Class Activity - 2/15/23

1. When might you want the compiler to ignore type differences in an expression?

Suppose you have an expression with integers and type1 (let it be a subrange of integers). It would be useful for the difference between an integer and type1 to be ignored by the compiler when evaluating the expression. Another use would be when we print integers, ignoring the integer value and changing it to a char/string would be preferable as well.

2. State your own arguments for and against allowing mixed-mode arithmetic expressions.

For: A mixed mode arithmetic expression is needed for calculating expressions that could have decimal results. It is required because it allows two different types of numerical data (for example, float and integer) to be summed without losing the float's precision.

Against: Though necessary to have mixed-mode expressions, the chance of an error occurring is higher when expressions are more likely to have non-decimal results. A mixed mode might produce a decimal result even though the desired result is non-decimal.

3. Do you think the elimination of overloaded operators in your favorite language would be beneficial? Why or why not?

No, it would not be beneficial. We would have to define new operators for specific functions. For example, the + operator has several different uses. It can sum two numeric values (ex- print(1 + 2)) or concatenate two strings (ex- print("hello" + "world")). If we remove overloaded operators we would have to find new operators for each of the + operator's uses.

13. Let the function fun be defined as

```
int fun(int* k) {
*k += 4;
return 3 * (*k) - 1;
}
```

Suppose fun is used in a program as follows:

```
void main() {
  int i = 10, j = 10, sum1, sum2;
  sum1 = (i / 2) + fun(&i);
  sum2 = fun(&j) + (j / 2);
}
```

What are the values of sum1 and sum2

A. **operands in the expressions are evaluated left to right?**
   sum1 is 46
   sum2 is 48
B. **operands in the expressions are evaluated right to left?**
   sum1 is 48
   sum2 is 46