Program

- The Dallas Symphony is trying to figure out which instrument family has the cheapest instrument to replace for its upcoming budget meeting.

Dynamic Memory

Lecture Overview

- Lecture
  - Dynamic memory
- Before We Code
  - malloc, free functions
- Sample Programs

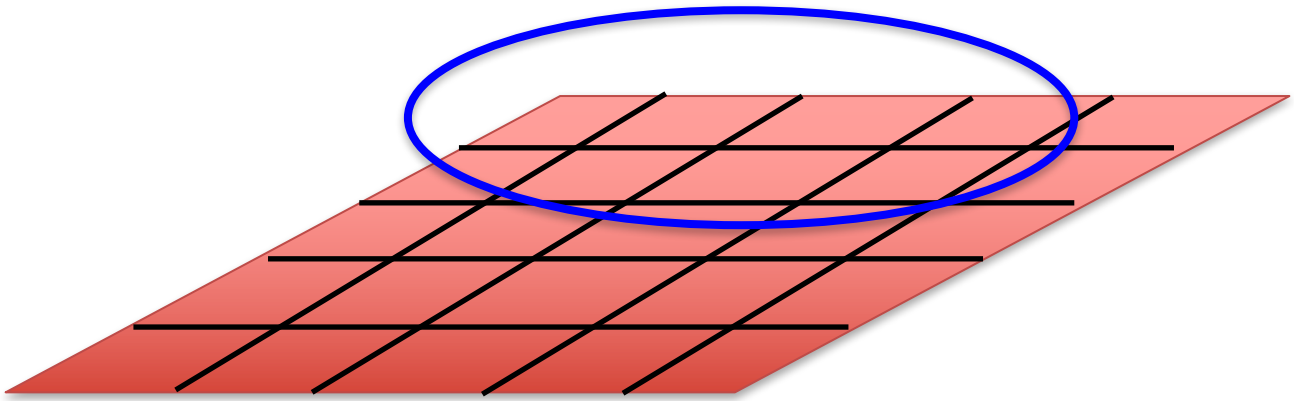
LECTURE

Dynamic Memory

- The idea of dynamically allocating memory is to set aside enough memory for your program at run time
  - The programmer didn't know a specific amount of space to allocate, so it is figured out at run time
    - Maybe the user needs to enter how much
- The main functions associated with this are:
  - malloc (allocates memory)
  - calloc (like malloc, but intializes all space to 0)
  - realloc (change size of previously allocated memory)
  - free (frees allocated memory)

Dynamic Memory

During the duration of our program, we can now actively decide to put aside memory for our program. For example, maybe the user decides to enter two grades. Our program can put aside 8 bytes for our 2 ints.

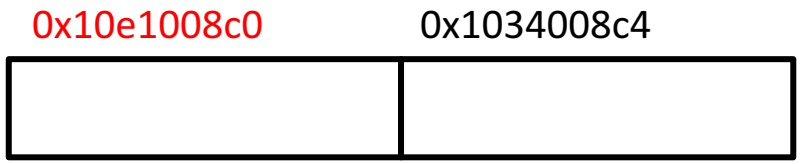


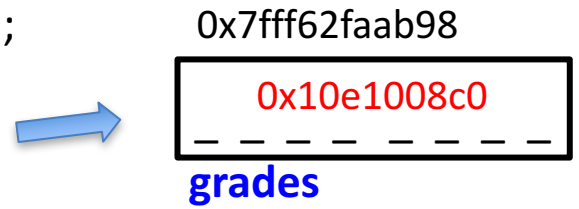
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
        int s, I;
        printf("Enter the number of students: ");
        scanf("%d", &s);
        int *grades=malloc(sizeof(int)*s);

        printf("Value in grades: %p\n", grades);
        printf("Address of first element %p\n", &grades[0]);
        printf("Address of next element: %p\n", &grades[1]);
        printf("Address of grades pointer: %p\n", &grades);
}
```

Addresses are 4 bytes apart (since they are ints)





grades is a pointer at enough space for 2 ints (8 bytes)

Output:

Enter the number of students: 2
Value in grades: 0x10fc008c0
Address of first element 0x10fc008c0
Address of next element: 0x10fc008c4
Address of grades pointer: 0x7fff6f792b98

Note: you should check if malloc returns NULL (meaning space was not allocated-I didn't do that here because of space)

You can put information into your dynamically allocated memory in both of the following ways:

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

int main(int argc, char **argv)
{
	int s, l;
	printf("Enter the number of students: ");
	scanf("%d", &s);
	int *grades=(int*)malloc(sizeof(int)*s);

	for(i=0;i<s;i++)
	{
		printf("Enter grade: ");
		scanf("%d", grades);
		grades++;
	}
}
```

(If you do this way, make sure to somehow keep track of where your pointer started)

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

int main(int argc, char **argv)
{
	int s, l;
	printf("Enter the number of students: ");
	scanf("%d", &s);
	int *grades=(int*)malloc(sizeof(int)*s);

	for(i=0;i<s;i++)
	{
		printf("Enter grade: ");
		scanf("%d", &grades[i]);
	}

}
```

Notice here I put (int) before malloc-sometimes you see this*

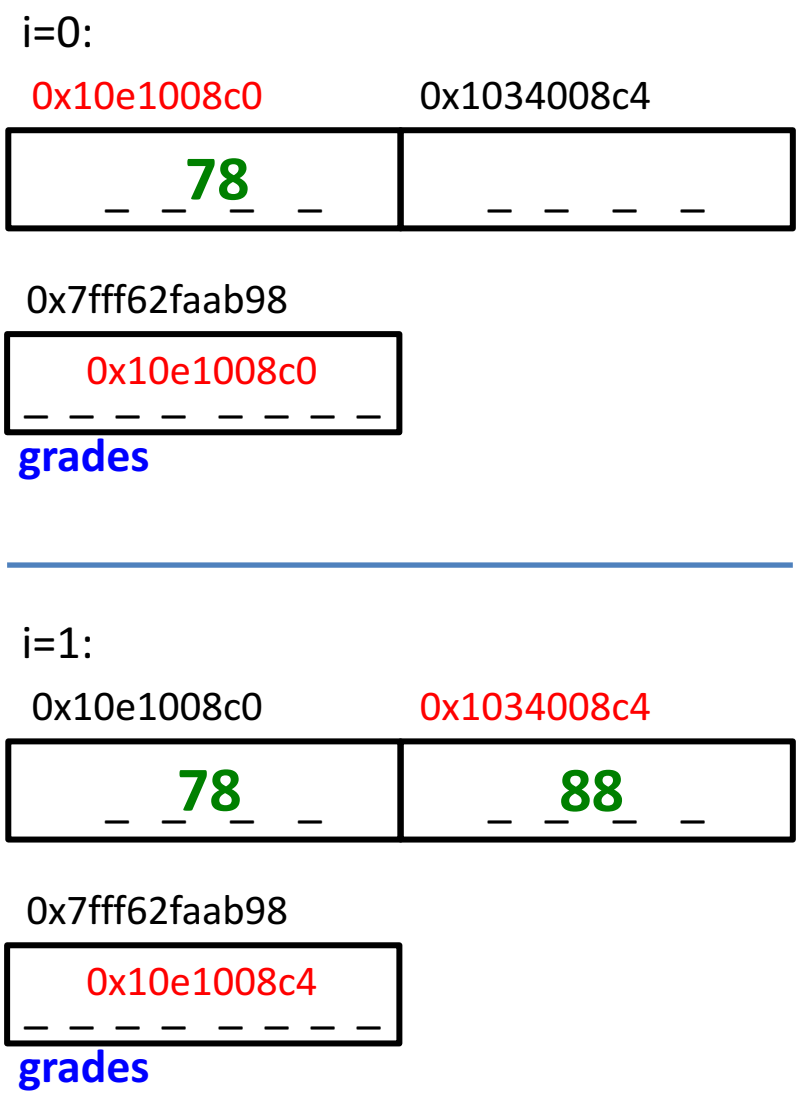
You can put information into your dynamically allocated memory in both of the following ways:

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

int main(int argc, char **argv)
{
        int s, I;
        printf("Enter the number of students: ");
        scanf("%d", &s);
        int *grades=malloc(sizeof(int)*s);

        for(i=0;i<s;i++)
        {
                printf("Enter grade: ");
                scanf("%d", grades);
                grades++;
        }
}
```

The value of the pointer changes-make sure if you do this way, you somehow keep track of where your pointer started



You can put information into your dynamically allocated memory in both of the following ways:

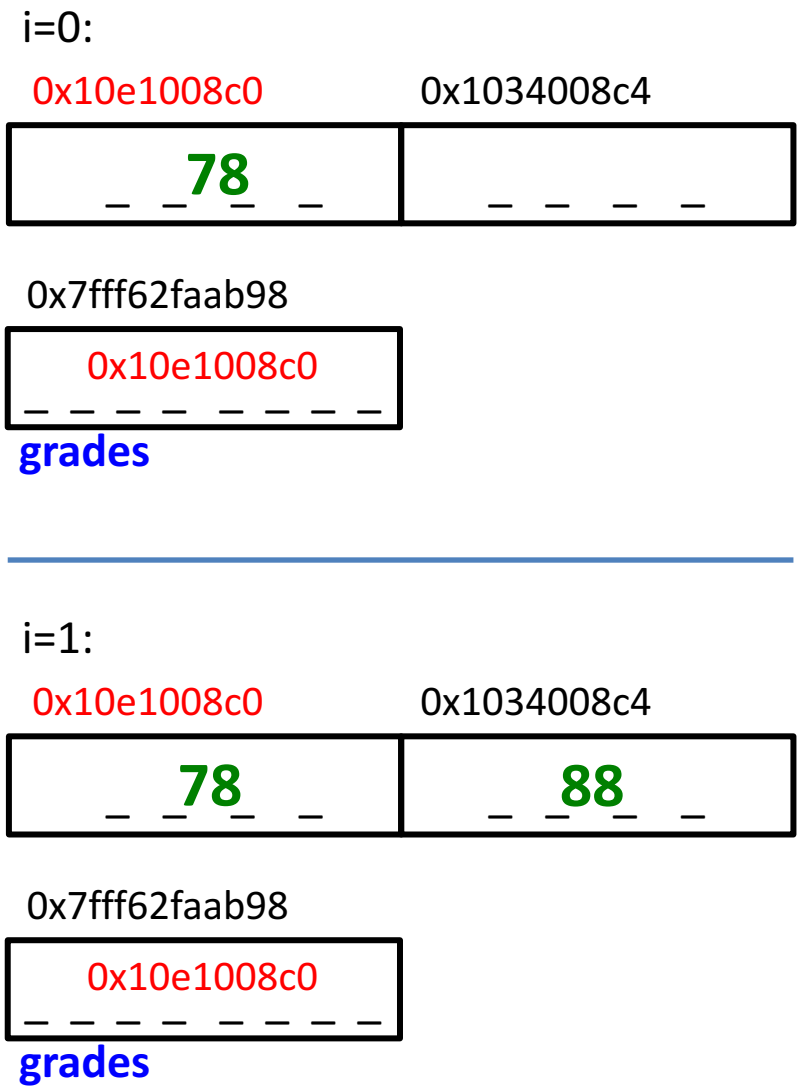
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
        int s, I;
        printf("Enter the number of students: ");
        scanf("%d", &s);
        int *grades=malloc(sizeof(int)*s);

        for(i=0;i<s;i++)
        {
                printf("Enter grade: ");
                scanf("%d", &grades[i]);
        }

}
```

The value of the pointer does NOT change



Dynamic Memory

- **Final notes:**
  - If you do a sizeof(grades), you are taking the size of the pointer itself so you will get 8 bytes.
  - An important problem that arises due to dynamic programming is the issue of memory leaks
    - We will talk more in detail about this in the future
    - This generally occurs when we allocate memory and never release it

BEFORE WE CODE

Before We Code

- Today you will see me use the *malloc* function and the *free* function.
  - *malloc* will return a **pointer** to a block of memory
  - *free* releases the memory when we are done with it
    - THIS IS VERY IMPORTANT
    - We don’t want memory leaks

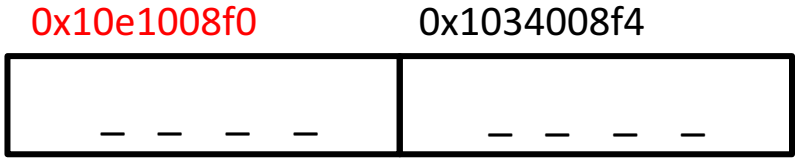
SAMPLE PROGRAMS

Program 1

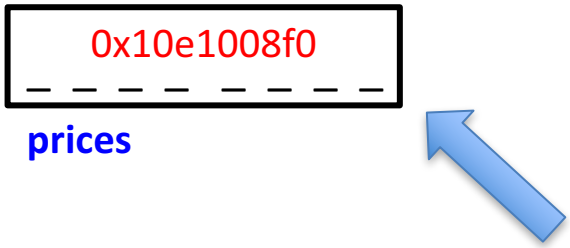
- Create a program that allows a user to type in the price of a certain number of items given by the user (use dynamic memory allocation)

Program 1

```
int n;
scanf("%d", &n);
float*prices=malloc(sizeof(float)*n);
```



assume n==2



This is returned by malloc. It is a float pointer, so the size is 8 bytes.
Its value is the address of a block of memory (the address of the first part)