POINTERS AND DYNAMIC MEMORY ALLOCATION (REVIEW)
Problem Solving with Computers-II

https://ucsb-cs24-sp17.github.io/

Read the syllabus. Know what's required. Know how to get help.

CLICKERS OUT – FREQUENCY AB
Announcements

- Midterm on Wed 04/26
- Study session today (04/23) from 7pm to 9pm in HFH 1132
Pointers

- **Pointer:** A variable that contains the address of another variable
- Declaration:  \( \text{type} \ * \ \text{pointer	extunderscore name}; \)

\[
\text{int} \ * \ p;
\]

\[
\text{p is a pointer to int}
\]

\[
\text{int} \ * \ p = \text{NULL};
\]

\[
\text{int} \ + \ p = 0;
\]

How do we initialize a pointer?
How to make a pointer point to something

```c
int *p;
int y;
p = &y
```

To access the location of a variable, use the address operator `&`

```c
y = 10;
p = 12;
int x = *p + 2;
```

```
*p is the same as y
```
Tracing code involving pointers

```c
int *p, x=10;
p = &x;
*p = *p + 1;
```

Q: Which of the following pointer diagrams best represents the outcome of the above code?

A. ![Diagram A](image)

B. ![Diagram B](image)

C. Neither, the code is incorrect

---

**Diagram A:**
- `x: 10`
- `p` points to `10`

**Diagram B:**
- `x: 11`
- `p` points to `11`
Dynamic memory: Make p point to an int on the heap

```c
int *p;
int y;
p = &y;
for (i = 0; i < 10; i++)
    int *p;
p = new int[10];
```
Two ways of changing the value of a variable

Change the value of y directly: \( y = 5 \);

Change the value of y indirectly (via pointer p): \( *p = 5 \);
Pointer examples: Trace the code

```c
int x=10, y=20;
int *p1 = &x, *p2 = &y;
p2 = p1;
int **p3;
p3 = &p2;
```

\text{prints 12}
**Pointer assignment**

```c
int *p1, *p2, x;
p1 = &x;
p2 = p1;
```

Q: Which of the following pointer diagrams best represents the outcome of the above code?

A. ![Diagram A]

B. ![Diagram B]

C. Neither, the code is incorrect
Mechanics of function calls on the run-time stack

double getAverage(int * sc, int len){
    double sum=0;
    for (int i=0; i<len; i++){
        sum+=sc[i];
    }
    return (sum/len);
}

int main(){
    int scores[5]={65, 85, 97, 75, 95};
    int len = 5
    double avg_score;
    avg_score = getAverage(scores,len);
    cout<< avg_score;
}
Dynamic memory allocation

- To allocate memory on the heap use the ‘new’ operator
- To free the memory use delete

```c
int *p = new int;
delete p;

new int(40);
```
Dangling pointers and memory leaks

- **Dangling pointer**: Pointer points to a memory location that no longer exists
- Memory leaks (tardy free) Memory in heap that can no longer be accessed
Q: Which of the following functions results in a dangling pointer?

```cpp
int * f1(int num){
    int *mem1 = new int[num];
    return(mem1);
}
```

```cpp
int * f2(int num){
    int mem2[num];
    return(mem2);
}
```

A. f1
B. f2
C. Both

```cpp
int * p = f2(42);
```

---

Diagram:
- Stack
- `p` is a dangling pointer
- `mem2` is deallocated after `f2` returns
- Scenario for a dangling pointer
Rewrite the code using dynamic arrays.

```cpp
double getAverage(int *sc, int len){
    double sum=0;
    for (int i=0; i<len; i++){
        sum+=sc[i];
    }
    return (sum/len);
}

int main(){
    int scores[]={65, 86, 97, 75, 99};
    int len = 5;
    double avg_score;
    avg_score = getAverage(scores,len);
    cout<< avg_score;
}

int *scores = new int[5];
// Some code to initialize values
// Code to add 1 element more than the current capacity
int *tmp = new int[6];
int k;
copy(scores, scores+5, tmp);
delete [] scores;
scores = tmp;
```
Write the declaration of the allocate space function

```java
int allocate(int n) {
    int *sc, *s[];
    sc = new int[n];
    return (n);
}
```

If scores should point to a dynamic array of size n, where n is input by the user.

```java
class bag {
    private:
        int data[30];
    public:
        // No copy constructor provided
        ...;

    int size() { return n; } // n is number of elements in bag
}
```

Stack

```java
bag b(10);
stack push(b);
```

If the dynamic copy constructor was used

Shallow copy
DEMO

- Dynademo.cxx (Program to demo dynamic arrays)
- How to use valgrind to detect memory leaks
- Debugging segfaults with gdb and valgrind
Next time

- Chapter 4 (contd): Bag class with dynamic arrays, intro to linked-lists