# Chapter 1 Basic ideas

## 1.1 View on RAM as a programmer

There is the memory in any PC, the fast random access memory called RAM.



Fig. 1-RAM device.

As a programmer you should think about the memory at the higher level than the physical device that you can see if you have dissembled system unit. So, all memory at the level of programming applications can be divided into three kinds.

First kind is a static memory. When you launch any application the operation system loads executable file of your application into the RAM. The operation system allocates memory for binary code and for variables and constants. This kind of memory lives until the application is closed. Operation system unloads all the application data from RAM after application was closed. So, “static” in this case means “to live while application works”. The size of static memory is not changed.

Second kind of memory is called “dynamic” or “heap”. When your application needs some additional memory it asks the operation system for that, and operation system allocates the memory if there is. When application no longer need early allocated memory it should return it back to operation system. So, this is why it is called “dynamic”.

Third kind of memory is called “stack”. It used by functions only. Each function has its own piece of stack memory.

Look at the fig.1-2 to view schematic image of memory used by application.

You always should understand in which piece of memory your data is located. For remember this kinds of memory you can use abbreviation “SSD” (Stack, Static, Dynamic). Ok, move on.

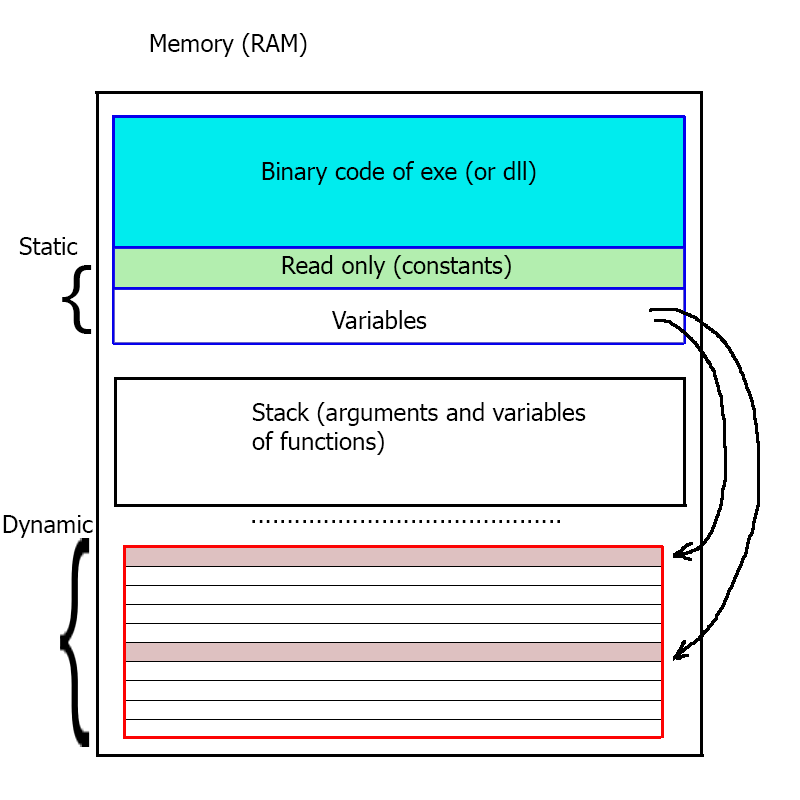


Fig. 2-Application memory.

## 

## 1.2 Types

Look at the string “IVI”. What do you see? Is this a sequence, containing ‘I’, ‘V’, ‘I’ characters? May be it is sequence of roman numbers: 1, 5, 1 or 4,1 or 1,6. You can’t say. It could be anything. So, you need a key, some description of how data organized in this sequence. This description is called “format”. Without knowing the format of data you can’t read image from a file. For example, you have 3 bytes for the color of a pixel. How are colors organized in those 3 bytes: RGB or BGR. Without knowing the format any data is just a useless bunch of bytes. In C++ format of the data is realized through the types. Each data (variables) must have a type, a format that describes what size that data occupies in the memory, how to interpret that data and what operations we can use with data. Syntax is following:

|  |
| --- |
| type\_with\_optional\_specifiers identifier; |

For to be more specific, C++ syntax give us keywords “int” to store integer numbers, and “float” to store floating-point numbers.

|  |
| --- |
| int intVar; float fltVar; |

The code in C++ consist of statements. There are two variable definition statements above. C++ syntax requires every statement to be terminated by semi-colon. Any variable must have a name (identifier). You can use a-z A-Z ANSII characters anywhere, 0-9 characters and underscore anywhere but at start for the variable identifier. So, language allow use underscore as first character in identifier, but really it is bad approach because many common used libraries have identifiers starting with underscore.

Compiler doesn’t understand you if you write something like that:

|  |
| --- |
| x; |

if you have not written

|  |
| --- |
| SomeType x; |

early in the source code.

Compiler doesn’t know what size variable x takes up in memory, how to interpret its content. What is “x”?

# Chapter 2

# Chapter 3 Memory allocation

## Pointer-like types

The STL containers can store their data in memory that is not accessible via regular pointers. The allocator concept does not require to implement allocator class using a void\* as void pointer type and regular pointer as pointer type. Yes, in common cases we use void\* and regular pointers. But there are areas where we can’t use just regular pointers. For example, it is an interprocess communication. Each process has its own address space. The process does not use direct access to the physical memory. There is separate address space for each process. And that address space is mapped to the physical memory by some mechanism of the operation system. Therefore, you can’t use regular pointers when you have memory chunk shared between two processes and STL containers located in it. You really need some pointer-like class with behavior similar to regular pointer. Look at the boost::interprocess::offset\_ptr type, if you need some real example. That’s why many STL classes provides its own types similar to the base C++ types.  
So, these are additional materials:

CppCon 2018: Bob Steagall “Fancy Pointers for Fun and Profit”