

# Re: In memory layout of the C++ program

---

*Source:* <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2005-05/msg00084.html>

---

- *From:* "Mark Randall" <[markyr@xxxxxxxxxx](mailto:markyr@xxxxxxxxxx)>
  - *Date:* Tue, 3 May 2005 12:55:35 +0100
- 

"DeltaOne" wrote:

> Hi,  
> I read the common object file format. But i am not pretty sure how  
> machine language is converted to exe file. How can i generate the  
> virtual address. I wann to know how the memory layout of a C++ program  
> in done in memory i.e virtual memory and the physical memory also. With  
> the VC++ compiler or some other compiler also i can get the assembly  
> code for that program but how is this program converted to exe file and  
> why is it converted as a assembly program is directly converted into the  
> binary code , so why need to convert it into the exe file or other  
> formats?Can any one suggest some links also.

Its unclear what you are asking, the physical (how it appears in physical RAM) can change, dependant on where the operating system wants to put it and if the process memory lies across any boundaries.

I dont know what you mean by other formats, on windows at least you have PE32, Public Executables, and DLL's as the only things that can run as native code.

Your fricken grammar is terrible, I hardly have any idea what you want, so sorry if this is all way OT.

In short:

[PE HEADER] – Contains information about the sections, code entry points etc  
[data (.DATA)] – Stores uninitialised data  
[statics (.RDATA)] – Stores initialised data, hence if you are a muppet and do article (i) you are screwed.  
[code (.TEXT)] – this is where the code is, it has a specified entry point in the PE header  
[pie (.TASTHEY)] (NB: This may not exist, or may not be in this order)

Article i:

-----

```
if (!strcmp(szPassword, "pAsSwOrD")) {  
// do something important  
}
```

## Re: In memory layout of the C++ program

"pAsSwOrD" shows up in plain text in the .RDATA section.

Anyway....

You then have your [Stack] – This is what grows whenever you put a function into execution, that goes.

[Function 1 Vars]  
[Stack Pointer]  
[Function 2 Vars]  
[Stack Pointer]  
[Function 3 Vars]  
[Stack Pointer]

And you then have your heap, which grows from the other end of the 2GB of virtual memory assigned to your process.

– – –

AS for what I think your other points are...

binary code is just a different way of showing assembly code – if I told you that 00000001 meant 'add' and 00000010 meant 'multiply' and perhaps 00000011 meant CPU register 1, 00000100 is CPU register 2.. well you get the idea.

so: 00000001 00000011 00000100

would be add the value in register 2, to the value in register 1. In assembly this would show up as:

```
add [reg1], [reg2]
```

If you would care to write back and uh.. make yourself a tad more clear?

– Mark R

---

- **References:**

- ♦ ***In memory layout of the C++ program***

- ◊ *From: DeltaOne*

- Prev by Date: ***Re: start off with port programming ....***
- Next by Date: ***Re: start off with port programming ....***
- Previous by thread: ***In memory layout of the C++ program***
- Next by thread: ***In memory layout of the C++ program***
- Index(es):

Re: In memory layout of the C++ program

- ◆ Date
- ◆ Thread