[Skip to main content](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/courseware/a71234edfd4f465e8d3e7e0082d04540/6f6cf3468dc649d791c51e0dbdd1b18b/1?activate_block_id=block-v1%3AIITBombayX%2BCS101.2x%2B2022_T1%2Btype%40vertical%2Bblock%406bbb49c83c614a4d85fa4ff3e8ab1ce8#main)

**[IITBombayX Home Page](http://iitbombayx.in/)**

**IITBombayX: CS101.2xObject-Oriented Programming**

[Courses](http://iitbombayx.in/courses)

[Help](https://iitbombayx.in/contact_us)

[[https://courses.iitbombayx.in/static/images/profiles/default_50.3455a6581573.png](https://courses.iitbombayx.in/dashboard) Dashboard for:Paramhans\_Singh](https://courses.iitbombayx.in/dashboard)

1. [Course, current location](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/course/)

1. [Discussion](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/discussion/forum/)

1. [Progress](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/progress)

1. [Grading Policy](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/487375f5363a49a492641b63d5e814ec/)

1. [Course Slides](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/pdfbook/0/)

1. [Practice Programming](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/b5c2b3240ef9416ebb0ea5b2087cb4ee/)

1. [FAQs](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/f2e9c32b35ae4ef083ed716b4981ff0d/)

[Course](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/course/)  [Topic 2: Review of Pointers and Structures](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/course/#block-v1:IITBombayX+CS101.2x+2022_T1+type@chapter+block@a71234edfd4f465e8d3e7e0082d04540)  [Session 8: Programming Using Structures](https://courses.iitbombayx.in/courses/course-v1:IITBombayX+CS101.2x+2022_T1/course/#block-v1:IITBombayX+CS101.2x+2022_T1+type@sequential+block@6f6cf3468dc649d791c51e0dbdd1b18b)  Walkthrough

 PreviousNext

1. video Programming Using Structures
2. other Slides: Programming Using Structures
3. other Walkthrough

**Walkthrough**

 Bookmark this page

**Storage of Program Entities in Memory**

**Introduction**

Now that all of you have written basic programs and have been coding for a while, many of you would have wondered how compilers work. How are programs(including all its variables and functions and structs) stored in the memory? In this walkthrough, we will explore answers to these questions. This extract will provide you with a very high-level overview of some of these functionalities and describe briefly what happens in the background when you run a program.

**PARTS OF program MEMORY**

The program memory can be divided into many sections that are stored differently in the RAM during its execution.

* Text or Code Segment
* Data Segment
* Stack
* Heap

**Text Segment**

Any code in C++ involving assignment, arithmetic operations etc.. is executed in what is known as instructions at the hardware level. All such instructions of a program comprise code segment of the program.

**Data Segment**

All global variables, static variables are stored in the data segment of the program.

**Stack**

The stack is the part of the program that enables function call abstraction. It is used for storing local variables that get generated as part of function calls.

**Heap**

Whenever we use the new operator to allocate memory to pointers, all that memory is allocated in the heap section.

Now what happens when a program executes is a bit out of the scope of this walkthrough. In a nutshell, when a program is running the instructions (refer code section above) are executed one by one and work is done accordingly.

**Structs storage**

After learning how the variables, functions are stored and implemented, now let's move on to understand how are structs stored in memory.

**Member Variables**

Struct members are not stored unless instantiated. If they are allocated using the new operator, the appropriate space is allocated in heap section. If the object is made global, then memory is assigned in the data section and if the object is local then the stack is used. Every object that is brought into existence by the use of a constructor is assigned the memory corresponding to all member variables.

**Member Functions**

Member functions do not contribute to the size of the struct. (Except in the case of virtual functions - will be covered later). This is because it will consume just too much memory if functions are stored in such a way.

Suppose a class V has a function void print(); then it is implemented in a way where the object's address is secretly passed as an argument to this function i.e the above function is equivalent void print(V\* object); For more details on using V\* object (this pointer) to implement function calls check [this](https://studio.iitbombayx.in/container/block-v1:IITBombayX+CS101.2x+2017T1+type@vertical+block@de4c5ce7848c4364ae2b28830e07c9d5) walkthrough.

This way a lot of memory is saved. We cannot do the same for member variables as these variables store the state of an individual object and belong to them.

The above discussion leads to the conclusion that size of structs is same as the size of member variables in them. But there is a slight twist to this statement.

**padding and word BOUNDARY**

Consider this program.

#include <iostream>  
struct Data{  
    int a;  
    char b;  
    int c;  
};  
int main(){  
    std::cout << "Data Struct size :" << sizeof(Data) << std::endl;  
}

**Expected Output**:

Data Struct size: 9    // 9 = 4 + 1 + 4

**Actual Output**:

Data Struct size: 12

Why is the size of struct not equal to the sum of the size of its member variables?

The answer to this question lies in the concept of padding and word boundary.

Member variable | Address Of storage

int a | add1

char b | add1 + 4

int c | add1 + 8

Even though the size of char is 1-byte b is allocated 4 bytes (3  bytes padded ).

Word boundary is a defined as boundary/end of something known as a **word** which is the smallest unit of memory that can be accessed efficiently. (machine dependent) In a 32 bit machine, that word boundary is 4 bytes. For efficiency reasons, only variables that are aligned at word boundary are easy to access. Thus in general, elements are to be allocated at a word boundary, which can provide much faster memory access. This explains the size of struct being 12 = 4 + (1 + 3) + 4

**Conclusion**

From this walkthrough, we have learnt an overview of how different parts of a program are stored in memory, how struct members and functions are stored. We learnt about the concept of padding and word boundary and its effects on the size of structs.

**Something to ponder upon**

Try to code this statement in a C++ program: int x = 'abcd';  Check what happens when you run this and try to reason about it based on above discussion.

 PreviousNext

© All Rights Reserved

[IITBombayX Home Page](http://iitbombayx.in/)

1. [About](http://iitbombayx.in/about)
2. [Research](http://iitbombayx.in/research)
3. [FAQs](http://iitbombayx.in/faq)
4. [Contact Us](http://iitbombayx.in/contact_us)

* [MOOCs Copyright](http://iitbombayx.in/mooc-copyright-guidelines)
* [Terms of Service and Honour Code](http://iitbombayx.in/honor)
* [Authorization Document](http://iitbombayx.in/authorization)
* [Privacy Policy](http://iitbombayx.in/privacy)

[Powered by Drupal](https://www.drupal.org/)

[Creative Common License](https://creativecommons.org/)

[](http://open.edx.org/)