Procedure Programming

Characters

Instructor: Jeeho Ryoo

Announcements

- Quiz1 today
- Lab clarification
    - Beginning of the lab
    - Not debugging time
    - Sharing my input
    - Same rule applies to all future labs
    - One chance
- Assignment 1 out
    - How I will be grading
    - Questions in Discord
    - Early submission gets +1 bonus
- Accessibility request
    - Please do it ASAP

Definition — *The Heap*

A region of memory provided by most operating systems for allocating storage *not* in *Last in, First out* discipline

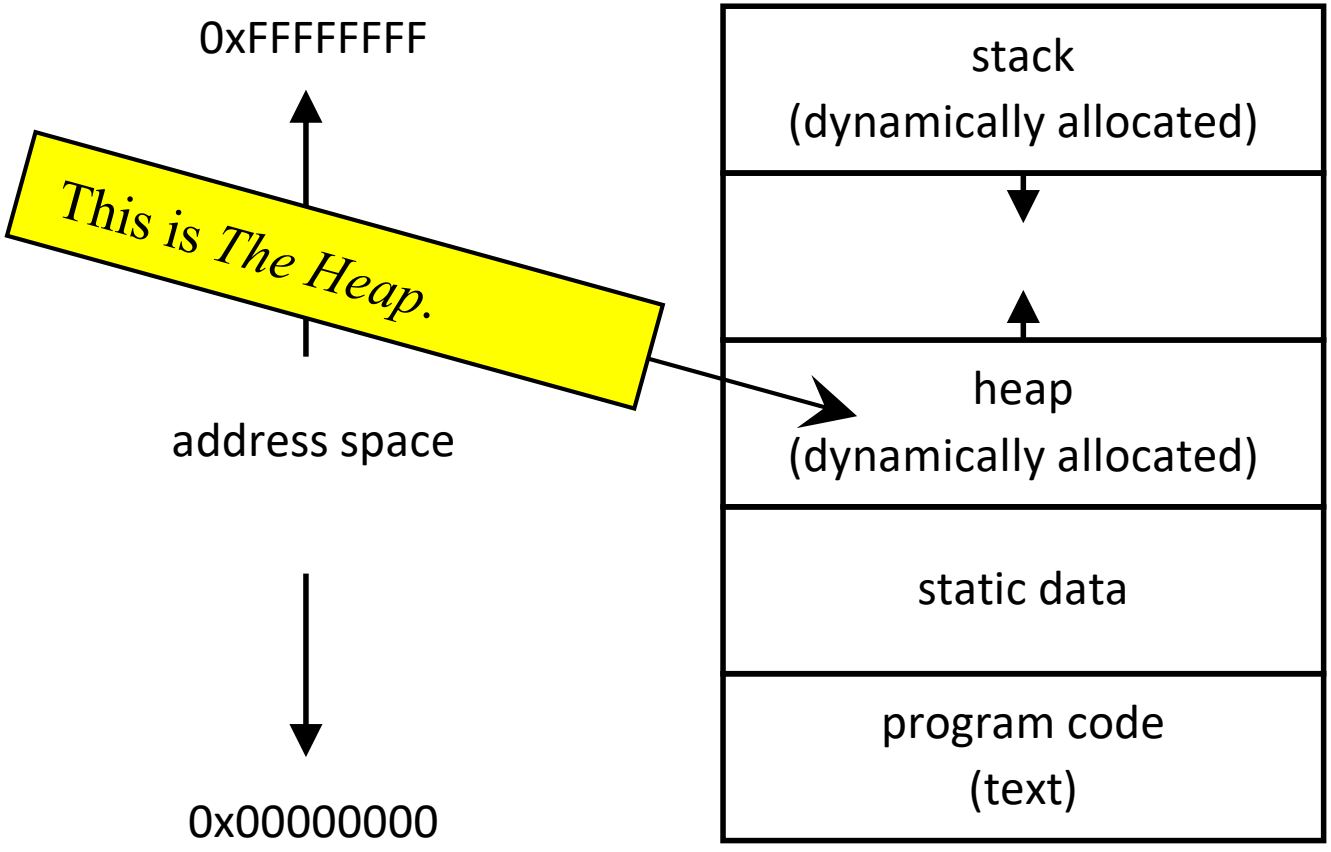
i.e., not a stack

Must be explicitly allocated *and* released
May be accessed *only* with pointers

Remember, an array is equivalent to a pointer

Many hazards to the *C* programmer

Static Data Allocation



Allocating Memory in The Heap

See **`<stdlib.h>`**

```
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

`malloc()` — allocates **`size`** bytes of memory from the heap and returns a pointer to it.

NULL pointer if allocation fails for any reason

`free()` — returns the chunk of memory pointed to by **`ptr`**

Must have been allocated by **`malloc`** or **`calloc`**

Allocating Memory in The Heap

See **`<stdlib.h>`**

```
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

`malloc()` — allocates **`size`** bytes of memory from the heap and returns a pointer to it.

- NULL pointer if allocation fails for any reason

`free()` — returns the chunk of memory pointed to by **`ptr`**

- *Must* have been allocated by **`malloc`** or **`calloc`**

`free()` knows *size* of chunk allocated by **`malloc()`** or **`calloc()`**

Notes

`calloc()` is just a variant of **`malloc()`**

`malloc()` is analogous to **`new`** in *C++* and Java

- **`new`** in *C++* actually calls **`malloc()`**

`free()` is analogous to **`delete`** in *C++*

- **`delete`** in *C++* actually calls **`free()`**
- Java does not have **`delete`** — uses *garbage collection* to recover memory no longer in use

Typical usage of **malloc()** and **free()**

```
char *getTextFromSomewhere(…);

int main(){
      char * txt;
      …;
      txt = getTextFromSomewhere(…);
      …;
      printf("The text is %s.", txt);
      free(txt);
}
```

Typical usage of **malloc()** and **free()**

getTextFromSomewhere() creates a new string using **malloc()**

```
char * getTextFromSomewhe
    char *t;
    ...
    t = malloc(stringLength);
    ...
    return t;
}

int main(){
    char * txt;
    …;
    txt = getTextFromSomewhere(…);
    …;
    printf("The text is %s.", txt);
    free(txt);
}
```

Typical usage of **`malloc()`** and **`free()`**

```
char * getTextFromSomewhere(…){
     char *t;
     ...
     t = malloc(stringLength);
     ...
     return t;
}

int main(){
     char * txt;
     …;
     txt = getTextFromSomewhere(…);
     …;
     printf("The text is %s.", txt);
     free(txt);
}
```

Pointer to text is assigned to **`txt`** in calling function

Usage of **`malloc()`** and **`free()`**

```
char * getTextFromSomewhere(...){
     char *t;
     ...
     t = malloc(stringLength);
     ...
     return t;
}

int main(){
     char * txt;
     ...;
     txt = getTextFr
     ...;
     printf("The tex          xt);
     free(txt);
}
```

`main()` must remember to free the storage pointed to by **`txt`**

Definition – *Memory Leak*

The steady loss of available memory due to forgetting to **`free()`** everything that was **`malloc`**'ed.

- Bug-a-boo of most large *C*

If you "forget" the value of a pointer to a piece of **`malloc`**'ed memory, there is no way to find it again!

- Killing the program frees *all* memory!

String Manipulation in *C*

Almost all *C* programs that manipulate text do so with **`malloc`**'ed and **`free`**'d memory

No limit on size of string in *C*

Need to be aware of sizes of character arrays!

Need to remember to free storage when it is no longer needed

- *Before* forgetting pointer to that storage!

Input-Output Functions

`printf(const char *format, ...)`

- Format string may contain **`%s`** – inserts a string argument (i.e., **`char *`**) up to trailing **`'\0'`**

`scanf(const char *format, ...)`

- Format string may contain **`%s`** – scans a string into argument (i.e., **`char *`**) up to next "white space"
- Adds **`'\0'`**

Related functions

- **`fprintf()`**, **`fscanf()`** – to/from a file
- **`sprintf()`**, **`sscanf()`** – to/from a string

Example Hazard

```
char word[20];
…;
scanf("%s", word);
```

`scanf` will continue to scan characters from input until a space, tab, new-line, or **`EOF`** is detected

- An unbounded amount of input
- May overflow allocated character array
- Probable corruption of data!
- **`scanf`** adds trailing **`'\0'`**

Solution:

```
scanf("%19s", word);
```