Dynamic Memory - General: Outline

This is an outline of the lecture that is intended to provide you with the images and code. It is not a tutorial. Students are expected to pay attention and fill in the details.

- **Dynamic memory** space is created *upon demand* sometime after program begins execution -- NOT when data is defined.
Dynamic Memory is used most often when:

- **How much** data to be stored unknown (dynamic memory can grow or shrink as needed)
  - E.g., list of names where count unknown

- When **unknown amounts** of stored data related in complex ways

  example: Alphabetic list of employees, their children, pets, education. Each attribute could have \{0, 1, n\} values.
Dynamic Memory - General

- **Dynamic memory** is used most often when:

  - When changing data is ordered, both:

    - In single sequence

```
Dynamic Memory - General

- Dynamic memory is used most often when:
  - When changing data is ordered, both:
    - In single sequence
```

```
(a)  20 ➔ 45 ➔ 51 ➔ 84

(b)  20 ➔ 45 ➔ 51 ➔ 84
    Old value
    Inserted item
    60

(c)  20 ➔ 45 ➔ 51 ➔ 60 ➔ 84
    Deleted item
```
Dynamic Memory is used most often when:

- When changing data is ordered, both:
  - In single sequence
  - In multiple sequences

Data table is shown as a collection of fixed length records. However, it also may be a collection of pointer lists.
Dynamic Memory - General

Points of interest:

- Pointers contain location (i.e., memory address) of an object stored in memory.

- By using pointer, can locate object; subsequently determine content.

- Following address in pointer object, program gets to another object. This is an indirect reference.
• **Pointers** may reference other **dynamic objects** (usually) or **statically defined space**. If referencing static space, need to use "&"; e.g.,

```c
huskyPointer = &huskies;
```

![Diagram: HuskyPointer and Huskies]

• **Static** objects can always be **assigned values directly**; e.g.,

```c
huskies = 5;
```

• If **referred by known pointer**, static objects can always be **indirectly assigned values**; e.g.,

```c
*huskyPointer = 5;
```
• Memory space for **static space** (e.g., `int`) **allocated before** program object (C++ function) execution; i.e., before object (function) execution takes place, it is *statically* allocated.

• Memory space for **dynamic space allocated during execution**

  ◦ Called **dynamic allocation**

  ◦ Done by using `new` operator; e.g.,

    ```
    huskyPointer = new int;
    ```

    ![ Husky Pointer Diagram ]

    • Dynamically allocated space can **only** be referenced indirectly.
• Dynamically allocated space can only be referenced indirectly.

• Two pointers reference same location (using assignment operator); e.g.,

```c
int* huskyPointer;
huskyPointer = new int;
int* cuteDoggy;
cuteDoggy = huskyPointer;
```
Dynamic Memory - General

• When **pointers first allocated**, pointer value **undefined**

• *Some* compilers set value to: “**NULL**”.

• **NULL** supplied by *some* compilers to indicate nothing referenced.

• **Note:** Good C++ programming style does not assume *any* initial values when *any* memory space is allocated, either dynamically or statically.

• Allocated **dynamic space** can be **destroyed** by **delete**; e.g.,

  ```
  delete huskyPointer;
  ```