.

Honors Project
C++ Vs. Rust
By: Alon Gottlieb Chernov

December 11, 2020

1 Abstract

This paper will examine and present examples of the differences between two programming languages, C++ and Rust. Rust is a new programming language. However, many programmers are already programming language in C++ right now, but it is not preferred as a beginner level language. Despite competing in the same arena, C++ has a more robust foundation for community, frameworks, and general information about its principles. Rust is a newcomer in the programming world, and many developers are afraid to pick it up. People wonder who uses Rust and what are the possible implementations with this language. This paper shows the different types of aspects of Rust and how they are implemented in C++ with code snippets for each aspect. Keywords:Programming language, Compiler, C++, and Rust

2 Introduction

C++ is a 35-year-old programming language that is currently undergoing a revival, according to Tiobe Software, which says it is the fastest-growing language of any right now. C++'s heyday was in 2003 when it took up 17.53 percent share of the programming market. However, since then, C++ popularity has gone down substantially. Rust is a newcomer to the programming language market and is not even in the top 10 programming languages. However, it has gone from place 38 in the Tiobe Software in 2019 to 20 in 2020. In fact, it is being phased into Microsoft's coding software because according to Microsoft cloud developer Ryan Levick,"C++ at its core is not a safe language" He also states that C++/C were languages made for a different era and that they would not be able to protect against new vulnerabilities. Rust is best known for being a systems language, same with C++, which means that they are used for systems compared to user interaction, languages like Java and C. With a systems language, its best use case is for drivers and compilers. Rust is starting to become important in web development due to its web assembly feature, which will allow programmers to build secure and fast web applications. In Rust, the programmer does not have to manage memory, which means that it removes unnecessary functions, but in C++, the programmer has to manage this by himself, making programming harder.

3 Template Overview

The template that is used in this paper is the ACM writing format.

4 Authors and Affiliations

Alon Gottlieb Chernov
aloncher.got@gmail.com
Carlos Moreno
cmoreno@laspositascollege.edu

5 Variables and Data Types

Variables and data types are essential in computer programming. Data types and variables can be assigned in programming, or the user can input them. They are used to keep a program flowing, and they are the backbone of programming.

C++ Variables/Data types Example:

```cpp
int main(){
  cout << "Hello World!\n";

  string name ="Alon";
  int age =17;

  cout <<"My name is " << name << " and I am " << age << endl;

  // True Example
  bool drank_coffe=true;
  cout << "Did I drink coffee?" << drank_coffe << endl;
  // False Example
  bool ate_pizza =false;
  cout <<"Did I eat Pizza?" << ate_pizza << endl;

  char alon='a';
  cout << alon << endl;

  // String example
  string hello="Hello";
  cout << hello << endl;
  // String Length Example
  cout << "Length: " << hello.length() << endl;
  return 0;
}
```

Rust Example Variables/ Data types Example:

```
1  fn main(){
2      println!("Hello World!");
3
4      let name="Alon";
5      let age=17;
6      println!("My name is {} and I am {}", name, age);
7
8      // True example
9      let is_true=10>5;
10     println!("{:?}",(is_true));
11
12     // False example
13     let is_false=10<5;
14     println!("{:?}",(is_false));
15
16     let alon='a';
17     println!("{:?}", (alon));
18
19     // String Example
20     let hello="Hello";
21     println!("{}",(hello));
22
23     // String Example Length
24     println!("Length{}", hello.len());
25 }
```

Similarities: C++ and Rust follow the same general syntax. In both examples, the program's body is very similar, as is evident in lines 2-24 for the C++ example and lines 2-24 in Rust. The output is the same in both programming languages. These features are helpful because a programmer considering Rust can easily understand variable and data types in Rust, which will make learning this new mysterious language more comfortable for the programmer.

Differences: C++ takes almost twice as long to compile as compared to Rust, which means that Rust is better at compiling. Rust's variables can all be drawn back to the keyword, let as compared to C++, which requires the programmer to state if he wants a char and other types of variables. Rust does use unsigned integers and other integers, but the programmer sets the range. These features can be useful for the programmer if he wants to create a range and use less memory. Rust does not have cout or a greater than or less than symbols in their code compared to C++, saving time during programming. Rust requires println! For output, which is the equivalent to Cout in C++. If there is an error, Rust's compiler will show the programmer what is wrong compared to C++, which will only tell the programmer which line the error is on. Rust does not require the endl statement to go to the next sentence, which is extremely helpful because it saves time. Overall, Rust beats C++ in the Variables and Data Types section.

6 Input/Output

Input and output are key in programming. Essentially input usually comes in the form of what a user in the program will put in. What they put in affects the functionality of the code based on their choice. The output will display what the user put in, and it will display the result of the program.

C++ Input/Output Example:

```
 1 int main(){
 2   // Output/Input
 3   //User input Example
 4   string input="";
 5   cout <<"Hey say something: \n";
 6   getline(cin,input);
 7   cout <<"You said: " << input << endl;
 8 
 9   // Using integer
10   int x =0;
11   cout << "Enter a number: \n";
12   cin >> x;
13   cin. ignore();
14   cout << x << endl;
15 
16   // Output
17   cout <<"This is for my Honors Project:)\n";
18 
19   // Formatting output
20     cout << "*" << -17 << "*" << endl;
21     cout << "*" << setw(6) << -17 << "*" << endl;
22   return 0;
23 }
```

Rust Input/Output Example:

```
 1 fn main(){
 2   // Output/Input
 3   //User input Example
 4     let mut input=String::new();
 5     println!("Hey say something: ");
 6   io::stdin().read_line(&mut input)
 7      println!("You said:{}",input );
 8   // Using integer
 9     let mut x=String::new();
10     println!("Enter a number: ");
11   io::stdin().read_line(&mut input)
12   println!("You said:{}",x );
13 
14   // Output
15   cout <<"This is for my Honors Project:)\n";
16 
17   // Formatting output, this example will right align
18      println!("{number:>width$}", number=1, width=6);
19   return 0;
20 }
```

Similarities: The goal of both of these programs is to get user input and output it back out, as seen in lines 2-14 for the C++ example and lines 2-12 for

Rust. Both programs ask for user input, which is for the user to insert input to run based on the input given. Rust and C++ can format their output to either move an integer or string or make the program cleaner's general output.

Differences: Once again, C++ takes almost twice as long to compile as compared to Rust. It is significantly harder to get user input in the Rust programming language, which makes sense because Rust is not meant for user interface and deals more with web assembly, and the language focuses on web assembly. Rust requires a new variable call mut, which stands for mutation because the programmer is mutating the string variable to input their values. Rust also requires the programmer to declare a string with the let mut to receive an integer's user input. Overall, C++ handles input much better as compared to Rust. However, Rust beats C++ in output.

7 Arithmetic

Arithmetic is used to update integers and perform different arithmetic operators on integer values to update them. With arithmetic in programming, the programmer can carry out many simple tasks such as addition, subtraction, division, multiplication, and modulus.

C++ Arithmetic Example:

```
int main(){
    // arithmetic operators
  int x=10;
  int y=5;
  // Addition operator
  x+=9;
  cout << x << endl;
  // Subtraction Operator
  x -=8;
  cout << x << endl;
  // Multiplication operator
  x*=7;
  cout << x << endl;
  // Division Operator
  x /=2;
  cout << x << endl;
  // Modulus Operator
  x %= 3;
  cout << x << endl;
  // Combined assigment operator example
  x *=y-2;
  cout << x << endl;

  // Random Formula Example
  srand(time(0));
  int minVal=21;
  int maxVal=11;
  int z=minVal +rand() %(maxVal-minVal+1);
  cout << z << endl;
  return 0;
}
```

Rust Arithmetic Example:

```
1  fn main(){
2      // arithmetic operators
3    let mut x=10;
4    let mut y=5;
5    // Addition operator
6    x+=9;
7    println!("{}",x);
8    // Subtraction Operator
9    x -=8;
10   println!("{}",x);
11   // Multiplication operator
12   x*=7;
13   println!("{}",x);
14   // Division Operator
15   x /=2;
16   println!("{}",x);
17   // Modulus Operator
18   x %= 3;
19   println!("{}",x);
20   // Combined assigment operator example
21   x *=y-2;
22   println!("{}",x);
23
24   // Random Formula Example
25   let z=rand::thread_rng().gen_range(11,21);
26   println!("Random Number: {}",z);
27 }
```

Similarities: Rust and C++ use the same arithmetic operations to change the integer value and update the integer value, making it extremely easy to switch between the two, as is evident from lines 2-22 C++ and lines 2-22 in Rust example. They also create a range when creating a random number. The output in both of the programs is the same.

Differences: In Rust, the programmer has to create the mut variable to update the value. Rust does not require libraries like ctime, required in C++ for the random number to be created. However, both languages share many commonalities for the section of arithmetic operations, which will make it easier for a programmer to switch between the two. However, arithmetic operators are a tiny bit easier in Rust so that Rust can handle arithmetic operators better as compared to C++.

8 Conditions

Conditional statements are created by the programmer and determined by the user by seeing if the user's input is correct to the logic that the programmer created. Without conditional statements, many websites, videogames, etc., would not be possible.

C++ Conditions Example:

```
int main(){
    // if statement
  int score =87;
  if (score>90){
    cout << "That's a good score\n";
  }
    // If Else If
  int z=5;
  if (z<0){
    cout << z <<" is negative\n";
  }
  else if (z>0){
    cout << z <<" is positive\n";
  }
  // If else if and trailing else
  int n = 5;
  if (n<0){
    cout <<n <<" is negative\n";
  }
  else if (n>0){
    cout << n <<" is positive\n";
  }
  else {
    cout << n <<" is zero\n";
  }
  // Nested If Example
  int a = 25;
  if (a<100){
    if (a <50)  {
      cout <<" a is less than 50\n";
    }
  if (a >=50) {
    cout << "a is greater than 50\n";
  }
  }
  // Switch example
    int number =2;
  switch (number){
  case 1:
      cout <<"There is one of them\n";
      break;
  case 2:
            cout <<"There are two of them\n";
      break;
  case 3:
      cout <<"Does not match\n";
      break;
  }
}
  // Break example
  for (int x=7; x>=10;x++){
    if (x ==5 ){
      break;
    }
    cout << x << endl;
  }
  // Continue example
    for (int x=8; x>=10;x++){
      if (x ==5 ){
        continue;
      }
      cout << x << endl;
    }
  return 0;
}
```

Rust Conditions Example:

```
    // if statement
  let score =87;
  if score>90{
    println!(”That’s a good score”);
  }
    // If Else If
  let z=5;
  if z<0 {
    println!(”{} is negative”,z);
  }
  else if z>0{
    println!(”{} is positive”,z);
  }
  // If else if and trailing else
  let n = 5;
  if n<0{
    println!(”{} is negative”,n);
  }
  else if n>0{
    println!(”{} is positive”,z);
  }
  else {
    println!(”{} is zero”,z);
  }
  // Nested If Example
  let a = 25;
  if a<100{
    if a <50  {
    println!(”{} is less than 50”,z);
    }
  if a >=50 {
    println!(”{} is greater than 50”,z);
  }
  }
  // Switch example
  let number =2;
  match number{
    1=>println!(”There is one of them”),
    2=>println!(”There are two of them”),
    _=>println!(”It doesn’t match”)
  // Break example
  for 0..10{
    if x ==5 {
      break;
    }
    println!(”{}”,x);
  }
  // Continue example
    for 0..10{
      if x ==5{
        continue;
      }
    println!(”{}”,x)
    }
}
```

Similarities: The general syntax of the condition statements is the same as the break and continue statement, as seen in lines 2-54 in Rust and lines 2-71 in C++. These features make it easier for a programmer to switch between the languages if they are considering Rust. The output is the same with both languages.

Differences: Rust uses the switch statement but calls it match instead of switch, and frankly, it is much more comfortable using match instead because there are fewer steps, and it is more forward compared to C++. Rust can also set ranges with two periods instead of greater than or less than symbols, which saves many headaches for the programmer. However, using user input in Rust for conditional statements is difficult. Finally, the Rust compiler compiles faster as compared to the C++ compiler. Rust takes the bait in this section due to the simplification of the conditional statements. However, C++ manages input better as compared to Rust.

9 Loops

Loops are essential to programming because they will repeat a sequence of instructions until the logic's condition is met.

C++ Example:

```cpp
int main(){
  int n =1;
  // While loop
  while(n<101){
    if (n%15 ==0){
      cout << "alon\n";
    }else if (n%3 ==0){
      cout <<"gottlieb\n";
    }else if (n%5==0){
      cout <<"Project\n";
    }else {
      cout << n << endl;
    }
  // Increment counter
    n +=1;
  }
  // For loop example
  for (int n=1;n<=101;n++){
    if (n%15 ==0){
      cout << "alon\n";
    }else if (n%3 ==0){
      cout <<"gottlieb\n";
    }else if (n%5==0){
      cout <<"Project\n";
    }else {
      cout << n << endl;
    }
  }

  // Logical Operators
    int a = 20;
    int b =30;

     if (a > 10 && b > 10) {
      cout << "True\n";
    }
     int c = 0;
     int d = 30;

     if (c>10 || d>10){
        cout <<"true\n";
     }

    return 0;
}
```

Rust Loops Example:

```
 1 fn main(){
 2   // A counter variable
 3   let mut n =1;
 4   // While loop
 5   while n<101{
 6     if n%15 ==0{
 7     println!(”alon”);
 8     }else if n%3 ==0{
 9     println!(”gottlieb”);
10     }else if n%5==0{
11     println!(”Project”);
12     }else {
13     println!(”{}”,n);
14     }
15   // Increment counter
16     n +=1;
17   }
18   // For loop example
19   for n in 1..101{
20     if n%15 ==0{
21     println!(”alon”);
22     }else if n%3 ==0{
23     println!(”gottlieb”);
24     }else if n%5==0{
25     println!(”Project”);
26     }else {
27     println!(”{}”,n);
28     }
29   }
30 
31   // Logical Operators
32     let a = 20;
33     let b =30;
34 
35      if (a > 10) && (b > 10) {
36         println!(”true”);
37     }
38      let c = 0;
39      let d = 30;
40 
41      if (c>10) || (d>10){
42          println!(”true”);
43      }
44 }
```

Similarities: The Syntax of both programs is exceptionally similar, which helps a programmer proficient in C++ and wants to program in Rust, as is seen in lines 3-43 in Rust and lines 2-44 in C++. The output they produce is the same. The ranges set in the loops are the same. The logical operators are the same, which once again is extremely helpful.

Differences: There are many differences. First off, a programmer can declare if they want an infinite loop in Rust, which helps the programmer. There are no increment or decrement operators in Rust, which means it is harder to set the conditions for the loops in Rust and that there would be more programming steps. Overall, Rust simplifies the loop programming process, and it would be easier to code loops in Rust over C++. However, loops requiring user input are once again much more difficult in Rust than C++.

10 Functions

Functions are a block of code that will perform a repeated action. Functions are used in modular programming. Modular programming is taking a problem that the code needs to solve and breaking it down into smaller pieces.

C++ Functions Example:

```
 1 #include <iostream>
 2 using namespace std;
 3 int is_even(int);
 4 int main(){
 5   // Function example
 6   int n =28;
 7   is_even(n);
 8     return 0;
 9 }
10 
11 int is_even(int num){
12   if(num%2==0){
13     cout << ”Is even!\n”;
14   }
15 
16   return num;
17 }
```

Rust Functions Example:

```
 1 fn main(){
 2   // Function example
 3   print_numbers10(20);
 4 
 5 }
 6     // Function that return value and a combination of the two
 7   fn print_numbers10(num:u32){
 8   for n in 1..num{
 9         if is_even(n){
10             println!(”{},Is even!”,n);
11         }
12         else{
13         println!(”{},Is odd!”,n);
14         }
15         }
16 }
```

Similarities: The syntax of both of the programs are very similar. Programmers can declare function definitions at the bottom of the code for C++ and Rust, as seen in lines 1-11 in C++ and 16-18 in Rust. Aspects of Modular programming is implemented in both. The output is the same.

Differences: Rust is solely a pass by language, making it take up less space that is useful with memory. In Rust, the programmer does not need to declare prototypes Rust, making it more efficient for the programmer. In Rust, it is possible to combine two functions and then only call one function in main. Creating and declaring functions is much simpler in Rust over C++, which is especially important if a programmer or company considers Rust.

11 Arrays

An array is a data structure, which can store elements of the same data type. An array is useful in programming because it can create variables to put into a particular array.

C++ Arrays Example:

```cpp
void change_value(int[],int size);
int main(){
  // Array Example
  const int SIZE=5;
  int xs[SIZE]={
      1,12,17,19
  };
    cout <<"First element of the array: " << xs[0]<< endl;
    // Using functions with arrays
    const int sIze=5;
    int coolarr[sIze]={
        32,13,14,52,92
    };
    change_value(coolarr,sIze);
    cout <<"This is " << arr[1]<< endl;
    // For range loop
    const int Size=8;
    int values[Size]={
        2,12,13,18,123,12,18,92
    };
    for (int v:values){
      cout <<"V=" << v << endl;
    }
  // create a 2d array of string
  const int ROW_SIZE = 3;
  const int COL_SIZE = 2;

  // declare and initialize the 2D array
  int fourSquare[ROW_SIZE][COL_SIZE]=
  {
      {12, 23},
      {19, 32},
      {21, 123}
  };

  // print out the 2D array elements
  for (int row=0; row < ROW_SIZE; row++){
    for (int col=0; col<COL_SIZE; col++){
      cout << fourSquare[row][col]<<'\t';
    }
    cout << endl;
  }
    return 0;
}
int change_value(int arr[], int size){
  arr=10;
    int size = 0;
    for (i = 0; i < size; i++) {
      sum += arr[i];
   }
```

Rust Array Example:

```
fn main(){
  // Array Example
  let xs:[i32;4]= [1,12,17,19];
  println!("first element of the array: {}",xs[0]);
    // Using functions with arrays
    let mut arr:[i32;5]=[32,13,14,52,92];
    change_value(arr);
    println!("this is {}",arr[1]);
    // For range loop
    let values=[2,12,13,18,123,12,18,92];
    for v in &values{
    println!("V={}",v);
    }
    // 2D Arrays
   let mytable:[[i32;2];3]=[[8;2];3];}
   for(i,row) in mytable.iter().enumerate(){
        for (j,row) in mytable.iter().enumerate(){
        println!("[row={}][col={}]={}",i,j,col);
        }
   }
void change_value(arr:&mut[i32]){
  arr[]=10;
}
```

Similarities: The syntax between Rust and C++ is exceptionally similar, which is beneficial if a programmer is considering switching to Rust. The output of both of the programs is the same.

Differences: One immediate big difference is that it is much harder declaring 2D arrays in Rust, but 2D arrays are not used that often in programming, so this downside is not a huge one. Overall, declaring arrays is much simpler in Rust as compared to C++.

12 Vectors

Vectors are essentially the same as arrays, but they can perform more tasks as compared to arrays. Another benefit to vectors is that they can manage data, but they are slower and require more memory than arrays.

Rust Example

```
fn main(){
    let mut my_vector=vec![1,2,3,4];

    println!("{}",my_vector[2]);
    my_vector.push(49);
    my_vector.remove(1);

    // Loops with vectors
    for number in my_vector.iter(){
        println!("{}",number);
    }
 }
```

C++ Example

```
 1 int main()
 2 {
 3     vector<int> my_vector{1,2,3,4};
 4        cout << my_vector[2];
 5       my_vector.push_back(49);
 6
 7       // Using loop with vector
 8       for (int i=0; i<my_vector.size();i++){
 9          cout << my_vector[i]<< endl;
10       }
11     return 0;
12 }
```

Similarities: Vectors in both Rust and C++ share a very similar syntax, as is evident in lines 2-12 Rust and lines 2-11 C++, and they both perform the same function and produce the same output. However, C++ requires a different library to be called, which can be confusing to many programmers.

Differences: Also, using vectors in Rust is overall a much simpler process as compared to C++ because the declaration process of Vectors is more straightforward in Rust as compared to C++.

13 C-Strings

C-strings are part of the C language, which predated C++. There were no strings in the C language; instead, the programmer had to create an array and assign each box in the array to a char. Legacy code is frankly inefficient today, but some use cases such as creating passwords and checking how strong they are. In Rust, C-strings are not as implemented as much, which is shown in the example.

C++ Example:

```
int main(){
  int userChoice;
    switch(userChoice){
    case 1:{
      char uppercase;
      cout <<"Enter an uppercase letter: ";
      cin >> uppercase;
      if (isupper(uppercase)){
          cout << "You entered an upper case letter:)\n";
      }
      else{
        cout << "You did not enter an uppercase letter:(\n";
      }
      break;
    }

    case 2:{
    const int SIZE=50;
    char name[SIZE];
      cout <<"Enter a string and I will return length ";
      cin.ignore();
      cin.getline(name,SIZE);
      cout <<"You entered:" << name << endl;
      cout <<"The length is " <<strlen(name);

    break;
   }
    case 3:{
    string fir="";
    cout <<"I will return the first letter of this string\n";
    cin.ignore();
    getline(cin,fir);
    cout << fir.front();
       }
       break;
    }
  // terminate
  return 0;
}
```

Rust Example

```
fn main() {
        let c_s = hello();
        str::from_utf8_unchecked(slice::from_raw_parts(c_s as *
    const u8, strlen(c_s)+1))
    println!("s == {:?}", s);

}
```

Similarities: Both languages have C-strings, but the way their use case is different. There are not that many similarities.

Differences: In Rust, it is much harder to implement C-strings has compared to C++, which makes sense because of C. In Rust, there are no functions for C-strings, and they are rarely used. It is tough to find information for C-strings in Rust. Overall, implementing C-strings is much more comfortable in C++ as compared to Rust.

14 Structs and Enums

Structs and Enums are essential in programming because structs help replace several arrays, and instead, the programmer can define a structure above the main where many values can be stored. The primary purpose of enums is to make our code more readable, but they are not as used as structs.

C++ Example

```
#include <iostream>
#include <cstring>
using namespace std;
enum Color{
  RED, ORANGE, YELLOW,
};
struct videogame{
  string game;
};
struct user{
  videogame play;
  string Name;
  string computerusername;
  int age;
};
string convert (Color);
int main(){
  // create a Color enum variable
    Color myColor= YELLOW;

    // Print out my color choice
    cout <<"My favoite color: " << convert(myColor) << endl;

    // print out a color menu for the user
    for (int i = 0; i < 3; i++){
      cout << i <<" means ";
      cout << convert(static_cast<Color>(i)) << endl;
    }
    // let the user pick a color
    Color userColor;
    int userColorNum = 0;
    do{
    cout <<"Enter your color favoite color (0-2): ";
    cin >> userColorNum;
    }while (userColor < 0 || userColorNum > 2);
    userColor = static_cast <Color>(userColorNum);
    cout <<"Your favorite color is: " << convert(userColor) << endl
    ;

  // My defined user
  user myUser{
    "Fallout",
    "Alon",
    "Alongot",
    17
  };
  cout <<"Name:" << myUser.Name << endl;
  cout <<"Computer Username: " << myUser.computerusername << endl;
  cout <<"Age: " << myUser.age << endl;
  cout <<"Favorite Videogame: " << myUser.play.game << endl;

  // User defined user
  user localuser;
  cout <<"Please insert your information human!\n";
  cout <<"Enter your Name: \n";
  getline(cin,localuser.Name);
  cout <<"Enter your Computer username\n";
  getline(cin,localuser.computerusername);
  cout <<"Enter how old you are\n";
  cin >> localuser.age;
  cout <<"Enter your favorite Videogame\n";
  cin.ignore();
  getline(cin,localuser.play.game);
  cout <<"Name:" << localuser.Name << endl;
  cout <<"Computer Username: " << localuser.computerusername << endl;
  cout <<"Age: " << localuser.age << endl;
  cout <<"Favorite Videogame: " << localuser.play.game << endl;

  return 0;
}

string convert (Color c){
  switch (c) {
  case RED: return "Red";
  case ORANGE: return "Orange";
  case YELLOW: return "Yellow";
  default: return "N/A";
  }
}
```

Rust Example:

```
1  enum Color{
2      RED,
3      BLUE,
4      ORANGE,
5  }
6  
7  struct videogame{
8      game: &str,
9  }
10 
11 struct MyUser {
12     play: &videogame,
13     Name: &str,
14     email: &str,
15     age: &u64,
16 }
17 
18 fn main(){
19     // Enumerated Value
20     let favoritecolor:Color;
21     favoritecolor{
22         Color : RED=>println!(”Red”),
23         Color : BLUE=>println!(”Blue”),
24         Color : ORANGE=>println!(”Orange”),
25 
26     }
27     // My defined User
28     let user1 = User{
29     play: ”Fallout”,
30     Name: ”Alon”,
31     email: ”Aloncher.got”,
32     age: ”17”,
33     }
34     println!(”Name{:?}”, Name);
35     println!(”Email{:?}”, email);
36     println!(”Favorite game{:?}”, play);
37     println!(”Age{:?}”, age);
38 }
```

Similarities: The syntax between both of the programs is exceptionally similar, and both use the enum and struct keyword, as we can see in line 1 and line 7 in the Rust example and line 4 and 7 for the C++ example. The programmer has to define the structs and enums above main, as is evident from lines 1-16 for the Rust example and 1-16 for C++.

Differences: In Rust, the programmer does not need to put a period to pull up the specific struct data type, which is efficient and saves time, as is evident in lines 26-30 in the Rust example. It is easier to declare structs and enums in Rust as compared to C++, and this is a bonus for Rust. However, once again, User input is complicated in Rust as compared to C++. Overall, Structs and Enums are easier in Rust as compared to C++.

15 Files

Files are essential in the programming world because they are the fifth component of the Von Neumann Architecture, which means that data is volatile after the program is over. However, we can save the data from being lost and store it with the implementation of files.

C++ Example:

```cpp
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
using namespace std;

int main()
{
  // create an fstream variable
  fstream outputFile;

  // open a file using output mode
  outputFile.open("hello.txt", ios::out);
  outputFile <<"Hello, World\n";
  outputFile.close();

  // try opening the file
  do {
    string filename;
    cout << "Enter the filename to load: ";
    getline(cin, filename);
    inputFile.open(filename, ios::in);
  } while (inputFile.fail());

  // read the data from the file
  loadData(inputFile);

  // close the file
  inputFile.close();

  // terminate
  return 0;
}

void saveData(fstream& appendFile) {
  for (int i = 0; i < 5; i++) {
    int x;
    cout << "Enter a number: ";
    cin >> x;
    appendFile << x << endl;
  }
}

void loadData(fstream& inputFile) {
  int x;
  double y;
  string z;
  while (!inputFile.eof()) {
    inputFile >> x;
50     inputFile >> y;
51     inputFile.ignore();
52     getline(inputFile, z);
53 
54     // print out data to verify
55     cout << "x: " << x << endl;
56     cout << "y: " << y << endl;
57     cout << "z: " << z << endl;
58   }
59 }
60   // terminate
61   return 0;
62 }
```

Rust Example:

```
1  use std::fs::File;
2  use std::io::prelude::*;
3 
4  fn main() -> std::io::Result<()> {
5      let mut f = File::create("foo.txt")?;
6      f.write_all(b"Hello, world!")?;
7 
8      f.sync_data()?;
9      Ok(())
10 
11   io::Result<Vec<u64>> {
12   let mut ret = Vec::new();
13   let mut file = Eof::new(File::open("foo.bin")?);
14   while !file.eof()? {
15     ret.push(third_party::parse_thing(&mut file));
16   }
17   Ok(ret)
18 }
19 }
```

Similarities: Rust and C++ let the programmer have the ability to insert input into the file, which is seen in line 14 for C++, and line 6 for Rust. However, this is where the similarities end.

Differences: It is more complicated to write files in Rust as compared to C++, and a programmer needs to implement vectors in this process. These confusing steps could be due to the Rust being new.

16 Time Test

Throughout this report, there have only been speculations about how efficient Rust is as compared to C++. It was my responsibility to test out if Rust is efficient with a time test that would test out how long a program takes to run in C++ compared to Rust.

C++ Example

```
#include <iostream>
#include <stdio.h>
#include <chrono>

using namespace std;

int main () {
    double sum = 0;
    double add = 1;

    // Start measuring time
    auto begin = std::chrono::high_resolution_clock::now();

    int iterations = 1000*1000*1000;
    for (int i=0; i<iterations; i++) {
        sum += add;
        add /= 2.0;
    }

    int n, i;
    float num[100], summ=0.0, average;

    cout << "Enter the numbers of data: ";
    cin >> n;

    for(i = 0; i < n; ++i)
    {
        cout << i + 1 << ". Enter number: ";
        cin >> num[i];
        sum += num[i];
    }

    average = sum / n;
    cout << "Average = " << average;

    // Stop measuring time and calculate the elapsed time
    auto end = std::chrono::high_resolution_clock::now();
    auto elapsed = std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin);


    printf("Time measured: %.3f seconds.\n", elapsed.count() * 1e-9);

    return 0;
}
```

Rust Example

```
use time::PreciseTime;
use std::collections::HashMap;
fn main() {
let start = PreciseTime::now();

  let  mut numbers = vec![42, 1, 36, 34, 76, 378, 43, 1, 43, 54, 2, 3, 43];

  let avg: f32;
  let median: i32;
  let mode: i32;

  {
      let mut sum: i32 = 0;
      for x in &numbers {
          sum = sum + x;
      }

      avg = sum as f32 / numbers.len() as f32;
  }

  {
      numbers.sort();
      let mid = numbers.len() / 2;

      median = numbers[mid];
  }

      let mut times = HashMap::new();


      for x in &numbers {
          let cnt = times.entry(*x as usize).or_insert(0);
          *cnt += 1;
      }

      let mut best: (i32, i32) = (*times.iter().nth(0).expect("Fatal.").0 as i32, *times.iter().nth(0).expect("Fatal.").1 as i32);

      for x in times.iter() {
          if *x.1 > best.1 {
              best = (*x.0 as i32, *x.1);
          }
      }
      mode = best.0;

  println!("AVERAGE: {}", avg);
  println!("MEDIAN: {}", median);
  println!("MODE: {}", mode);

  let end = PreciseTime::now();
  println!("{} seconds", start.to(end));
}
```

Results: For the C++ program, the program takes input from the user on how much data the user can have and then enter those numbers in another function. The execution time was approximately 18.6 seconds. For the Rust program, the program takes a vector of numbers that the programmer inputted and does all sorts of calculations to calculate the average, median, and mode, as seen through lines 12-42; refer to Rust example. Surprisingly the program only took 16.5 seconds, which beats C++.

17 Conclusion

Rust is still a newcomer, and many more features will come in the future. However, Rust is looking very promising due to the high rise of popularity in recent years and the promising reviews that programmers programming Rust have. However, C++ is still widely used, and although it is not as modern, there are still many who use it. As Bjarne Stroustrup once said, “There are two kinds of languages: the ones people complain about and the ones nobody uses.” With that said, Rust will most likely only climb in popularity due to its efficiency, fast program speeds, and general likeability over C++.

References

[1] Liam Tung *Programming languages: C++ just jumped in popularity. Here's why* Retrieved November 15, 2020

[2] Lance Whitney *C++ is now the fastest-growing programming language* Retrieved October 13, 2020

[3] Paul Bilokon *C++ or Java? Which is best for trading systems?* Retrieved October 18, 2020

[4] Nick Kolakowski *C++ Growing Faster Than Other, Older Programming Languages: TIOBE* Retrieved November 11, 2020

[5] Liam Tung *C++ Programming language Rust: Mozilla job cuts have hit us badly but here's how we'll survive* Retrieved October 27, 2020

[6] Liam Tung *C++ Programming languages: Rust enters top 20 popularity rankings for the first time* Retrieved October 27, 2020

[7] Ambika Choudry *C++ Why Microsoft Is Dumping C C++ For This New Programming Language* Retrieved October 27, 2020

[8] Sarah Butcher *C++Is Rust a better programming language than C++ for finance jobs?* Retrieved October 29, 2020

[9] Jeffrey M. Perkel *Why scientists are turning to Rust*

Retrieved December 1, 2020

[10] *Rust Cookbook* Retrieved Decmber 8, 2020