# Elementary Systems Programming

# Mid Term Test: September 27, 2022

**General instructions**

**Short Answer Questions**: Answer the questions with a short phrase or sentence. ***Most*** individual questions can be answered in, at most, two sentences. Some questions may only need a single word answer. Short fragments of code may be required too.

Minor English grammar errors or minor Rust syntax errors will be ignored. To achieve 100% marks, your answers will be need to be perfect, but only a few minor grammar or syntax errors may still lead to an A+ grade.

Write your answers on the test ‘paper’ in the space provided. If your answer needs more space or you want to completely change an answer, write the answer on additional sheets, making sure that your name, number and the question number appears clearly on any additional sheets.

A mark is indicated for each question or part of a question on the right: use it to guide the length of your answer.

There are marks on this test. A score of ………… will be sufficient to obtain an A grade.

Hand in your A4 sheet of notes with your completed test paper. It will ***not*** be assessed, but they generally provide comic relief for the markers and thus keep them sane and focussed in the long and dull marking process 😊.

|  |  |
| --- | --- |
| **Q1** Adding **mut** to variable protects it from  Explicitly allows it to be changed .. the default (without mut) is that you can assign a value to any varible ONCE only (when it is defined0\_ | 2 |
| **Q2(a)** What is the meaning of ‘mixed mode arithmetic’?  An expression which contains a mixture of data types, e.g. int and float  **Q2(b)** Is it allowed in Rust?  No | 2  1  2 |
| **Q3(a)** How do convert one type to another?  Use the ‘as’ operator: e,g, x as i32  **Q3(b)** You write in your program:  **let n = 100; let x = 23.1; let pc = x/n;**  However the compiler rejects this. Write a correct statement.  Mixed mode expressions are not permitted  let pc = x/(n as f64); | 2  2 |
| **Q4** Show how to declare a constant float, e.g. the value for **p**  const PI: f64 = 3.14….; | 2 |
| **Q5(a)** What is a ‘magic number’ in a program?  A literal constant (number or string) embedded in the middle of an expression  let pc = x/(155 as f64);  ‘155’ here is a ‘magic number’ and should be replaced by  const div\_factor: f64 = 155;  (or any name that tells a reader what the purpose of the ‘155’ is?  **Q5(b)** Are magic numbers allowed in a good program?  No  **Q5(c)** Explain why?   * hard to maintain (need to find the magic number), * danger that all instances are not found * less readable than a named constant | 2  1  2 |
| **Q6(a) let name = “Some name”;**  What type will be assigned to **name**?  &str .. a slice  **Q6(b)** Complete this sentence A Rust String must be a **valid set of Unicode characters** | 1  2 |
| **Q7(a)** How many bytes are needed for a Unicode character?  1 to 4  **Q7(b)** There are two iterators to scan over the elements of a string:  **s: String = ”abcdef”.to\_string();**  **for c in s.xxx() {**  **}**  **xxx()** can be (i) **chars**\_\_\_ or (ii) \_\_\_**bytes**\_\_\_\_\_\_\_\_\_\_\_\_  **Q7(c)** Why are there two? What is the diffence between them?  chars iterator returns each valid unicode characters  (ii)  bytes iterator returns individual bytes in the string | 2    4  4 |
| **Q8(a)** The compiler rejects this array declaration:  **let x:[f64;8];**  Why?  Rust arrays must be initialized.  **Q8(b)** Correct it so that it will be accepted.  let x:[f64;8] = [0.0;8]  or  let x:[f64;8] = [0.0,1.0,2.0,….. <any 8 numbers>] | 2  1   2 |
| **Q9** For a computer language, the *syntax* and the *semantics* are defined. Explain them.  Syntax The formal rules that a program must follow to be a ‘legal’ program accepted by the compiler.  Semantics The actions that the program will take when a legal program is translated for execution on a cmputer | 2 |
| **Q10** How many iterations will the following loops execute:   1. **for j in 0..10 { … } ……. 10** 2. **let x:[N;i32] = [0;N];…..  for xj in x { … } N** 3. **let s = “abcdefg”**   **for c in s.chars() { … } 7**   1. **for c in s.bytes() { … } ……. 7**   **Unicodes are variable length, ASCII codes need 1 byte**   1. **let s23 = “สองสาม”;**   **for c in s23.chars() { … } 6** | 5 |
| **Q11(a)** Your program has acquired a long sequences of readings of pressure and strored them in an array, **press:[f64;N]**. Write the code to find the maximum and minimum for these pressures. The code is basically trivial, most marks will be allocated for well-engineered code.  **const N:usize = 4;**  **fn min\_max(press:&[f32] ) -> (f32,f32) {**  **let mut p\_max = f32::MIN;**  **let mut p\_min = f32::MAX;**  **for j in 0..N {**  **let pj = press[j];**  **if pj > p\_max { p\_max = pj; }**  **if pj < p\_min { p\_min = pj; }**  **}**  **(p\_min, p\_max)**  **}** | 18 |
| **Q11(b)** For the next step, you need to classify the pressures as low, weak, ok, strong and high. Write the type that you would use for the **press\_class** variable.  enum press\_class {low=0, weak = 1, …, high = 4}  **Q 11(c)** Now write a function that generates a text label to appear in warning box on the operator’s console. A function **write\_console( s:String)** is available.  **const p\_levels:[f64;5] = [1.0,10.0, …]; // values set by application**  **const lev\_labels:[press\_class;5] = [low,weak,ok, …];**  **fn class\_press( p: f64 ) -> press\_class {**  **let p\_class = press\_class::low;**  **for j in 0..press.len() {**  **if press[j] > p\_levels[j] { return p\_levels[j-1]; }**  **}**  **press\_class::high**  **}**  But instead of providing the simple solution that is available in C++, etc,  Rust wants you to add another crate with a new trait in in \*and\* learn how to use that trait!! | 2  2  4  10 |
| **Q12** Write out the structures needed to model a **province**, which has a **name** and a collection of **districts**. Each district has a **name** for its main town, the **location** of that town and the district **population**. These structures form the basis of a GIS (Geographical Information System), which have many uses in planning and finances, e.g. counting people to calculate taxes 😉.  **Q 12(a)** the province  struct Province {  name: String;  dist\_list : Vec<district>m  }  **Q12(b)** a district  struct District {  name: String,  locn: GPSloc,  pop: u63,  }  Struct GPSLoc { east: f64, north : f64, }  **Q12(c)** *<any other structures that might be useful>*  struct GPSLoc { east: f64, north : f64, } | 3 |
| **Q13(a)** A Rust vector is a \_Last-In-First-Out (LIFO)\_ queue.  **Q13(b)** List the fundamental ways that elements of a vector can be accessed. *<List the major ones only – do not attempt to list all the functions operating on one!>*  For a vector, v: v[j], v.pop(), v.push()  **Q13(c)** A vector can be created with a command to set the capacity. Why is the initial capacity set?  Note: Vectors will add extra space when needed, but this may require copying the whole of the existing vector, so reallocations should be avoided.  Useful if you know an expected maximum number of entries, this will avoid expensive reallocations when the current capacity is exceeded.  **Q13(d)** What happens when the initial capacity is exceeded?  Extra space will automatically be allocated.  Note: The current Rust run-time library doubles the avoilabke space every time. However, this rule may not be followed in every library, e.g. in the next 6 weeks, the only requirement is that more space will be allocated.  Q13(e) Why should you try to set a likely initial capacity?  If the initial capacity is sufficient to avoid any reallocations, then your program will run faster overall.  Q13(f) What is the consequence of a poor initial choice?  Many reallocations may be caused and your program will run slower.  It may also waste memory, if the last reallocation (which may be the current alloaction) is not used. | 2  4  3  3  3  3 |
| **Q14(a)** Rust provides several floating point types: what are they?  f32, f64, f128  Note: f8, f16 are too small .. 16 bits does not have enough bits for mantissa + exponent  **Q14(b)** Add some notes to explain their differences and some comments to guide which one you should choose for your application. Remember, you are not expected to remember the actual precisions and ranges – you can always look them up when needed – but add some rough idea in your notes.  **As you use more bits, the range and the precision (least significant difference between two different values) that can be handled) will increase.**  On a modern high-end processor, f32 and f64 will probably compute at the same (or very similar) rate. f128 will be slower on any current computer, so should be avoided unless you need the range or the precision. For microprocessors, there may be a significant time penalty for f64 vs f32.  **Q14(c)** There are some predefined constants for the maxima and minima of each type which you can use in your program. What are they?  f64::MAX, f64::MIN, f32::MAX, f32:MIN | 6 |
| **Q15** There are also several types for integers.  **Q15(a)** What are they?  u8, i8, u16, i16, …, u128, i128  **Q15(b)** Why should you choose **u32** instead of **i32**?  If your number must be positive, you should use u32 as the compiler will check if your program computes a negative value for it.  The is NO difference in space or performance – both use 32 bits. | 3  1  2 |
| **Q16(a)** How are the codes used for representing characters in foreign languages used defined?  unicodes  **Q16(b)** Who is responsible for setting these codes?  Unicode consortium – an industry supported, quasi- standards organization  **Q16(c)** The Chinese character, 中(= middle, as in ‘middle kingdom’) has code 4E2D. Declare a Rust **char** for this character.  let zhong = ‘\u{4E2D}’; | 2  2  2 |
| **Q17** You can define collections of objects with arrays or vectors.  **Q17(a)** Advantages of arrays  Efficient, easy to accss (by index), fast, minimum space used  **Q17(b)** Disadvantages of arrays  Fixed, cannot be extended  Rust run-time system will check for overflow, but speed penalty is small  **Q17(c)** Advantages of vectors  Flexible – space is automatically allocated as needed, can be used as stacks or queues in some applications  **Q17(d)** Disadvantages of vectors  Slight penatly in run-time (slower) and extra space | 3  2  3  2 |
| **Q18(a)** What is a ‘**reserved word**’ in Rust? (What syntax rules apply to it?)  A word reseverd for use by the language itself; it cannot be used as a variable of function name  **Q18(b)** List at least 5 Rust reserved words?  fn, let, for, loop, if, else, impl, trait  Note: Rust distinguishes between reserved words (like these) and  pre-allocated types, like f32, f64 .. these are not true reserved words, as they can probably be changed if you’re a compiler specialist 😊 | 3  2 |
| **Q19(a)** Why was it strongly recommended that you add parentheses ‘( ..)’ to mathematical or logical expressions in your program?  To avoid memory overload for remembering things that you simple don’t need to remember (the precedence rules). Adding parentheses will ensure that YOU get it right AND readers of your code can follow it.  Rust’s standard advice for you to remove the parentheses is an appalling practice: the compiler should default to simply allowing them (or even encouraging them!!) | 3 |
| **Q20(a)** How many 8-bit bytes are used by a Rust **char**?  Unicodes vary from 1 to 4 bytes, a Rust char usually uses 4 bytes.  Input and output streams may be variable length ‘unicode points’.  **Q20(b)** How do I define an array of 8-bit bytes?  A: [u8:N] = [‘a’,’b’, …]; | 1  2 |
| **Q21** A Rust function to calculate an absolute value is desfined as  **abs(self)**  **Q21(a)** What does the **self** imply?  It is a reference to the object to which the function (abs) is applied OR the argument to the function  **Q21(a’)** Alternatively, you can answer this question by showing the two ways that the **abs** function can be used in an expression.  let ux = x.abs();  let uv = f64::abs(x); | 3 |
| **Q22(a)** What is the meaning of ‘named association’?  When attributes of a struct or arguments to a function are labelled with the name of the attributes or argument.  **Q22(b)** What is its advantage? Give an example of its use.  The attributes or arguments may appear in any order,  easing programmer’s trivial memory overload  Also helps documenting code.  let p: GPSLoc{ east: 104.0, north: 15.0 }; | 3  2 |
| **Q23(a)** What does a **break** statement do?  Exits from the current block (usually a loop)  **Q23(b)** Where can you use it?  Any loop – loop, while, for  **Q23(c)** What does a **continue** statement do?  Skips all statements from the continue statement to end of the current loop, and continutes with next iteration of the loop | 2  2  2 |