Guillermo Herrera A.
A01400835

# Exercise based in Rust Language

1.1

a) A lexical error detected by the scanner

```
fn main() {
   let x = 5;
   println!("Hello, x = {}!", x);
```

# Error is in the bracket to close the function

b) A syntax error detected by the scanner

```
funn main() {
   let x = 5;
   println!("Hello, x = {}!", y);
}
```

# the variable y doesn’t exist yet in the symbol table and the declaration of the function is invalid

c) A semantic error detected by the scanner

```
fn main(){
   let x = 1;
   if x == 1{
      let a = 10;
      println!("A = {}", a);
   }
   println!("a = {}", a)
}
```

# the variable “a” created inside the if condition doesn´t have ownership outside of the block and its eliminated, so it doesn’t exists by the time the main wanted to print its value

d) A dynamic error not detected by code generated by the compiler

```
fn main(){
   let mut a:i64 = 2147483647;
   a = a + 1;
   println!("Value of a is {}", a);
}
```

# This addition causes an overflow that is not detected by the compiler without special flags

e) An error that the compiler neither catch not easily generate code to catch (this should be a violation of the language definition not just a program bug).

```
use std::fs::File;
fn main() {
   let f = File::open("archivito.txt")
   // Correct way to do it
   // let f = File::open("archivito.txt").expect("we dont have the file
yet");
   //
}
```

# in the case the file doesn’t exist, it will throw a panic signal and end the stack process.

1.5 )

The limit size of an Integer in Rust and C has the same range: -2,147,483,648 to 2,147,483,647, using 2 or 4 bits to store the value.
If there´s an overflow, the instance throws a panic signal to end the stack execution. A lot of languages do the same safe mechanism.

1.6 )

Regard to the “make” unix utility, this dependence management is accurate in order that verifies that any change of a file, must be represented in all the project due a consistency mechanism.

It would be an unnecessary work if the modification of the file doesn’t change the behavior of the result that it had before, for example, if you have a lib where it has a sorting function, and inside that lib a reactor was made, but the arguments that receives, the structs that it returns or the method doesn’t affects other registers, the full recompile would not be necessary,

1.7)

To tell that a program is incorrect has a direct relationship on if the code doesn’t do what is expected, those cases are placed on happy path execution of the code and cases where it is supposed to fail, the expected cases where the product does what is expected can be not only the result, also might be cases of memory usage, time expected, etc.