Lecture 6.2

Dynamic Memory Allocation

What is memory allocation?

- To allocate memory is to reserve a space in memory for the variables we are declaring.
- In C++, space in memory for variables may be either statically or dynamically allocated.

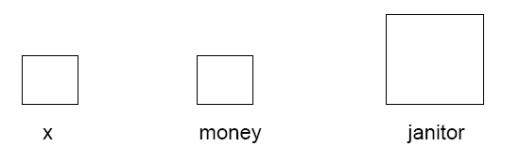
Statically allocated?

- Statically allocated objects are those we have been using through out this course.

  int x;

  float money;

  Employee janitor;



Sizes in bytes of different variable types

```
Size of int     4
Size of char    1
Size of short   2
Size of long    4
Size of float   4
Size of double  8
Size of ptr     4
Size of *ptr    4
```

Statically allocated objects

- The compiler arranges the required space as it turns source code into a binary or executable program.
- Statically allocated objects that are of local scope are put into a memory space known as the stack.
- Statically allocated objects of global scope live in the global address space.

What happens if we don't know the size of an array?

- We could try to size the buffer or array to be large enough to hold the worst case (big enough to hold anything we should encounter)

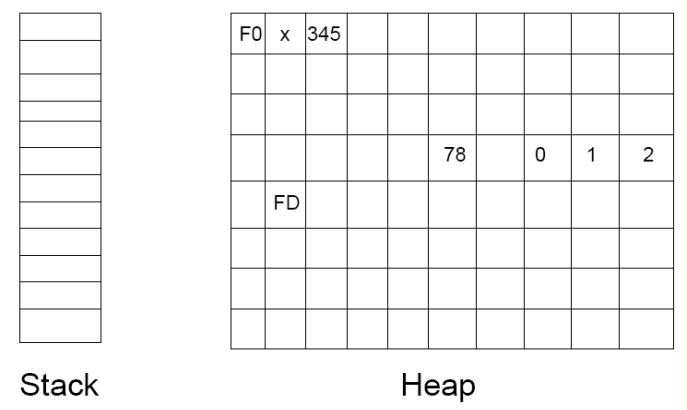
What's wrong with this strategy?

- It consumes memory unnecessarily.
  - This is less of an issue than in the past when memory was more limited, but still impacts overall system performance.
- No matter how much memory is statically set aside for our object, we can never be sure it will be large enough.

Dynamically allocate an object?

- The memory for the object comes from a pool of memory known as the **heap** or **free store** instead of the stack.
- The memory is allocated when the program is run instead of when it is compiled.

Stack and Heap

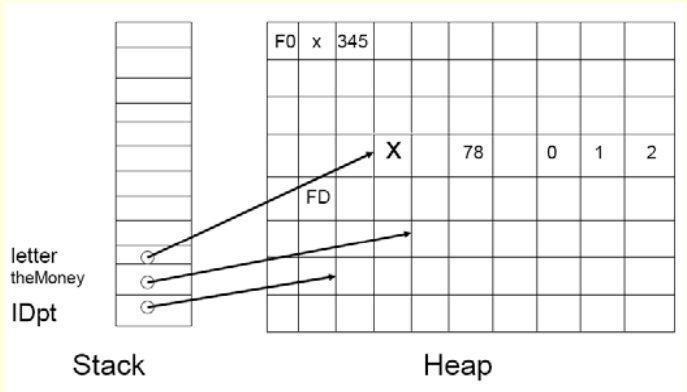


What is the syntax for doing this?

- You will have to use two operators:
  - new
  - delete
- Example:

```
int *IDpt = new int;
float *theMoney = new float;
char *letter = new char;
```


Stack and Heap with dynamic memory allocation



The “new” operator

- The "new" operator returns the address to the start of the allocated block of memory.
- This address must be stored in a pointer.
- “new” allocates a block of space of the appropriate size in the heap for the object.

The “new” operator (cont)

```
int *myPtr = new int;
```

- Notice that the reserved block of memory is anonymous; it has no identifier (name).
- dynamically allocated memory is accessed indirectly via a pointer.
- If there’s no memory available “new” will fail.
- “new” will throw a "bad_alloc" exception in this case.

Initializing a dynamically allocated Object

- Dynamically allocated objects contain whatever random bits happen to be at their memory location.
- Therefore a value must be assigned **before** use.

```
int *IDpt = new int;
*IDpt = 5;
```


Initializing a dynamically allocated Object (cont)

- There is another syntax we can follow when using the “new” operator:

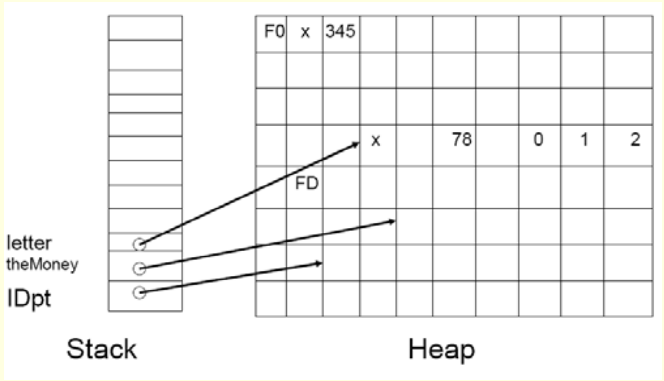
```
int *IDpt = new int(5); //Allocates an int
                        //object and initializes it to value 5
char *letter = new char('J');
```


“delete” operator

- Dynamically allocated objects must be explicitly deleted when no longer used by a program.
- “delete” releases the memory used by the object.
- That memory is then available for reuse.

```
delete IDpt;
delete theMoney;
delete letter;
```

Stack and Heap with dynamic memory allocation



What happens with statically allocated objects?

```
void myFunction (int x)
{  char m = 'r';
   cout << m ;
}
 int main (void)
{
  myFunction(3);
}
```

into the stack

popped off the stack

int x 3

char m r

cout <<m;

x

m

These are deallocated now

What happens with dynamically allocated objects?

```
void myFunction (int x)
{  char *m = new char('r');
   cout << m ;
}
 int main (void)
{
  myFunction(3);
}
```

into the stack

popped off the stack

int x 3

char* m → r

cout <<m;

x

m → r

The location where r is stored does NOT get deallocated!

What is the problem?

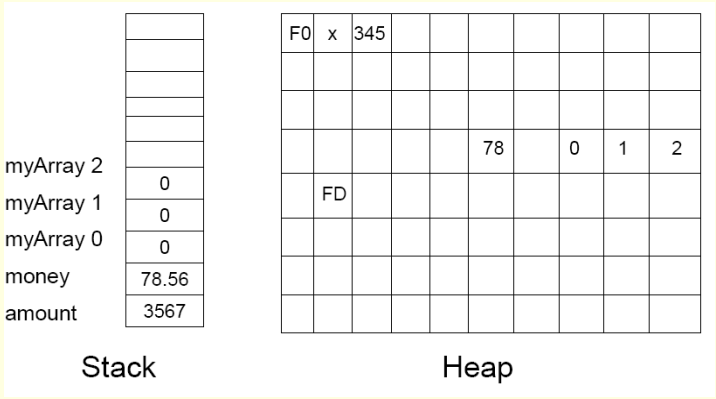
- The dynamically allocated object still exists!
- And now we no longer have a pointer to it so we cannot release it!
- This is known as a **memory leak**

Function with static memory allocation

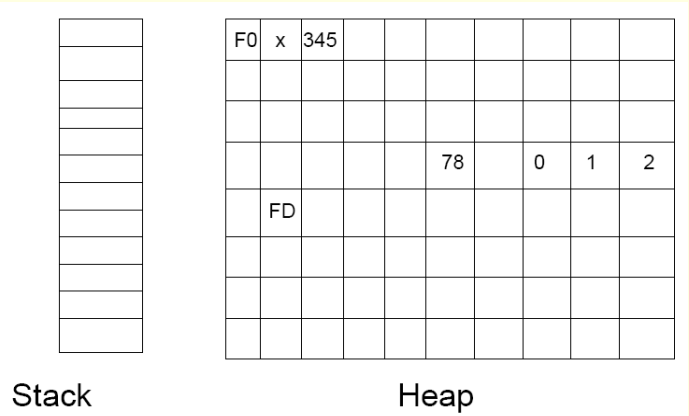
```
void Function ( int amount) {
   float money = 78.56;
   char myArray[3]={0}
}

int main (void) {
   Function ( 3567);
   return 0;
}
```


Stack and Heap when function is compiled



Stack and Heap when function goes out of scope



Function with dynamic memory allocation

```
void myFunction (int x){
   char *m = new char('r');
   cout << m ;
}

int main (void)
{
   myFunction(3);
}
```