BE-Discussion 12, 2020-11-20

Announcements

- Final project due last day of class ... have plenty of time, but don't procrastinate! (12/9)
- Exam #3 is just on C material and two weeks from today
- Next week we start on a new data structure:
  LINKED LISTS: structures are dynamically allocated and linked together w/pointers
- Last Quiz today!
  - released at 4^40 pm Boston time, due 5 pm. 15 min to complete, 5 min to upload
  - alternate time zone quiz Saturday at 7am Boston time
- When downloading something from Gradescope, make sure you always compare what you downloaded to the original! Sometimes formatting can be off.
- Have a nice (and safe!) Thanksgiving !!!

Quiz #7 Review

- questions?

Review of Material

- Pointers
- Call-by-reference
  → use call-by-reference only when a function is calculating or initializing more than one value; if it is doing only one, use return instead
- Note that arrays (including character arrays) cannot be returned from a function

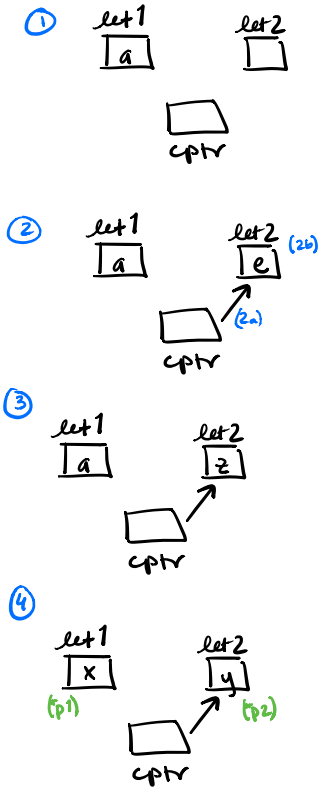
```c
#include <stdio.h>

void dostuff(char *, char *);

int main( )
{
    char let1 = 'a',
         let2,
         *cptr;

    cptr = &let2;
    *cptr = 'e';
    printf("let1 is %c and let2 is %c\n", let1, let2);

    let2 = 'z';
    printf("let1 is %c and let2 is %c\n", let1, let2);
    printf("*cptr is %c\n", *cptr);

    dostuff(&let1, &let2);
    printf("let1 is %c and let2 is %c\n", let1, let2);

    return 0;
}

void dostuff(char *p1, char *p2)
{
    *p1 = 'x';
    *p2 = 'y';
}
```

step ①: initialization

(2a) (2b)

you could also pass: (char * p1, char * p2)

④ a



② let1 is a and let2 is e

③ let1 is a and let2 is z
*cptr is z

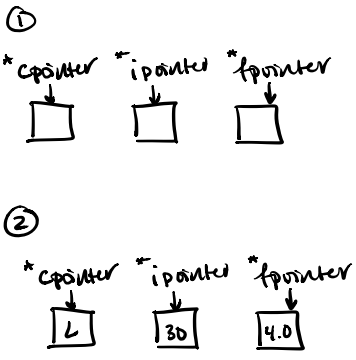
④ let1 is x and let2 is y

```c
#include <stdio.h>
#include <stdlib.h>
// Necessary to include these header files for dynamic memory allocation

void makespace(char **, int **, float **);
// The function prototype must be before int main()

int main()
{
    char *charptr;
    int *intptr;
    float *floatptr;
    // Initialize the pointer variables needed to pass in the function makespace()

    printf("\nThis program uses a function to initialize pointer variables using\n");
    printf("Dynamic Memory Allocation (DMA).\n");
    printf("The values in these allocated places are arbitrarily chosen in the function.\n");

    makespace(&charptr, &intptr, &floatptr);
    // Notice the '&' before the pointer variables because we are passing the
    // addresses of these pointer variables to the function makespace()

    printf("\nHere's what each pointer points to:\n");
    printf("charptr points to %c\n", *charptr); // The '*' before charptr dereferences it
    printf("intptr points to %d\n", *intptr); // The '*' before intptr dereferences it
    printf("floatptr points to %.2f\n", *floatptr); // Use '*' to dereference

    printf("\n***\nThis is the end of the program.\n***\n\n");

    free(charptr);
    free(intptr);
    free(floatptr);
    // Important to free pointers at the end!

    return 0;
}

void makespace(char **cpointer, int **ipointer, float **fpointer) // The function header with body beneath
{
    *cpointer = (char *) malloc(sizeof(char));
    *ipointer = (int *) malloc(sizeof(int));
    *fpointer = (float *) malloc(sizeof(float));
    // These create space for data and values!
    // Notice the derefencing before cpointer, ipointer, and fpointer

    **cpointer = 'L';
    **ipointer = 30;
    **fpointer = 4.0;
    // This inserts values/data in these spaces
    // Notice the double dereferencing to put values in!
}
```

skip 4 : initialization

void function : call-by-ref.

① ② ③ ④



③
charptr points to L
intptr points to 30
floatptr points to 4.00

④ FREEING POINTERS!

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

void initthem(int **, char **);

int main()
{
    int *intptr;

    char *chptr;

    initthem(&intptr, &chptr);

    printf("*intptr is %d and *chptr is %c\n", *intptr, *chptr);

    free(intptr);
    free(chptr);

    return 0;
}

void initthem(int **ipoint, char **chpoint)
{
    *ipoint = (int *) malloc(sizeof(int));
    **ipoint = 33;
    *chpoint = (char *) malloc(sizeof(int));
    **chpoint = '!';
}
```

step ① : variable/pointer initialization

← step ②

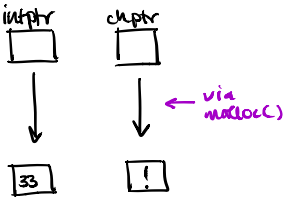
← step ④

step ⑤

step ③

① intptr chptr

② & ③



④ *intptr is 33 and *chptr is !

⑤ free pointers!

OTHER SUPPLEMENTAL NOTES

Intro to Dynamic Memory Allocation & Linked Lists in C

```
(int *) malloc(sizeof(int))
```

type cast to change to pointer of desired type

function: allocates the memory (and returns address of memory allocated as pointer to type void)

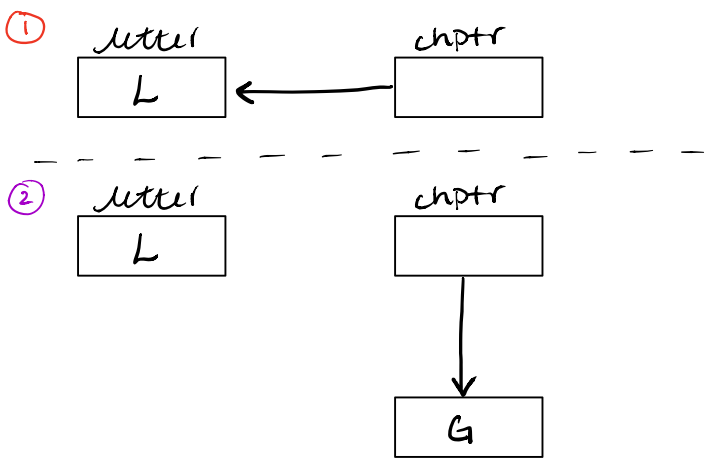
sizeof(): function, in this case allocates enough bytes to store an integer

this will allocate enough space for an integer and casts the address of the location to an int pointer

```
intptr = (int *) malloc(sizeof(int));
```


```
*intptr = 30;
```

```c
// Example 1


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


int main()
{
    char letter = 'L',
         *chptr;

    chptr = &letter;
    printf("letter is %c\n", letter);
    printf("*chptr is %c\n", *chptr);

    chptr = (char *) malloc(sizeof(char));
    *chptr = 'G';
    printf("letter is %c\n", letter);
    printf("*chptr is %c\n", *chptr);

    free(chptr);
    return 0;
}
```



output:

letter is L
*chptr is L
letter is L
*chptr is G