HW1.

Q1. a) Goals:

1) To find (and be prepared for) interview opportunities that demand the following software skills, for both software and hardware teams of technical companies:
   A) OOP (using C++) and structured programming (using C)
   B) Algorithms
   C) Data Structures
   D) Scripting (using Python) and regex needed for automation and verification tasks
   E) Several software management techniques (including versioning testing, database management, data stream also multi-thread)
2) To do a project that helps you differentiate yourself from the rest of the large crowd of new graduates who would more or less take the same courses and therefore list very similar skills and projects in their resumes.
3) In addition to getting trained for the current and short term needs of industry, to develop the self-training skills in anticipation of future technologies. You will be trained to be efficient in doing research (studying, planning, devoloping new ideas). In addition to hard working Master's students, we highly recommend first year and second year Ph.D. students of various engineering disciplines to take this course.

b) Because verification, design automation, and development jobs requires optimal ways to perform better so that they need strong algorithm design and programming skills.

Q2、Q3、Q4: In seperate files

Q5. In Python, it includes automatic memory management and dynamic type system While in C++, it does not provide garbage collection so it is more likely to see memory leak in C++

In Python, when define x=5, system will assign a memory space for x to store 5. In C++, before defining x=5, there should be type declaration for x to be int type, so that system will assign an appropriate memory space for x. Then define x=5, 5 will be stored into that pre-allocated memory space.

Q6. For dynamically typed language, you do not need to type a variable beforehand It performs type checking at run-time while statically typed language checks at compile-time.

Dynamically typed language takes more time running. Therefore, when the scenario requires fast outcome, the statically typed language would be preferred.

Q7. A Strongly typed language is more likely to generate an error or refuse to compile if the arguement passed to a function does not closely match the expected type; while a weakly typed language may produce unpredictable results or may perform implicit type conversion.

I prefer strongly typed language because it is more rigorous so that it can reduce many uncertain errors which is better for debugging.

Q8. List: like an array but can stores heterogeneous types of objects;
After initial creation, individual elements can be reassigned;
Comma separated values between square brackets
e.g.: list = ['Hi', 5, 6.5]

Tuple: like list but individual elements cannot be changed after initial creation
Comma separated values between parentheses
e.g.: x = ('Hi', 5, 6.5)

Dictionary: Associates key, value pairs;
keys needs to be something hashtable, values can be anything;
can get keys or values back in the list;
can check membership
e.g.: dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

Q9. Compiled languages:

(1) requires code to be converted to the native machine language of the processor in the target system before it can be run.

(2) Analogy: Taking a speech and translating it to a different language ahead of time so that speaker can just read it

(3) Faster

(4) Often allows programmer closer access to the hardware

Interpreted languages:

(1) requires an interpreter program on the target system that will interpret the program source code command by command to the native system at run-time

(2) Analogy: Speaking through an interpreter where the speaker waits while the translator interprets

(3) Better portability to different systems

(4) Often abstracts HW functionality with built-in libraries (networking, file I/O)

Q10. Sample code is in seperate file.

Result:

```
10
11
--------------------------------
10
11
11
```

Q11.

Compiler		Physical System	
Name	Type	Address	Value
gpa	float	0x120	3.9
id	long int	0x124	11231423
family_name	char[]	0x128	'B'
			'r'
			'o'
			'w'
			'n'
			...
		0x163	00
ptr	char*	0x164	0x128

Q12. Code is in seperate file.

Result:

```
32
34
24
```