/C++ code (e.g., [0-9]+ {return(Integer);}
  - User code:
    - Create routines (functions, identifiers)
    - main() is placed:

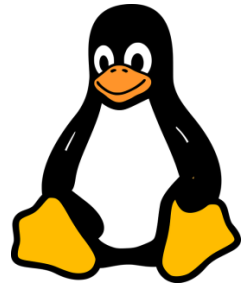
```
int main()
{
 yylex(); //calling the rules section
}
```

Flex Hello World

```
%{
//definition section

#undef yywrap
#define yywrap() 1

%}


%%
 //Rule section: print "Hello World" after pressing Enter

[\n] {
 printf("Hello World!\n");
}


%%
//The lexer produced by flex is a C routine called yylex(), so we call it.

int main()
{
 yylex(); //calling the rules section
}
```

!!! This code works only in Windows. In Linux an additional line of code should be written after %}: %option noyywrap

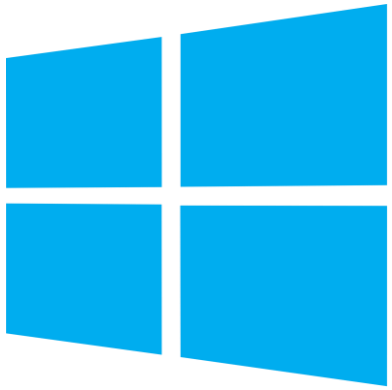
Running Flex
-Linux-



- In order to run FLEX on Linux you need to use Terminal.
- Step 1: In Terminal, you need to change the directory where the code is, using the command ‘cd’ (e.g. cd /Desktop/Ex)
- Step 2: Write flex <nameoffile> (e.g. flex example.l)
- Step 3: The above command will create a file name lex.yy.c. This file will be compiled by using the command cc lex.yy.c -lfl
- Step 4: To run the program, the command ./a.out must be introduced

```
alle@alle-VirtualBox:~/Desktop/Ex$ flex example.l
alle@alle-VirtualBox:~/Desktop/Ex$ cc lex.yy.c -lfl
alle@alle-VirtualBox:~/Desktop/Ex$ ./a.out

Hello World

Hello World
```

Running Flex -Windows-



- In order to run FLEX on Windows you need to use Command Prompt.
- Step 1: In CMD, you need to change the directory where the code is, using the command ‘cd’ (e.g. cd D:\Facultate\Predat\FormalLanguagesAndCompilers\Lab\Ex)
- Step 2: Write flex <nameoffile> (e.g. flex example.l)
- Step 3: The above command will create a file name lex.yy.c. This file will be compiled by using the command gcc lex.yy.c
- Step 4: To run the program, the command a.exe must be introduced.

```
D:\Facultate\Predat\FormalLanguagesAndCompilers\Lab\Ex>flex example.l

D:\Facultate\Predat\FormalLanguagesAndCompilers\Lab\Ex>gcc lex.yy.c

D:\Facultate\Predat\FormalLanguagesAndCompilers\Lab\Ex>a.exe

Hello World

Hello World
```

Exercises

1. Write a program that adds two numbers inputted by the user using pointers.
2. Write a program that takes as user input 2 numbers and computes their sum, difference, multiplication, division and mean.
3. Write a function that checks if a number inputted by the user is even or odd.
4. Write a function that computes the square of a number and call it twice.
5. Write a program that creates a new text file, then adds some text lines to it and, finally, reads the content of the file.

Homework

- Install flex and bison

1. Run example.l
2. Create a FLEX program which recognizes if a given letter is a vowel
3. Create a FLEX program which recognizes if a word is a pronoun
4. Extend exercise 3 by making it recognize also verbs, adjectives, nouns, prepositions (at least 10 of each) and digits (0-9).