Session #2

Data, Variables, Operators, and Expressions

ICPC SCU

Session Structure

- What is the memory?
- Meaning of data?
- What is a variable?
- How do we manipulate variables?
- The usage of variables (from a mathematical aspect)
- What is an expression?
- Operators: Unary (-, !, --, ++, ~), Binary (+, -, >, <, ≤, >=, ==, !=, %, /, *, *=, -=, +=, /=, ^, &, |, |=, ^=, &=, >>, >>=, <<, <<=), Ternary (?:)
- Categories of these operators (Arithmetic, Logical, Bitwise, Assignment)
- Precedence of operators (Search)
- Logical operators and short circuits
- Mapping numbers to logical values

What is the memory?

- It is where **data** and instructions are stored and accessed by the computer.
- Can be divided into different types such as: primary, secondary, cache, ... etc.
- Each type of memory has its own characteristics, advantages, and disadvantages.
- Our written programs are stored in different types of memory depending on their stage and usage.
- Initially, our programs are stored in the secondary memory until we edit or delete it.
- Primary memory: fastest, most accessible type of memory. It is where the data and programs that are currently being used by the computer are stored.
- After compilation process, the result is an executable file that contains the instructions for the computer to run our program.
- The executable file is also stored in the secondary memory, but when we want to run our program, it is loaded into the primary memory, such as RAM, temporarily while it is being used by the computer.
- We may encounter some strange words referring to the memory like (Stack, Heap)

What is data?

- Data is a term that refers to any information that can be stored, processed, or communicated by a computer.
- Data can be anything that you can represent using numbers, letters, symbols, or sounds
- For example: your name, your favorite song, or a picture of your pet are all data.
- Data is the basic building block of any computer program, because it is what the program operates on and produces as output.
- How can you understand the meaning of data by a recipe?
- The program itself is also data!

Variables, what are they?

- Is a way of naming and storing a value that can change during the execution of a program.
- You can use it instead of using the value itself (it is easier to use “my age” instead of writing your actual age every time.
- What if you want to generalize a formula instead of writing actual data? (n * (n + 1) / 2)
- You can visualize the variable as a box containing anything you want
- You can use it in a way similar to the mathematical usage of it.
- In mathematics, a variable is a symbol that represents an unknown value.
- For example, in the equation x + y = z, x, y, and z are variables that can have different values.
- You can use variables to express general rules or patterns that apply to different values
- For example, you can use the variable n to represent any natural number, then write n + 1 to represent the next natural number after n

Operators

- Operators and operands: the operator is a symbol that does something meaningful (operation) given some data (operands), and the operand is the data given to the operator as input (3 + 4 = 7, 4 * 3)
- We can categorize the operators using their number of operands needed to do the operation. (Unary: one operand, Binary: two operands, Ternary: Three operands) (-x, ++, --) -> -3, x++, ++x, –y, y–
- Some operators that we already know: +, -, /, *
- Arithmetic (+), Assignment (=), Comparison (<), Logical (&&), Bitwise (>>)

Arithmetic operators

- These are operators that perform mathematical calculations on numeric operands, such as addition, subtraction, multiplication, division, and remainder
- 10 + 3 = 13
- 10 - 3 = 7
- 10 * 3 = 30
- 10 / 3 = 3.33333
- 10 % 3 = 1 (gives the remainder of the division)
- 10 / 3 = 3 * 3 + 1
- 6 % 3 = 0

Assignment Operators

- These are operators that assign a value to a variable.
- The most common assignment operator is =
- X = 5 (sets X's value equal to 5)
- X += 5 (Adds 5 to X, it is the same as writing X = X + 5)
- X -= 5 (The same but it does the subtraction)
- /=, *=, %= work the same as the above operators

Comparison Operators

- These are operators that compare two operands and return a boolean value (True, or False) based on the result of the comparison
- 3 < 10 (True)
- 3 > 10 (False)
- 3 >= 3 (True)
- 3 == 3 (True)
- 3 <= 3 (True)
- 3 != 3 (False)

Logical Operators

- These are the operators that combine or negate boolean operands and return a boolean value based on the logic of the operation
- X && Y (returns true if both are true, and false otherwise) (Search for Truth Tables)
- X || Y (returns true if either of them are true, false otherwise)
- Combine boolean values by giving the operands to the operator output of another operator.
- For example, (True) && (True) (True because both the first and second operands can be evaluated to True)
- !(x == y) takes the value of x == y which evaluates to either True or False, and then negates it X != Y

Mapping numbers to boolean values

- When dealing with logical operators, if the input to the operator is a number, C++ automatically treats it as a boolean value (True or False) as follows:
- If the number is 0 (Zero), the languages treats it as False
- Otherwise, it treats the number as True (Yes, even the negative values are being evaluated to True)
- True && False (False, because we can replace 3 and 0 with the boolean values True and False, respectively)
- 3 || 0 (Find the final value, and explain why)

Expressions

- They are pieces of code that can be evaluated to produce a value
- 3 > 10 was an expression. X = 5 was an expression as well
- Constant expressions are the expressions consist of only constant values (3)
- Variable expressions are the expressions that consist of variables (x + 5, y * z)
- Relational expressions are the expressions that use relational operators to compare two operands (x < 3)
- Arithmetic expressions are the expressions that use arithmetic operators to perform calculations on numeric operands. They evaluate to a numeric value (x + y, x / 7, y * z)
- Logical expressions are these use logical operators to combine or negate boolean operands, and of course, they evaluate to boolean values (x < 3 && y < 10)
- Assignment expressions also returns a value equals to the assigned value

Short-circuit evaluation of logical expressions

- Do we actually need to check the whole expression to find the value?
- 3 < 2 && 3 < 10 (logical && only returns true only if all operands given to it are true expressions, so we don’t need to check for the second expression if we already know the first expression evaluates to false)
- We should check all the expressions combined by the logical && while we are finding True expressions, if there’s at least one False expression, we can stop and return a big False
- The same applies to the logical ||, it evaluates to true if at least one of the operands evaluates to True.
- So we should check all the expressions combined by the logical || while we are finding False expressions, if there’s at least one True expression, we can stop and return a big True.
- C++ works in the same way when evaluating the logical expressions

Evaluate the following expressions, and mention all the operators, operands, sub-expressions they contain

- Example : (x < y) + (z -= 5)
- Expressions: ((x < y) + (z -= 5), x < y, z -= 5)
- Operators: (<: logical, +: arithmetic, -=: assignment)
- Operands: ({(x < y), (z -= 5)} are operands given to +, {x, y} are operands given to <, {z, 5} are operands given to -=)
- Sub-expressions: (x < y evaluates to True or False depending on the values, z -= 5 evaluates to z - 5 depending on the value of z, x is an expression itself that evaluates to True or False depending on the value of X, ... Try to complete it yourself)

Evaluate the following expressions, and mention all the operators, operands, sub-expressions they contain

- 2 + 3 * 4 (evaluate)
- Y = 5 (evaluate)
- X == Y
- !X
- !(-2) (evaluate)
- 2 + 3 * 4 / 2 (evaluate)
- X += Y * Z
- (X < Y) && (Y < Z)
- (3 > 1) && (3 > 2) && (1 >= 2) && (10 > 5) (The same as the following)
- (3 < 1) || (3 == 3) && (12 / 3) (evaluate and give the expression stops the evaluation)
- (3 < 1) || (3 == 3) || (12 / 3) (evaluate and give the expression ...)