Hack `postgres` Source Code: Vol I

Chapter 1 : C & Rust

1.41 Dynamic Memory Allocation - C

We can use memory from `heap` to store data using C program.

But, why do we need `heap` for storing data? And, what is the problem with `stack` memory?

C follows `static memory allocation` process to assign fixed memory to types during compile time. For instance, if we declare `int a[5];` it allocates memory block of 20 bytes to store only integers in those 20 bytes. This can't be changed during runtime/execution-time. It is fixed. So, if we use less memory, there would be wastage. If we want to store more integers in the middle of the program, C doesn't allow to do that. If we force it, C throws an error. All this things happen in `stack` memory.

So, if we want to allocate memory dynamically during execution of program, do we use heap!!??

Yeah.

But, how can we access that heap memory?

C has few functions to access and manage heap memory. We can use `malloc` and `realloc` to deal with memory allocation and `free` to manage allocated memory.

Can you elaborate one by one.

Here comes `malloc` . It takes a number as its input parameter to use it to allocate the amount of the memory that we need. And, it return a `void` pointer pointing to that allocated memory. It returns `void` pointer.. it's because.. it allows us to store any type data (int data, float data) etc., If doesn't return `void` pointer, we would end up having a memory where we can only store one type of data.

Explain with an example..

```
#include<stdio.h>

// malloc function prototype is available in `stdlib.h`
#include<stdlib.h>

int main() {

    /*
    * Input parameter takes only `unsigned int` type values.
```

```
    * In other words, `size_t` type values.
    *`size_t` is typedef of `unsigned int` in `stdlib.h` file.
    * In this case, it returns a memory block of size equals to
    * 8 bytes and in contiguous manner.
    * If there is no sufficient memory in heap, it just returns NULL
    * pointer.
    * `malloc` returs a `void` pointer pointing to first byte of
    * 8 bytes.
    * As we want use this 8 bytes of memory to store int type data,
    * we typecast `void` pointer to `int` pointer.
    * Now, *ptr is the pointer which is pointing to first byte of
    * 8 bytes in heap.
    */
    int *ptr = (int*) malloc(2*sizeof(int));

    // Using heap memory
    *ptr = 10;
    *(ptr+1) = 20;

    printf("Value in first 4 bytes - %d", *ptr);
    printf("Value in next 4 bytes - %d", *(ptr+1));

    return 0;

}
```

If you want to add one more integers in the above program, you can use `realloc` function. We need to pass a pointer pointing to previously created memory block and a number which defines size of memory block(in our case, bigger than previous one). After creating the memory block, it moves data from old memory block to new memory block using the pointer that we passed as input parameter.

```
#include<stdio.h>
#include<stdlib.h>

int main() {

    // heap memory allocation to store 2 integers
    int *ptr = (int*) malloc(2*sizeof(int));

    // Using heap memory
    *ptr = 10;
    *(ptr+1) = 20;

    printf("Value in first 4 bytes - %d\n", *ptr);
    printf("Value in next 4 bytes - %d\n", *(ptr+1));

    /*
    * Reallocating memory to store 3 integers.
```

```
    * realloc allocates 12 bytes of memory. If there is no
    * enough heap memory, it just return NULL pointer.
    * After typecasting, that memory is used to store only
    * integer types.
    * After getting 12 bytes, it uses `ptr` to get 2 integers
    * and store them in first 8 bytes.
    * We can use last 4 bytes to store our third integer.
    */

    int *rptr = (int *) realloc(ptr, 3*sizeof(int));

    *(rptr+2) = 30;
    printf("Value in first 4 bytes - %d\n", *rptr);
    printf("Value in next 4 bytes - %d\n", *(rptr+1));
    printf("Value in next 4 bytes - %d\n", *(rptr+2));

    // Check data in old memory block once again after realloc
    printf("Value in first 4 bytes - %d\n", *ptr);
    printf("Value in next 4 bytes - %d\n", *(ptr+1));

    return 0;
}
```

We have this `free` function in and it is used to cleanup the data in memory block used by us in heap.

```
#include<stdio.h>
#include<stdlib.h>

int main() {


    // heap memory allocation to store 2 integers
    int *ptr = (int*) malloc(2*sizeof(int));

    // Using heap memory
    *ptr = 10;
    *(ptr+1) = 20;

    printf("Value in first 4 bytes - %d\n", *ptr);
    printf("Value in next 4 bytes - %d\n", *(ptr+1));

    // reallocating memory
    int *rptr = (int *) realloc(ptr, 3*sizeof(int));

    *(rptr+2) = 30;
    printf("Value in first 4 bytes - %d\n", *rptr);
    printf("Value in next 4 bytes - %d\n", *(rptr+1));
    printf("Value in next 4 bytes - %d\n", *(rptr+2));
```

```
    // Check data in old memory block once again after realloc
    printf("Value in first 4 bytes - %d\n", *ptr);
    printf("Value in next 4 bytes - %d\n", *(ptr+1));

    // Cleaning up memory blocks
    free(rptr);

    return 0;

}
```