CSC 215
Memory Management

Dr. Achraf El Allali

Static Allocation

- Allocation of memory at compile-time before the associated program is executed
- Let’s say we need a list of 1000 names:
  - We can create an array statically
    - char names[1000][20]
    - allocates 20000 bytes at compile time
    - wastes space
    - restricts the size of the names

Dynamic allocation of memory

- Allocate memory during runtime as needed
- #include <stdlib.h>
- Use **sizeof** number to return the number of bytes of a data type.
- Use **malloc/calloc/realloc** to find a specified amount of free memory and returns a void pointer to it.

Example

- char * str = (char *) malloc(3 * sizeof(char));
  strcpy(str, “hi”);

- str = (char *) realloc(str , 6 * sizeof(char));
  strcpy(str, “hello”);

Dynamic Deallocation

- #include <stdlib.h>
- **free** releases the memory pointed to by a pointer variable back to the OS:

```
char * str = (char *) malloc( 3 * sizeof(char) );
strcpy(str, “hi”);
...  use str ...
 free(str);
```

Free

- Can only be used on pointers that are dynamically allocated
- It is an error to free:
  - A NULL pointer
  - A pointer that has already been freed
  - Any memory address that has not been directly returned by a dynamic memory allocation routine

Dynamically Allocated Arrays

- Allows the user to avoid declaring array size at declaration.
- Use malloc to allocate memory for array when needed:

```
int *a;
a= (int *) malloc( sizeof( int) * 10 );
a[0]=1;
```

Example

```
int size;
char *s;
printf(“How many characters?\n”);
scanf(“%d”, &size);
s = (char *) malloc(size+1);
printf(“type string\n”);
gets(s);
```

Calloc

```
void* calloc (size_t num, size_t size);
```

- Alternative to malloc
- Originally written to allocate arrays
- Allocate and zero-initialize array
- Takes two parameters:
  - Number of elements to allocates
  - Size of an element

Example

```
#include <stdlib.h>
#include <stdio.h>
int main(){
        int *ap, i;
        ap =(int*) calloc(10, sizeof(int));
        for(i=0;i<10;i++)
                printf(“%d\n”,*(ap+i));
        return 0;
}
```

Realloc

```
void* realloc (void* ptr, size_t size);
```

- Reallocate memory block
- Changes the size of the memory block pointed to by ptr
- May move the memory block to a new location
- If the new size is larger, the value of the newly allocated portion is indeterminate
- Behaves like malloc if ptr is a null pointer

Example

```
#include <stdio.h>      /* printf, scanf, puts */
#include <stdlib.h>     /* realloc, free, exit, NULL */

int main ()
{
  int input,n;
  int count = 0;
  int* numbers = NULL;
  int* more_numbers = NULL;

  do {
     printf ("Enter an integer value (0 to end): ");
     scanf ("%d", &input);
     count++;

     more_numbers = (int*) realloc (numbers, count *
sizeof(int));
     if (more_numbers!=NULL) {
        numbers=more_numbers;
        numbers[count-1]=input;
     }
     else {
        free (numbers);
        puts ("Error (re)allocating memory");
        exit (1);
     }
  } while (input!=0);

  printf ("Numbers entered: ");
  for (n=0;n<count;n++) printf ("%d ",numbers[n]);
  free (numbers);

  return 0;
}
```