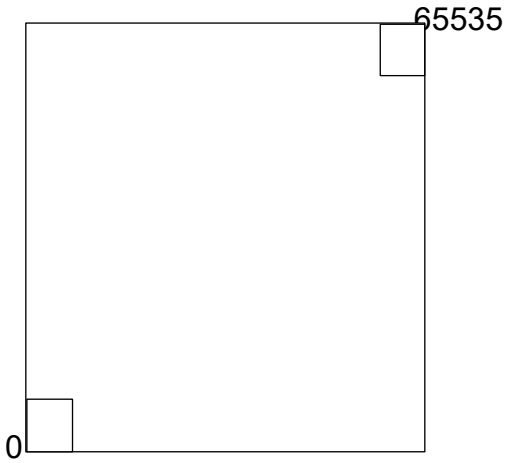
Stack & Heap Memory

Let us discuss static vs dynamic memory allocation. For that you should understand memory.

8	9	10					
0	1	2	3	4	5	6	7

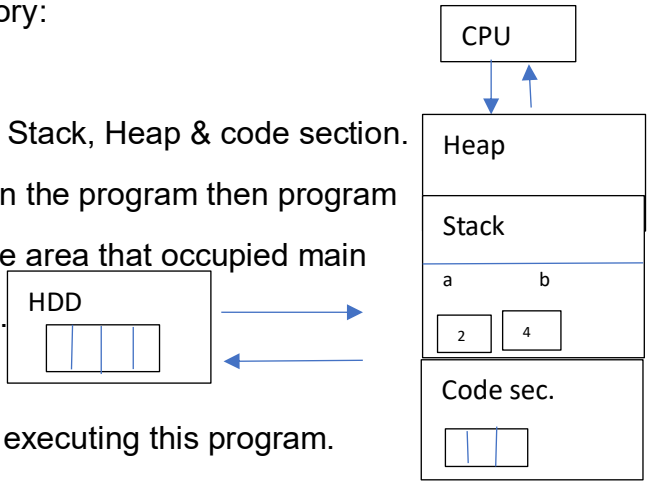
Just you think this block represents as memory. Memory is divided into smaller addressable unit, that is called byte. Let us assume these boxes are bytes. Every byte is having address…address is this 0,1,2,3,4,5,6…like this. Diagram is two dimension but address is single dimension. So, addresses are linear not like co-ordinate system[x,y].



So, I am assuming my main memory is 0-65535→64X1024=64KB. Now a days you are using 4/8/16 GB. To understand I am taking 64KB memory. Start from 0 and last address 65535 byte. Every byte is having address. If you have larger size RAM is 8/16 GB. Then entire memory not use as a single unit that is divided into manageable piece that is called segment, usually 64KB. So, size of the pieces 64 KB.

Next how the program utilize main memory:

Main memory divided into three section- Stack, Heap & code section. I have a program on the hard disk. If I run the program then program come to main memory from HDD. So, the area that occupied main memory that section called code section. That would may not fixed. It depends on size. Once it is loaded then CPU is start executing this program.



Remaining memory divided into stack and heap.

Now let us learn how this stack and heap works?

This is my example code:

```
void main ()
{
        int a;   ------→2 byte          //assume here
        float b;  ----→4 byte



}
```

**listen carefully: in C/ C++ programing…. number of bytes taken by integer/float depends on compiler, OS or Hardware. We say mostly depends on compiler.

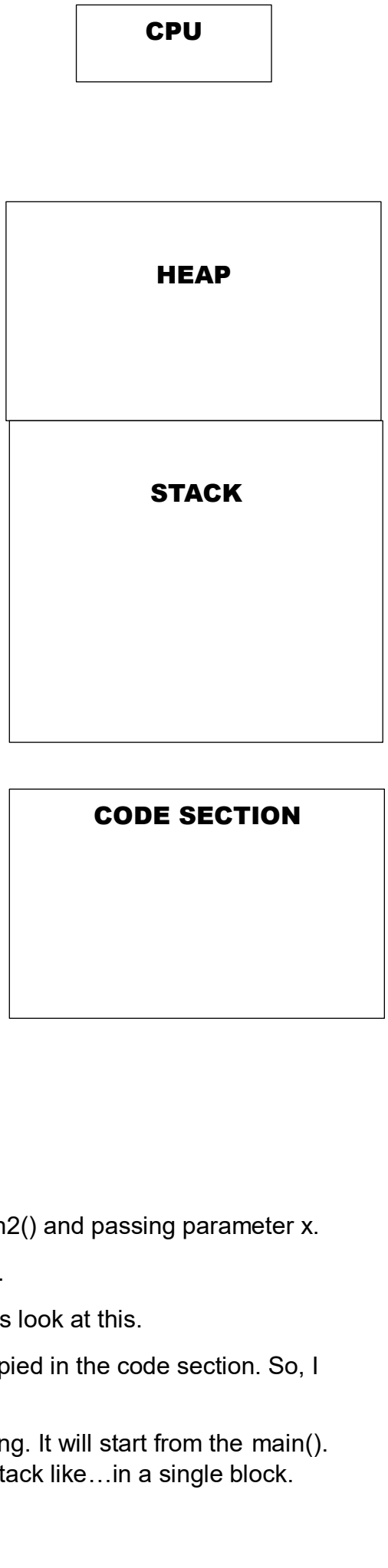
If you take Turbo C which is 16 bit compiler it takes 2 byte for integer. If you use Dev studio or codeblock then integer will take 4 byte….it's a 32 bit compiler. So, integer can take 2 byte also 4 byte also. So, I am assuming integer 2 byte here.

As per 2 byte + 4 byte → 6 byte allocated inside the stack

This block of memory that is belongs to main function that is called stack frame or activation record of main function. So, whatever variable should declare inside the program or function the memory for this variable created inside the stack. So, allocated inside stack. The portion of memory inside stack that is called activation record. Size of the main memory required the function decided by compile time only. This is static memory. So, what is static? Size of the memory required during compile time that is static. so, it is before run time.

Here I have sample piece of code:

```
void fun2(int i)
{
        int a;
        -----
        ----
}
_____________________________
void fun1( )
{
        int x;
        fun2(x);
        ---------
}
_____________________________
void main( )
{
        int a;
        float b;
        ----------
        ----------
        fun1( );
}
```



Description:

main() is having two variables and calling fun1().

Then fun1() is having one local variable and calling fun2() and passing parameter x.

fun2() is taking parameter i and having own variable a.

then how the memory allocated for this function…let us look at this.

First when I run this program then machine code is copied in the code section. So, I will write all function in code section.

Let us start executing. when the program start executing. It will start from the main(). So, memory for the variable a, b allocated inside the stack like…in a single block.

main() call fun1() then control goes to fun1(). 1st things inside the fun1() the variable declared is int x; then x allocated in another block of the stack. This portion belongs to fun1(). Top most activation record belongs to currently activated / control function. fun1() call to fun2() then control goes to fun2(). That is having two variable one is parameter another is local variable. One region allocated for local variables i & a. that's region belongs to fun2(). Now presently fun2() is running and top most activation record is fun2(). So, currently executing fun2(). So, we start from main() then fun1 and fun2(). Main() activation record as it is then memory for fun1() and fun2() allocated.

Now let us continue execution:

fun2() is currently executing function. When fun2() finish or terminated then control goes fun1(). After this point it will comeback inside the fun2() statement of fun1(). What happened activation record of fun2()? This will be deleted.

When fun1() finish then control goes back to fun1() statement of main(). Activation record of fun1() will be deleted. So, record of this fun1() removed from stack.

Then main() also end then activation record of main() removed from stack.

The way activation record created like this and deleted like this mechanism is called stack. Activation record created….top…top..top… and deleted from top…then top..top. that's why this section of memory behave like stack during the function call. So, this portion of memory named as stack.

That's all…this is the main memory use, stack used during the function call.

One important thing is observed. How much memory is required by the function depends on number of variable and their sizes. This is decided by compiler only. This memory automatically created and automatically destroyed. So, programmer doesn't have to do anything for allocation or destruction. Just programmer has to declare variables.

Next topic Heap and see how dynamically memory allocation?

Let us learn how heap memory utilized by program:

Heap: Heap means just piling up things. So, objects placed on top of each other. Heap is used in two cases one properly organized things like a tower/ pyramid like things then also it is a heap another unorganized way like a tower / pyramid like also called heap.

Heap can be used organized or un-organized. Here heap is used unorganized. Stack is organized. Heap memory treated like resource. So, application is used as resource. When required you take the memory and not required just release the memory. This is the practise we must do while dealing with heap memory. Program directly access heap memory. It can directly access code section and stack not heap.

Then how do the access heap memory? Using pointer.

Let me so you how we can get some memory inside the heap with help of pointer.

Sample code explanation:

I have take a pointer inside main() as int *p.

1st question how many byte pointer take?

For making it simple I say pointer is 2 bytes.

The moment of memory taken pointer depends on size of integer usually. If size of integer is 2 byte as per compiler. Let us assume it is taking 2 bytes.

Where the memory pointer should be allocated?

Already I told you whatever variable declared in function they will occupied memory inside the stack.

So, this the pointer. It takes 2 bytes. This is the activation record for main function.

Now I want to allocate a memory heap. So, How much memory I want to allocate?

I want to create a array of integer size 5.

Like p=new int[5];

This new statement allocate memory in heap of size of 5.

This pointer will point the memory. Suppose the starting address of array is 200 then 200 store in p. this is the method allocating memory in heap. Wherever you see 'new' mean allocated in heap memory. In C++ it is new.

But if I write same things in C language I have to use malloc function. I want 5 integer then I will write

P=(int*) malloc(2*5);

int*---. Type cast

2---int 2 bytes

So, program cannot directly access heap. It has to access pointer and pointer will give the address of that memory and program will reach that and access this integer.

Heap memory treated as resource after some time you don't need that array. You want to make p as NULL.

p=NULL;

p hold '0'

You don't need p is any more. Nothing will be pointing on this memory. Then what about this memory. Is it lost or gone? No, it is not deallocated. It's a good practice when not required this memory you have to de-allocated. Then you should deallocate this memory:

delete [ ]p;

so, heap memory explicitly requested and explicitly released or dispose.

Otherwise, if you not releasing it then memory will be still belonging into program and that memory can't be use again. It causes loss memory and loss of memory is called

as memory leak. If you continue same things in the program and not release then heap memory will be full. Their may be no free space on heap memory. So, whenever heap memory is not needed then simply release it.

Let us conclude, we have seen how the static memory allocation done form the stack and how dynamic memory allocation done from the heap.