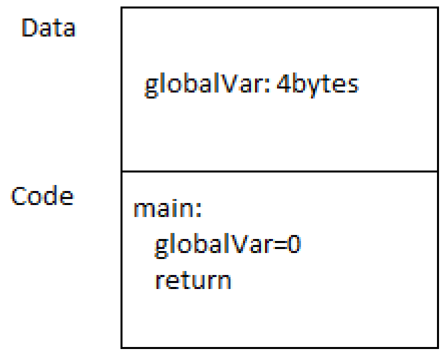
Program Structure

Let us take an example of a simple C program and see how it looks when loaded into memory for execution.

Here is a simple C program:

```
int globalVar;
void main()
{
    globalVar = 10;
}
```

Usually the main function returns int but for simplicity it is declared as void here. There are two things to notice in this program. There is a global variable and a function. The function contains executable code and the global variable contains some data. If this program is translated into machine instruction and loaded into memory, this will get space in the random access memory for data and code parts.
This can be represented by a simple diagram like -



The data part consists of some bytes allocated for the global variable and the code part contains bytes allocated for and loaded with the instruction which is generated from the code of the function. CPU executes these instructions which manipulate the data.

Lets see the actual assembly code which is generated for the above program:

```
    .comm   globalVar,4,4
    .text
.globl main
    .type   main, @function
main:
    pushl   %ebp
    movl    %esp, %ebp
    movl    $10, globalVar
    popl    %ebp
    ret
```

Comments on the generated assembly code:

```
# declaration of global variables.
# .comm is a way to declare the uninitialized data.
# The assembler will allocate the equivalent space in .bss segment.
    .comm  globalVar,4,4
# .text segment starts now
    .text
# export the function name main so that it can be used outside of the file.
.globl main
    .type   main, @function
# the code generated for the function main
main:
    pushl   %ebp
    movl    %esp, %ebp
    movl    $10, globalVar
    popl    %ebp
    ret
```

Section and Segment

Section is area in object file which contains information which is used while linking. It contains data that is used by linker to produce executable binary. For example- a section can contain program code, global variables, relocation table etc. And a segment is like a section but this is what is actually loaded into memory for program execution. An executable binary may contain many segments which get loaded by loader when you run the program. The sections and segments are named like .text, .data, .bss etc.

Here are few examples of section/segment which is mostly used while generating assembly code from C code:-

.text => This contains executable instructions.
.bss => uninitialized (all bits set to zero) data. These are typically static variables and global variables which are uninitialized. The program can write data into this.
.data => This contains initialized global variables. for example, if you declare globally, int x = 20; then it will go to the data segment. The program can later write data into this section.

.bss and .data almost similar. The only difference is that of initial values of memory allocated while the program is loaded. For .bss segment all bits will be set to zero and for the .data segment, it will be initialized according to what declared in the C code while declaring the variable.

.rowdata - this contains read only data. This is loaded into memory when the program is loaded and the program can not change this. This, typically, contains string const.

There are few sections which are present in object and executable files but not loaded into memory when the program is executed. For example **.debug** segment which contains information which is needed by debugger only. Few sections exist only in the object module and later merged into other sections. for example **.common**

‹ Introduction (/cin/introduction.html)
up (/cin/cin.html)
Stack and Local Variables › (/cin/memorymanagement.html)
