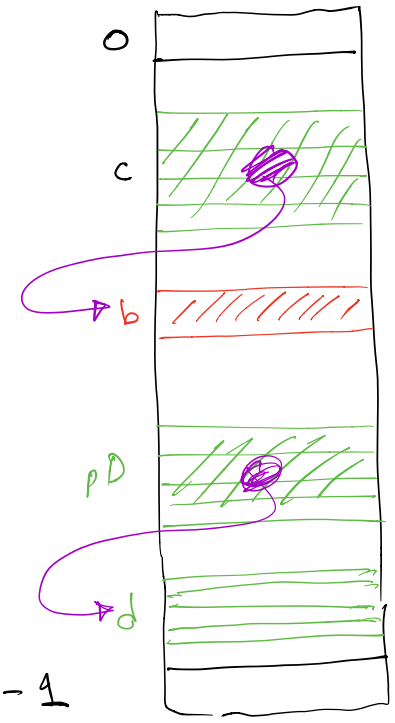
```
char * c;
c = 'A';  X
c = 0xFA1028;  X
```

```
char b;
c = &b;
```

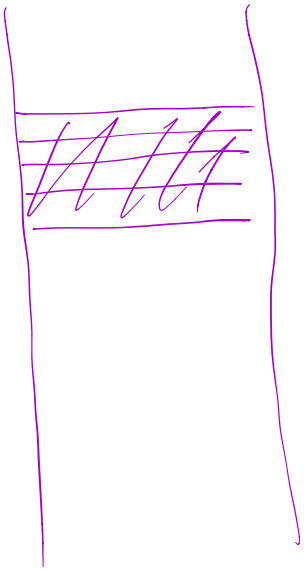
```
double * pD;
double d;
pD = &d;
```

```
cout << pD << endl;
cout << *pD << endl;
```

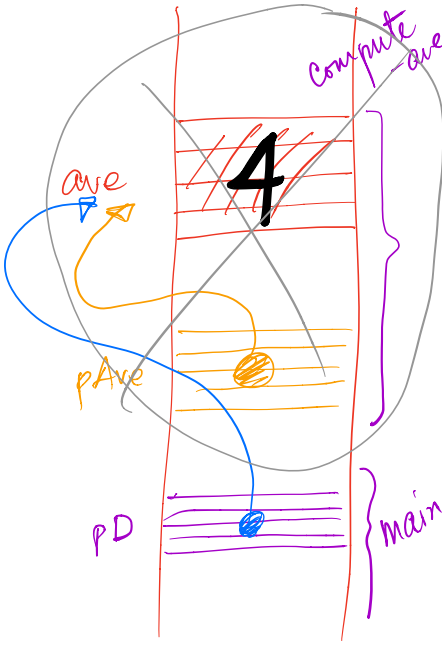


```
int compute_ave()
{
    double ave;
    ...
    ...

}
```



```
double * compute_ave()
{
    double ave;
    ...
    ...
    double * pAve = &ave;
    return pAve;
}
...
main()
{
    double *pD = compute_ave();
    ...
    ...
    cout << "Average = " << (*pD) << endl;
}
```



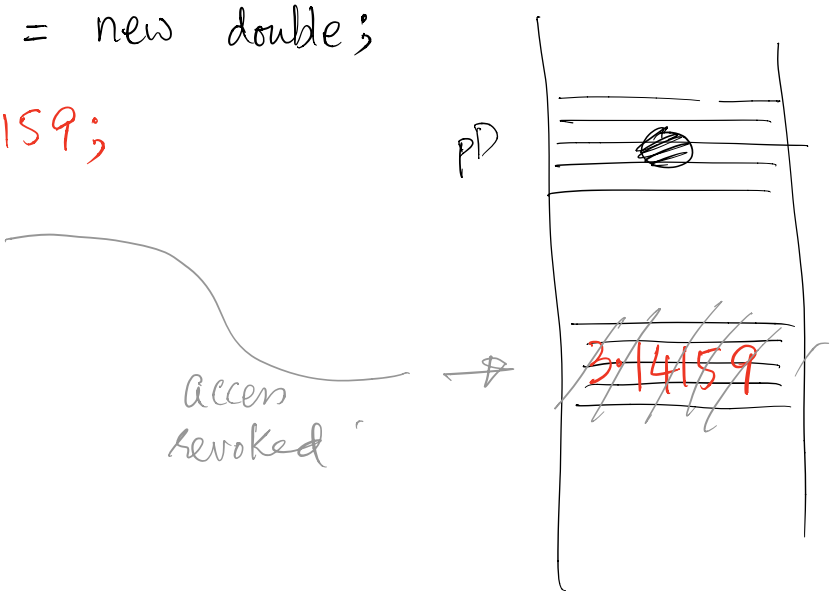
Dynamic Memory Allocation

- new (keyword)
- You have to deallocate dynamic memory!

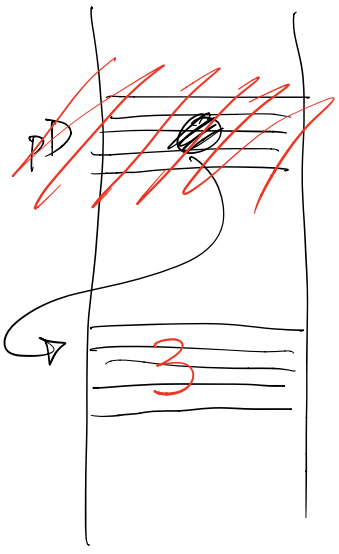
delete (keyword)

- Memory leaks

```
double *pD = new double;
*pD = 3.14159;
delete pD;
```

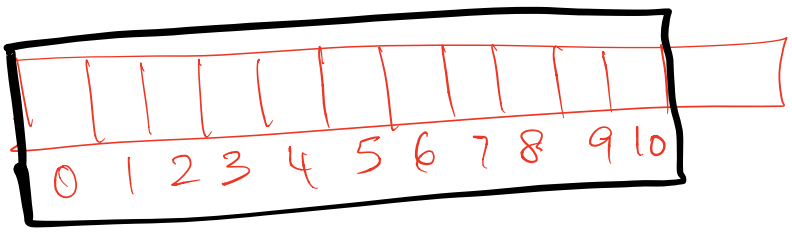



```
Void compute_ave( )
{
    double *pD = new double;
    *pD = 3;
    ...
    return;
}
```



access revoked

memory leak



Allocate an array of integers:

```
int c[10];
int *c = new int[10];
```

Static allocation

- # slots fixed
- Stack
- don't need to deallocate
- no danger of memory leak

Dynamic allocation

- # slots can be changed at runtime
- Heap
- Need to deallocate
- Memory leak

```
delete[] c;
```

- Difference between static and dynamic allocation.
- Pointers
- Interplay with function calls.

File I/O

- text

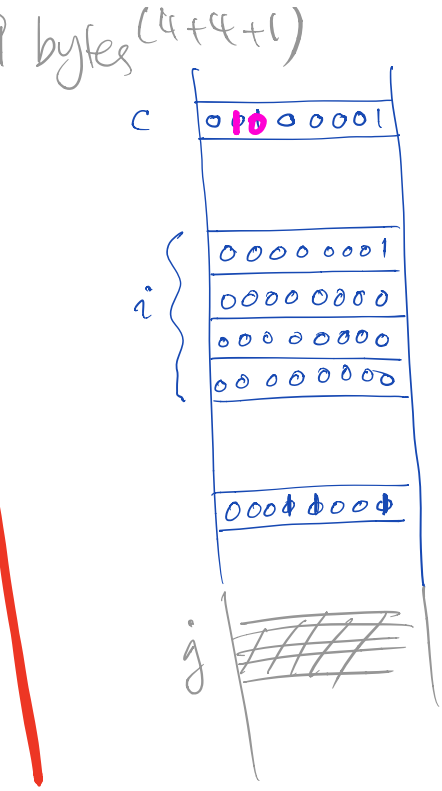
ifstream

>>

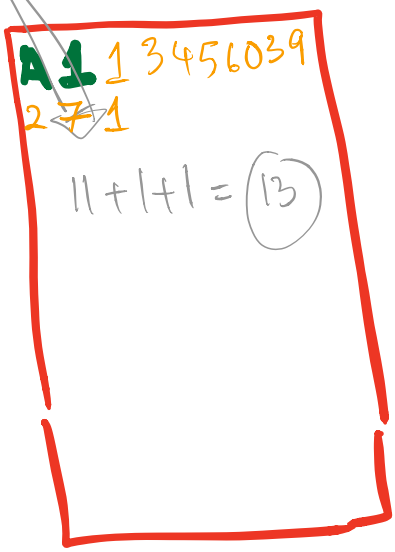
ofstream

<<

```
char c = 'A';
int i = 1;
int j = 1345603927l;
```




```
f << c;
f << i;
f << j;
```



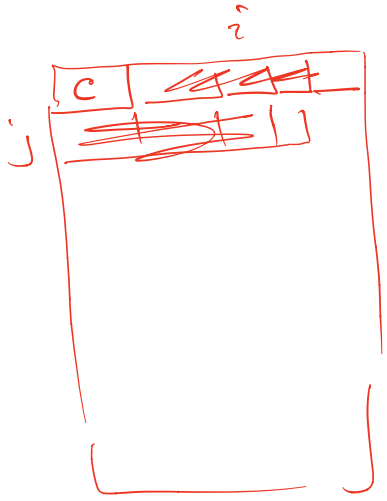
ofstream f (text)

```
f << c << endl;
f << i << endl;
f << j << endl;
```

```
f << c << " ";
f << i << " ";
f << j << " ";
```

Binary Mode File I/O

c →
i →
j →



- ifstream f("file.name", ios::binary);
- ofstream f("file.name", ios::binary);
- f.write(char*, size_t);
- f.read(char*, size_t);
- Don't use << and >>