Dynamic Memory Allocation

Sections 17.1 to 17.4

Objectives

- Learn how to
  - Write programs that allocate memory for data structures at run time.
  - Set the size of data structures at run time.

Dynamic Memory Allocation

- In everything we have done so far, our variables have been declared at compile time.

- Today we will see how to allocate memory ***dynamically***
  - Allocate **at run time**.
  - Determine size at run time.

Dynamic Memory Allocation

- Why allocate memory dynamically?

- In many cases, we can’t predict in advance what the program will need.
  - May depend on data not known at compile time.
  - May vary dramatically from one run to another.

malloc

- To allocate memory at run time, call the library function **malloc()**.
  - *memory allocation*.

- Function prototype:

void* malloc (size_t sz);

Number of bytes in the block be to allocated (positive integer)

A generic pointer

Can be used as *any* pointer type.

Optionally typecast as desired pointer type.

malloc

- malloc returns a pointer to the first byte of the memory that it allocated.
  - Store the returned value in a pointer so that you can use it.
- malloc can fail!
  - Returns NULL if it cannot allocate the requested amount of memory.
  - Always check for NULL being returned.
  - Normally have to abort the program if this happens.

malloc

- The function declaration for malloc is in **stdlib.h**.

- You will need

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

Array

- Recall that we used a pointer `p` to traverse an array `A` by first setting `p` = `A`
- This sets `p` to the address of the first byte of `A`.
- Similarly, when we call `malloc`, it returns the address of the first byte of the allocated block
- How does the compiler compute the address of `A[j]` for `A` an array of some type `t` and `j` a nonnegative integer?
- ```
  address of A[j]
  = (address of first byte of A) + j*sizeof(t)
  = A + j*sizeof(t)
  ```
- For this reason, after we do

  ```
  int * p = malloc(10*sizeof(int));
  ```

  we can treat p as representing an array of `int`s of length 10.
- This is what is meant by a ***dynamic array***.
- For any type t, we create a length-L dynamic array of type t by:

  ```
  t *p = malloc(L*sizeof(t));
  ```
  ```

Example

1. Read a sentence from the keyboard
   - using a large input buffer (array of char).

3. Allocate memory to hold the sentence
   - just large enough to hold what was entered.

5. Copy the sentence from the buffer into the dynamically allocated memory block.

Example

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

int main()
{
    char input_buffer[1000];
    int length = 0;
    char* sentence = NULL;

    printf ("Please type a sentence:\n");
    fgets(input_buffer, 1000, stdin);
     // Echo input
    printf ("You typed: \n%s\n\n", input_buffer);
```

Example

why +1 ?

```
    length = strlen(input_buffer);
    printf ("Allocating %d bytes to hold the sentence\n\n", length+1);

    sentence = malloc(length+1);
    assert (sentence != NULL);

    strcpy (sentence, input_buffer);
    printf ("The sentence in dynamically allocated memory:\n");
    printf ("%s\n", sentence);
    getchar();  /* Not needed       */
    getchar();  /* when using CLUE */
    return 0;
}
```

Program Running

```
c:\Documents and Settings\Rollins\Desktop\malloc.c\Debug\malloc.c.exe

Please type a sentence:
The quick brown fox jumped over the lazy dog's back.
You typed:
The quick brown fox jumped over the lazy dog's back.


Allocating 54 bytes to hold the sentence

The sentence in dynamically allocated memory:
The quick brown fox jumped over the lazy dog's back.
```

Pointer

- Recall that if `A` is an array of characters, then A is a constant of type `char *`

- Similarly, if `A` were an array of integers it would be a constant of type `int *`

- In fact, for any type t, if `A` is an array of objects of type then it is a constant of type `t *`

- Now, a two-dimensional array `C` of characters is an array of character arrays (the rows of `C`)

- Since a character array has type `char *`, `C` is an array of object of type `char *`.

- Thus, `C` is a constant of type `char * *`.

- If we execute `char **p = C`, then `*p` points to the first row of `C`, `*(p+1)` points to the second row of `C`, etc.

A Bigger Example

- Initial version of program.
  - No dynamic allocation.
  - Max size arrays to hold strings entered by the user.
  - Sorts pointers to the strings.

- We will replace fixed allocation with dynamic allocation.

string_sort.c

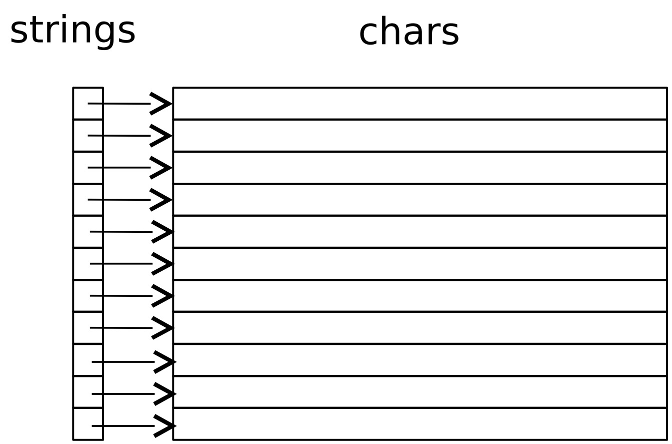
2-dimensional array of characters

```
int get_strings( char** strings, int max_nr_strings, int max_string_len);
void swap(char** s1, char** s2);
void sort(char* s[], int length);
void output_strings(char* s[], int length);

int main()
{
    char chars[10][1000];
    char* strings[10];
    int i;
    int count = 0;

    for (i = 0; i < 10; i++)
    {
        strings[i] = &chars[i][0];      // or: strings[i] = chars[i];
}

    printf ("This program accepts up to 10 strings from the keyboard\n");
    printf ("sorts them, and outputs the sorted strings.\n");
    count = get_strings(strings, 10, 1000);
```



string_sort.c

```
    printf ("Strings prior to sort:\n");
    output_strings(strings, count);

    sort(strings, count);

    printf ("Strings after sort:\n");
    output_strings(strings, count);

    getchar();
    getchar();
    return 0;
}
```

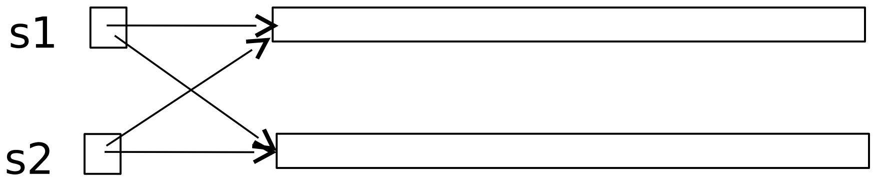
```c
/* Get up to N strings from the keyboard.
 * Return number of strings entered.       */
int get_strings(char** strings, int max_nr_strings, int max_string_len)
{
    int i = 0;
    printf ("Enter up to %d strings to be sorted\n", max_nr_strings);
    printf ("Enter a zero length string to terminate input\n\n");

    for (i = 0; i < max_nr_strings; i++)
    {
        int length = 0;
        printf ("%d: ", i);
        fgets(strings[i], max_string_len, stdin );
        length = strlen(strings[i]);
        strings[i][length - 1] = 0;        // Delete newline character
        if (length < 2)
        {
            break;
        }
    }
    // i is the number of elements filled

    printf ("%d strings were entered\n\n", i);
    return i;
}
```

Top of File

```
void swap(char** s1, char** s2)
// char * parameters passed by reference ("by address")
{
    char* temp = *s1;
    *s1 = *s2;
    *s2 = temp;
}
```



The Sort Function

```
void sort(char* s[], int length)
{
    int i;
    int swap_done = 0;

    do
    {
        swap_done = 0;
        for (i = 1; i < length; i++)
        {
            if (strcmp(s[i-1], s[i]) > 0)
            {
                swap(&s[i-1], &s[i]);
                swap_done = 1;
            }
        }
    } while (swap_done);
}
```

output_strings()

```
void output_strings(char* s[], int length)
{
    int i;
    for (i = 0; i < length; i++)
    {
        printf ("%s\n", s[i]);
    }
    printf ("\n");
}
```

Program Running

```
c:\documents and settings\rollins\my documents\visual studio 2005\projects\program_design_2009_spring\string_sort\debug\String_...
This program accepts up to 10 strings from the keyboard
sorts them, and outputs the sorted strings.
Enter a zero length string to terminate input

0: Now is the time
1: The quick brown fox
2: jumped over the lazy dog's back
3: 123456 789
4: abcdef
5:
5 strings were entered

Strings prior to sort:
Now is the time
The quick brown fox
jumped over the lazy dog's back
123456 789
abcdef

Strings after sort:
123456 789
Now is the time
The quick brown fox
abcdef
jumped over the lazy dog's back
```

Using malloc

- Replace the declared arrays of char with dynamically allocated arrays.
  - Keep the array of pointers to the arrays of char.
  - We will fill in the pointers as the arrays are allocated.

The Text Sort with Dynamically Allocated Memory

```
int main()
{
    /* char chars[10][1000]; */

    char* strings[10];
/*    int i;   */
    int count = 0;

/*    for (i = 0; i < 10; i++)          */
/*    {                                 */
/*        strings[i] = &chars[i][0];    */
/*    }                                 */

    printf ("This program accepts up to 10 strings from the keyboard\n");
    printf ("sorts them, and outputs the sorted strings.\n");

    count = get_strings(strings, 10);

/* Remainder same as before */
```

```
int get_strings(char** strings, int max_nr_strings,
                int max_string_len) // strings is an array of char *,i.e., strings
{
    int i = 0;
    char input_area[1001];  // input buffer
    assert (max_string_len <= 1000);
    printf ("Enter up to %d strings\n", max_nr_strings);
    printf ("Enter an empty string to terminate input.\n");
    for (i = 0; i < max_nr_strings; i++)
    {
        int length = 0;
        printf ("%d: ", i);
        fgets(input_area, max_string_len, stdin);
        length = (int) strlen(input_area);
        input_area[length-1] = 0;  // Delete newline
        length--;
        if (length < 1)
        {
            break;
        }
        strings[i] = malloc(length+1);
        assert(strings[i] != NULL);
        strcpy(strings[i], input_area);
    }
    // i is the number of elements filled
    printf ("%d strings were entered\n\n", i);
    return i;
}
```

The Text Sort with Dynamically Allocated Memory

- Add at the top

```
...
#include <stdlib.h>        // for malloc
#include <assert.h>
```

The Text Sort with Dynamically Allocated Memory

- Everything else is unchanged.
- Behavior is unchanged.

Revised Program Running

c:\Documents and Settings\Rollins\Desktop\malloc_demo\Debug\malloc_demo.exe

```
This program accepts up to 10 strings from the keyboard
sorts them, and outputs the sorted strings.
Enter up to 10 strings
Enter an empty string to terminate input.
0: Now is the time
1: The quick brown fox
2: jumped over the lazy dog's back
3: 123456 789
4: abcdef
5:
Strings prior to sort:
Now is the time
The quick brown fox
jumped over the lazy dog's back
123456 789
abcdef

Strings after sort:
123456 789
Now is the time
The quick brown fox
abcdef
jumped over the lazy dog's back
```

Questions?

- Any questions about the program that inputs and sorts strings and uses dynamic arrays throughout?

Things to Notice

- Always check the pointer returned by malloc to be sure the malloc was successful.
- We can make the dynamically allocated memory block any type that we need.
  - Just copy the void* pointer returned by malloc() into a pointer of the desired type.
  - (Some older compilers may require a typecast.)

C Standard Library Functions for Dynamic Memory Allocation

- void* malloc (size_t sz);
  - Returns pointer to uninitialize block of memory of specified size (in bytes)
- void* calloc (size_t n, size_t sz);
  - Allocate and clear.
  - Returns pointer to an array of n entries, each of size sz bytes, cleared to binary 0's.
- void free (void* pt);
  - Deallocate block at address pt.
  - Must be a block allocated with malloc or calloc
  - We will discuss this next slides.

C Standard Library Functions for Dynamic Memory Allocation

- void* realloc (void* pt, size_t sz);
  - Reallocate block at address pt to a new size.
  - Must be a block allocated by malloc or calloc.
  - sz can be either larger or smaller than original size.
  - Moves the contents if necessary.

Deallocating Memory

- Use **free()** to recycle memory blocks
  - Important if program uses dynamically allocated blocks temporarily.
  - Also a major source of errors!

- Caution
- You *must*  use a pointer after the block to which it points has been freed.
  - This includes calling free() a second time for the same block.

Deallocating Memory

- As a precaution, set the pointer to NULL following call to free().

- Same principle as initializing a pointer to NULL.
  - Get an immediate runtime error if the pointer is dereferenced.
  - The alternative can be much worse!

Deallocating Memory

- To be safe, avoid having multiple pointers to the same block of memory
  if at all possible.

- ***Exception***: in a loop where the second pointer is changed repeatedly and is finally changed to NULL

Deallocating Memory

- You have to be careful about pointers stored in data structures
- If you free the block to which a pointer points, the pointer becomes a *dangling reference*.
  - Toxic waste in a C program!
- If you have a pointer in a data structure and deallocate the block that it points to either
  - deallocate the block containing the pointer; or
  - reset the pointer.
  - Sometimes it should point to a new location.
  - Otherwise set it to NULL.

Deallocating Memory

- Avoid deallocating a block containing a pointer while the block the pointer references is still allocated.
- This can result in a *memory leak*.
  - Blocks of allocated memory that you can’t get to.
  - Causes the program to use more memory than necessary.
  - Antisocial on a multiuser system!
  - Will eventually crash a long running program.

Deallocating Memory

In long running programs --

- If you allocate memory for temporary use, you should be sure each malloc() has a matching free().

- If blocks have pointers to other blocks, work back from the end of the chain of pointers toward the start.

Assignment

- Read Chapter 17
  - Sections 17.1 – 17.4
- Do the examples from this class for yourself
  - if you did not do them in class.
- Study the examples carefully and critically.
  - Ask *why?* for each major step
  - Be sure you understand every line!
- Do problems 1, 3, and 4, page 453

- End of Presentation