Name:	Harshit Jain
User ID:	hmj5262

Problem 1

number: 01010000010100110011001000110011

The given binary value is equivalent to the decimal number 1347629619 in a 32-bit unsigned integer.

The given binary value is equivalent to the ASCII codes for the ”PS23”.

The given binary value is equivalent to the floating point number 14173129728 in IEEE 754 single precision format.

The given binary value is equivalent to the array int8 t[4] = $80, 83, 50, 51$.

Problem 2

1. Each union occupies the maximum size of its member types, which are:

   U1: 8 bytes (double)

   U2: 12 bytes (3 ints $*$ 4 bytes)

   U3: 8 bytes (char pointer)

   U4: 5 bytes (char array of size 5)

   The size of struct S is 25 bytes.

2. The starting address of s is 2020, so we can calculate the starting addresses of s.info.other and s.student.name by adding the sizes of the previous members:

   s.info.other: $2020 + 4$ (index) $= 2024$

   s.student.name: $2024 + 8$ (info) $= 2032$

3. The total size will be 28 bytes.

Problem 3

1. Static typing and dynamic typing are two different approaches to type checking in programming languages.

   In **static typing**, the type of each variable and expression is declared explicitly by the programmer. This information is then used by the compiler to check for type errors at compile time. Static typing can help to catch errors early and make code more reliable. Some examples of statically typed languages are Java, C++, and C#.

   In **dynamic typing**, the type of each variable and expression is inferred from its value at runtime. This means that the programmer does not need to declare the type of each variable explicitly. Dynamic typing can make code more concise and easier to write, but it can also make it more difficult to debug and maintain. Some examples of dynamically typed languages are Python, JavaScript, and Ruby.

2. The choice between static and dynamic typing depends on the requirements of the project and the preferences of the developer.

   The advantages of using static typing include:

   Early detection of errors: The compiler can detect type errors at compile-time, which can save time and reduce the likelihood of errors occurring during runtime.

   Improved performance: Static typing can allow the compiler to generate more efficient code.

   Better code documentation: The type declarations in the code can make the code more readable and self-documenting.

   The disadvantages of using static typing include:

   More verbose code: The type declarations can make the code more verbose and harder to read.

   Less flexibility: The type system can be restrictive, and it can be difficult to express certain types of data structures.

   The advantages of using dynamic typing include:

   More flexibility: The lack of type declarations can allow for more flexibility and expressiveness in the code.

   Less verbose code: Without type declarations, the code can be more concise and easier to read.

   The disadvantages of using dynamic typing include:

   Runtime errors: Since type errors are not detected until runtime, it can be more difficult to find and fix errors.

   Reduced performance: Dynamic typing can lead to less efficient code, since the interpreter must dynamically determine the type of each variable at runtime.

3. A **strongly typed language** is a language in which the type of each variable and expression is strictly enforced by the compiler. This means that the compiler will not allow you to assign a value of one type to a variable of another type. Some examples of strongly typed languages are Java, C++, and C#.

   A **weakly typed language** is a language in which the type of each variable and expression is not strictly enforced by the compiler. This means that the compiler may allow you to assign a value of one type to a variable of another type, even if this is not semantically correct. Some examples of weakly typed languages are Python, JavaScript, and Ruby.

Problem 4

1. The type of variable x is **integer**, y is **float**, and z is **string**.
2. a is a **float**. The value of a is 1.1. This is because the $+$ operator performs integer arithmetic if both operands are integers, but float arithmetic if one or both operands are floats. In this case, x is an integer and y is a float, so float arithmetic is performed.

   b is an **integer**. The value of b is 1. This is because the $/$ operator performs integer division if both operands are integers. In this case, x and 1 are both integers, so integer division is performed.
3. c will raise a **TypeError exception**. This is because the $+$ operator cannot perform arithmetic on an integer and a string.
4. The output for the statement $1+1$ is 2. This is because the $+$ operator performs integer arithmetic when both operands are integers.

   The output for the statement ”1” $+$ ”1” is ”11”. This is because the $+$ operator performs string concatenation when both operands are strings.

Problem 5

Memory regions for dynamic and flexible arrays:

Static data area: The static data area is not suitable for dynamic and flexible arrays because the size of these arrays is not known at compile time.

Stack: The stack is also not suitable for dynamic and flexible arrays because the size of these arrays can change at runtime.

Heap: The **heap** is the most suitable memory region for dynamic and flexible arrays because the size of these arrays can be changed at runtime. In heap, memory can be allocated and deallocated at runtime. Heap memory is managed by the operating system and is typically used for data that needs to persist beyond the lifetime of a function or program, and whose size is not known at compile-time.

Bounds checking:

Bounds checking can be supported by using a pointer to the start of the array and a pointer to the end of the array. When accessing an element of the array, the pointer to the start of the array can be used to check if the index of the element is within the bounds of the array. The language can also check array access against this size. If the array access is out of bounds, the language can throw an exception or trigger a runtime error.