1.Solution:

(a)

	Variables	Functions
Function main	<a, 15> < b, 15> < h, 1> <i, 1>	<A, 8> <B, 2>
Function A	<x, 8> < y, 8> < i, 9> <j, 9> <h,1>	<B, 2>
Function B	<i, 1> <j, 3> <k, 3> <w, 2> <h, 1>	

(b) <i, 1> < j, 3> <k, 3> <w, 2> <h, 1>

(c) <i, 9> < j, 9> <h, 1> <x, 8> <y, 8> <B, 2>

(d) <i, 9> < j, 3> <k, 3> <w, 2> <h, 1>

(I am a little bit confused about this question. For example, in (a) question, function main could use var i which is declared at line 1. But in the main function, var i didn't appear. According the Piazza answer, professor said "Visibility is about whether a var can be used, not about whether it is used." So I consider var i is visible. Question (b) to (d) also use this rule. Please give me some comments whether I am right or wrong. Thank you!)

2.Solution:

(a)What is namespace in C++? How it works?

A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries. All identifiers at namespace scope are visible to one another without qualification. Identifiers outside the namespace can access the members by using the fully qualified name for each identifier, for example std::vector<std::string> vec;, or else by a using Declaration for a single identifier (using std::string), or a using Directive for all the identifiers in the namespace (using namespace std;). Code in header files should always use the fully qualified namespace name.

The following example shows a namespace declaration and three ways that code outside the namespace can accesses their members.

```
namespace ContosoData
{
    class ObjectManager
    {
    public:
        void DoSomething() {}
    };
    void Func(ObjectManager) {}
}
```

Use the fully qualified name:

```
ContosoData::ObjectManager mgr;
mgr.DoSomething();
ContosoData::Func(mgr);
```

Use a using declaration to bring one identifier into scope:

```
using WidgetsUnlimited::ObjectManager;
ObjectManager mgr;
mgr.DoSomething();
```

Use a using directive to bring everything in the namespace into scope:

```
using namespace WidgetsUnlimited;
ObjectManager mgr;
mgr.DoSomething();
Func(mgr);
```

The using directive allows all the names in a namespace to be used without the namespace-name as an explicit qualifier. Use a using directive in an implementation file (i.e. *.cpp) if you are using several different identifiers in a namespace; if you are just using one or two identifiers, then consider a using declaration to only bring those identifiers into scope and not all the identifiers in the namespace. If a local variable has the same name as a namespace variable, the namespace variable is hidden. It is an error to have a namespace variable with the same name as a global variable.

Typically, you declare a namespace in a header file. If your function implementations are in a separate file, then qualify the function names, as in this example.

```
//contosoData.h
#pragma once
namespace ContosoDataServer
{
    void Foo();
    int Bar();

}
```

Function implementations in contosodata.cpp should use the fully qualified name, even if you place a using directive at the top of the file:

```
#include "contosodata.h"
using namespace ContosoDataServer;

void ContosoDataServer::Foo()
{
   //no qualification because using directive above
   Bar();
}

int ContosoDataServer::Bar(){return 0;}
```

A namespace can be declared in multiple blocks in a single file, and in multiple files. The compiler joins the parts together during preprocessing and the resulting namespace contains all the members declared in all the parts. An example of this is the std namespace which is declared in each of the header files in the standard library.
Members of a named namespace can be defined outside the namespace in which they are declared by explicit qualification of the name being defined. However, the definition must appear after the point of declaration in a namespace that encloses the declaration's namespace. For example:

```
// defining_namespace_members.cpp
// C2039 expected
namespace V {
        void f();
    }

    void V::f() { }         // ok
    void V::g() { }         // C2039, g() is not yet a member of V

    namespace V {
        void g();
    }
}
```

This error can occur when namespace members are declared across multiple header files, and you have not included those headers in the correct order.

(b)Benefits:

The namespace is used to deal with the namespace clashes. In the development of large projects, there may be many people writing code at the same time. Different modules may use the same global variables, and name collisions may occur when the system is integrated. Using the namespace could solve this problem very well.

3.Solution:

This question talks about the variable lifetime and scope. Lifetime determines whether the variable is still in the allocated memory. However, the scope of the variable is whether it can access the variable by its name. So it is possible that a variable is live, but not visible (not in scope) and it is also means that a variable could be not alive but in scope. So for this question, the answer is absolutely yes.

Example 1:

```
1.{int x = 1;
2.    { int y = 2;
3.       {
4.              int x = 3;
5.       }
6.     }
7.}
```

In this example, the outer x lifetime is from row 1 to row 7, but its scope is row 1, row 2, row 6 and row 7. It means a variable can be allocated but not visible.

Example 2:

```
1.{Bird a;
2.    { int y = 2;
3.       {
4.              int x = 3;
```

```
5.          }
6.        }
7.}
```
In this example, the variable a scope is from row 1 to row 7 but I didn't use "Bird a = new Bird();" to allocate memory for it. So it can be visible but not allocated.