Cp2SRust: A Transpiler for C/C++ to Safer Rust

Appendix Contents

Appendix A: Conversion Examples

This section demonstrates a few examples of various constructs that our auto transpiler Cp2SRust is capable of converting from C++ to Rust.

```cpp
// Driver code
int main()
{

    // Simple Declarations
    int n = 11, some_random_variable = 90;
    int b = 10 + 9 * 23 / 2;
    int c = n + b;
    int d = 99;
    int e = 34;

    unsigned int u = 1;
    long int l = 1234567890;
    char ch = 'h';
    float f = 1.23;

    int a = (b < c | g & e) == b ^ (j != t);
    printf("This is the something a: %d\n", a);
    // Ternary Operators
    int alpha = 10 > 5 ? 1 : 0;
}
```

```rust
fn main() {
    let mut n: i32 = 11;
    let mut some_random_variable: i32 = 90;
    let mut b: i32 = 10 + 9 * 23 / 2;
    let mut c: i32 = n + b;
    let mut d: i32 = 99;
    let mut e: i32 = 34;
    let mut u: u32 = 1;
    let mut l: i32 = 1234567890;
    let mut ch: char = 'h';
    let mut f: f32 = 1.23;
    let mut a: i32 = (b < c | g & e) == b ^ (j !=
     t);
    println!("This is the something a: {}\n", a);
    let mut alpha: i32 = if 10 > 5 { 1 } else { 0
     };
}
```

Figure A1. Conversion of Simple Declarations using Cp2SRust

```cpp
int main()
{
   // While Loop
   int n = 15, a = 10, b = 10;
   int c = 200;
   while (a + b)
   {
        b = b + 1;
   }

    // For Loop
    for (int i = 2; i <= n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
}
```

```rust
fn main() {
    let mut n: i32 = 15;
    let mut a: i32 = 10;
    let mut b: i32 = 10;
    let mut c: i32 = 200;
    while a + b {
        b = b + 1;
    }
    let mut i: i32 = 2;
    while i <= n {
        c = a + b;
        a = b;
        b = c;
        i += 1;
    }
}
```

Figure A2. Conversion of Loops using Cp2SRust

```c
// Driver code
int main() {
    // Conditionals and nested conditionals
    if (n < 10) {
        printf("This is easy!");
    } else {
        if (n == 100)

                printf("Do able Thing...");

        else {
            printf("Not possible");
            if (a + b || true) {
                a = 2;
                m = n;
                if (a) {
                    a = 3;
                }
            }

            for (int i = 0; i < (a && true); i++)
     {
                a = a + b;
            }
        }
    }
}
```

```rust
fn main() {
    if n < 10 {
        println!("This is easy!");
    } else {
        if n == 100 {
            println!("Do able Thing...");
        } else {
            println!("Not possible");
            if a + b || true {
                a = 2;
                m = n;
                if a {
                    a = 3;
                }
            }
            let mut i: i32 = 0;
            while i < (a && true) {
                a = a + b;
                i += 1;
            }
        }
    }
}
```

Figure A3. Conversion of Conditional Statements using Cp2SRust

```c
int main()
{
    // Switch Case
    int m_direction = 3;
    switch (m_Direction)
    {
        case 1:
        cout << "North";
        break;
        case 2:
        cout << "East";
        break;
        case 3:
        cout << "South";
        break;
        case 4:
        cout << "West";
        break;
        default:
        cout << "Invalid";
        return 0;
    }
    return 1;
}
```

```rust
fn main() {
    let mut m_direction: i32 = 3;
    match m_Direction {
        1 => {
            println!("North");
        }
        2 => {
            println!("East");
        }
        3 => {
            println!("South");
        }
        4 => {
            println!("West");
        }
        _ => {
            println!("Invalid");
            return 0;
        }
    }
  return 1;
}
```

Figure A4. Conversion of Switch Statement using Cp2SRust

```
#include <bits/stdc++.h>
using namespace std;

class Bool_Array {
    public:
    Bool_Array(unsigned int size)
    : _size(size),
    _iteration_number(1),
    _storage_array(new unsigned int[size]) {
        memset(_storage_array, 0, size * sizeof(_storage_array[0]));
        if (option[DEBUG])
        fprintf(stderr, "\nbool array size = %d, total bytes = %d\n", _size,
        static_cast<unsigned int>(_size * sizeof(_storage_array[0])));
    }

    bool set_bit(unsigned int index) {
        if (_storage_array[index] == _iteration_number)

        return true;
        else {
            _storage_array[index] = _iteration_number;
            return false;
        }
    }

    private:
    unsigned int const _size;
    unsigned int _iteration_number;
    unsigned int *const _storage_array;
};
```

Figure A5. Constructor Initializer in C++ present in bool-array.cc file of gperf module

```rust
// Using Namespace directives are not yet supported in this transpiler... Copying as it is
// using namespace std ;
#[derive(Default)]
pub struct Bool_Array {
    _size: u32,
    _iteration_number: u32,
    _storage_array: *mut u32,
}
impl Bool_Array {
    pub fn new(mut size: u32) -> Bool_Array // Handling constructor initializer
    {
        Bool_Array {
            _size: size,
            _iteration_number: 1,
            _storage_array: u32 size,
        };
        memset( _storage_array, 0, size * mem::size_of_val((_storage_array[0])),
        );
        if option[DEBUG] {
            fprintf(
            stderr,
            "\nbool array size = %d, total bytes = %d\n",
            self._size,
            (self._size * mem::size_of_val((_storage_array[0]))) as u32,
            );
        }
        /*
        This is a constructor method.
        Please appropriate members to the struct constructor as per your logic.
        Currently the constructor returns a struct with all the defaults for the data types
    struct.
        */
        Bool_Array { ..Default::default()  }
    }
    pub fn set_bit(&mut self, mut index: u32) -> bool {
        if _storage_array[index] == self._iteration_number {
            return true;
        } else {
            _storage_array[index] = self._iteration_number;
            return false;
        }
    }
}
```

Figure A6. Constructor Initializer in C++, present in bool-array.cc file of gperf module, converted to Rust using Cp2SRust