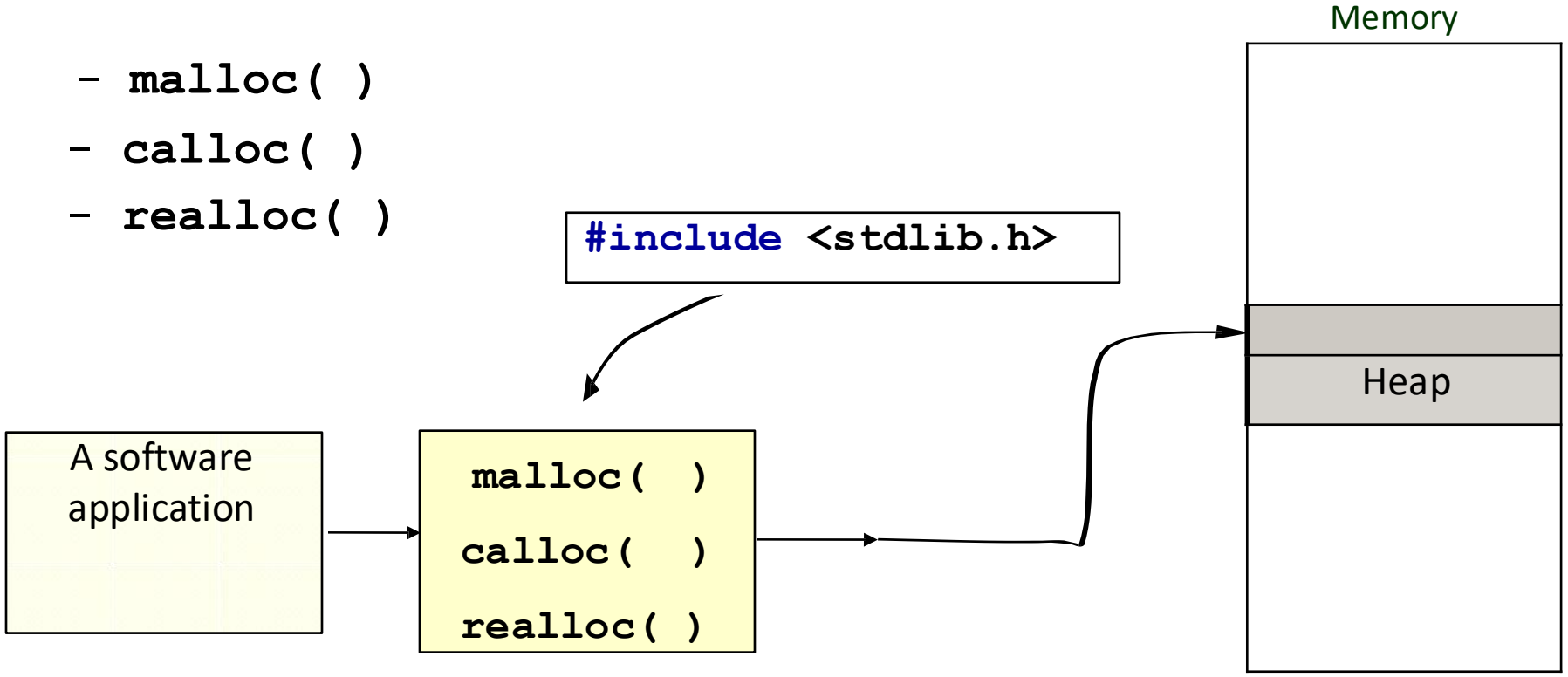
# Dynamic Memory Allocation

**Note:** In the assessment, you expected to know the function malloc, calloc and memset only

# Memory Allocation Functions

- There are three functions specified by ANSI C for memory allocation:
  - `malloc( )`
  - `calloc( )`
  - `realloc( )`




UNIVERSITY OF WOLLONGONG IN DUBAI

# malloc( )

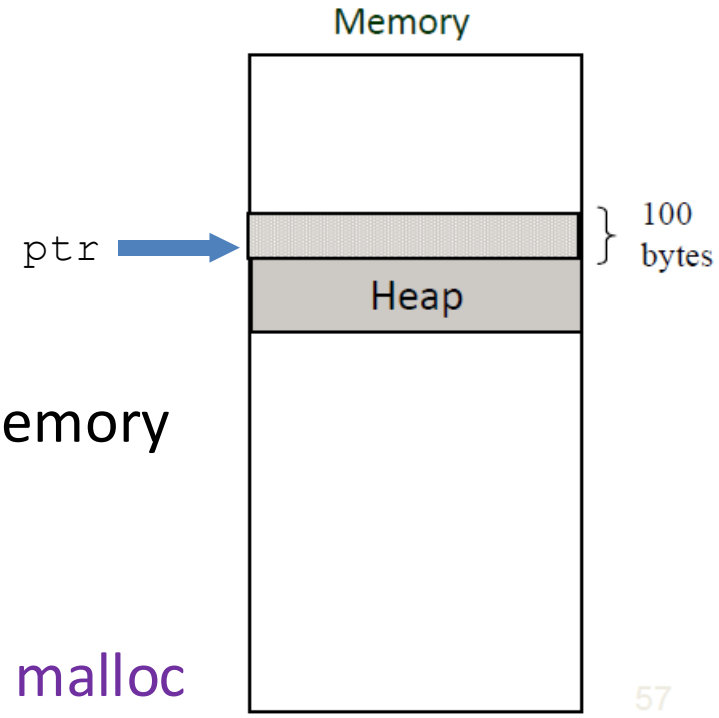
- Allocates a specified number of bytes of memory.

```
void* malloc( int numOfBytes );
```

*Example:*

```
char *ptr;
ptr = malloc(100);
```

a memory block of 100 bytes




- malloc does not initialize allocated memory space.
- heap
  region of memory in which function malloc dynamically allocates blocks of storage.



UNIVERSITY OF WOLLONGONG IN DUBAI

# Caution

- **`malloc`** returns the generic type **`void*`** pointer

```
void* malloc( int numOfBytes );
```

- Why `void*` ?

  Memory allocated by **`malloc()`** can be used to store data of any type

*Example:*

```
char   *ptrC;
float *ptrF;

ptrC = malloc(100); /* memory to store char type data */
ptrF = malloc(100); /* memory to store float type data */
```

- The generic **`void*`** pointer is implicitly converted to a pointer of the type matching the *lvalue* when assigned

UNIVERSITY OF WOLLONGONG IN DUBAI

# Type Casting

- You do not have to explicitly cast the void pointer returned by `malloc` when you assign it to a pointer of a different type.

  *Example:*

```
float *ptrF;

ptrF = malloc(100); /* usually OK */
```

- Some compilers may require explicit casting

```
ptrF = (float*) malloc(100);      /* explicit casting */
```

- Explicit casting may help you to avoid bugs in your code

```
ptr1 = (float*) malloc(100);
```

You'll be warned if ptr1 has already been converted from float* to another type

UNIVERSITY OF WOLLONGONG IN DUBAI

# `malloc` failure

- Do not assume that **`malloc`** will always succeed.
- When memory allocation fails, **`malloc`** returns the NULL pointer.

*Example:*

```
char *ptr;

if( (ptr = (char*)malloc(100)) == NULL )
{
    printf(" Memory allocation failure");
    return (-1);
}
```

- The error recovery is application specific, but very often the program has to be terminated.

UNIVERSITY OF WOLLONGONG IN DUBAI

# malloc( )

- To allocate memory for a specified number of elements of any type use the **sizeof()** operator.

*Example:* Allocate memory to store 100 integers

```
int *ptr;

ptr = (int*)malloc( 100*sizeof(int) );
if(ptr==NULL)
{
   printf(" Memory allocation                failure");
   return (-1);
}
```

- Do not specify the size of a data type explicitly as it may be different on other platforms

```
ptr = malloc( 100*4 ); /* bad style */
```

UNIVERSITY OF WOLLONGONG IN DUBAI

# Caution

- Do not modify the pointer returned by **`malloc()`**

*Example:*

```
float *ptr;

ptr = malloc( 100*sizeof(float) );
if(ptr==NULL)
   printf( " Memory allocation failure");


ptr += 5;
```

may lead to memory management problems

- The pointer must point to the beginning of the allocated memory space to avoid confusion of the system memory manager.

UNIVERSITY OF WOLLONGONG IN DUBAI

# calloc( )

- Allocates memory for a specified number of elements and initialises them to `0` .

```
void* calloc( int numOfElem, int sizeOfElem  );
```

*Example*:

```
float *ptr;

ptr = (float*)calloc( 100, sizeof(float) );
if( ptr == NULL )
  { error recovery };
```

- **`calloc()`** is more computationally expensive and as a result, it is slightly slower than **`malloc()`**

UNIVERSITY OF WOLLONGONG IN DUBAI

# realloc( )

- Changes the size of a previously allocated area

```
void* realloc( void *ptr, int newSize );
```

*Example*: Increase a dynamic array size from 100 to 2000 bytes

```
char *ptr;
if( (ptr = malloc(100)) == NULL )

 { error recovery }
 .   .   .   .
 .   .   .   .
 If (ptr = realloc(ptr, 2000)) == NULL )
{error recovery }
```

- The size of the originally allocated area can be increased or decreased.
- Existing array values are preserved.

UNIVERSITY OF WOLLONGONG IN DUBAI

# Caution

- **`realloc`** will not operate correctly if you have previously modified the original pointer.

*Example:*

```
char *ptr;

if( (ptr = (char*)malloc(50) ) == NULL )
  .   .   .   .   .   .
ptr += 4;
  .   .   .   .   .

if((ptr = (char*)realloc(ptr, 200)) == NULL )
```

does not point to the beginning of the allocated memory space

- Debugging may be difficult as errors are usually not reported.

UNIVERSITY OF WOLLONGONG IN DUBAI

# Caution

- **`realloc`** will fail if new memory requirements are more than currently available.

*Example:*

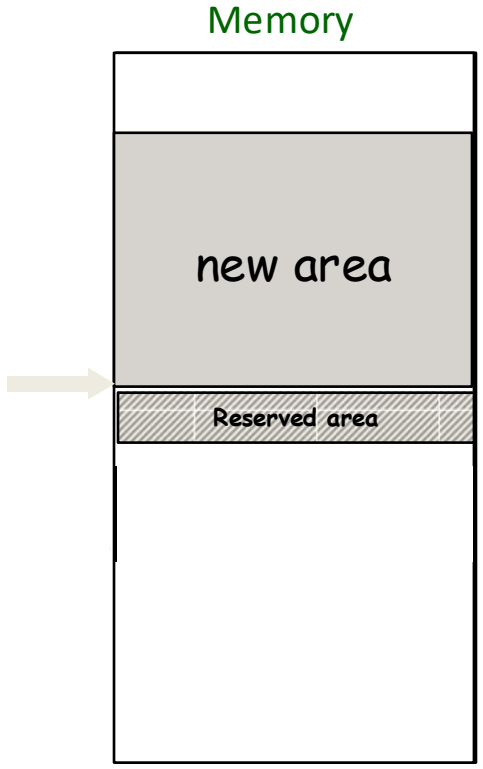
```
char *ptr;

if( (ptr = (char*)malloc(500)) == NULL )
  .   .   .   .   .   .
if( (ptr = (char*)realloc(ptr, 2000000)) == NULL )
```

- **Side effect**: The original pointer will be assigned with NULL value and the location of the original dynamic array with all data will be lost.

UNIVERSITY OF WOLLONGONG IN DUBAI

# realloc( )

- If there is enough room beyond the end of the originally allocated area, **realloc**:
  1. extends the original area
  2. returns the same pointer
- If there is not enough room available at the end of the original area, **realloc**:
  1. allocates a new area of the specified size
  2. copies existing data to the new area
  3. returns the pointer to the new area




UNIVERSITY OF WOLLONGONG IN DUBAI

# free( )

- The space for automatic variables declared in a function is freed when the function returns.

```
void sortByValue(float list[], int size )

{
int index;
Float maxVal, tempVal;
float tempArr[M_SIZE];
.   .
   return;
}
```

The memory space is freed automatically when the function executes return

- You need to free dynamically allocated memory explicitly so that it can be returned to the system

```
void free( void *ptr);
```

UNIVERSITY OF WOLLONGONG IN DUBAI

# free( )

- You can free a dynamically allocated area any time you think it is not needed any longer.

```
void sortByValue( float list[], int size )
{
    int index;
    float maxVal, tempVal;
    float *tempArr;
    .   .    .    .    .
    if( (tempArr = malloc(size*sizeof(float))) == NULL)
    .   .    .    .    .
    free(tempArr);
    .   .    .    .    .
    return;
}
```

you allocate memory when it is needed

you free memory when it is not needed

A very simple function call, but it may cause a lot of problems if not used properly

UNIVERSITY OF WOLLONGONG IN DUBAI

# Caution

- Use **`free()`** function only with pointers obtained from **`malloc()`, `calloc()`** and **`realloc()`** functions.

```
void sortByValue( float list[], int size )
{
   int i;
   float tempArr[M_SIZE];     /* automatic allocation */
   .  .   .   .   .
   for(i=0; i<M_SIZE; i++)
     tempArr[i] = 0.5*i;
   .  .   .   .   .
   free( tempArr );
   .  .   .   .   .
   return maxVal;
}
```

ERROR
`tempArr` was not allocated dynamically

UNIVERSITY OF WOLLONGONG IN DUBAI

# Caution

- As you call **`free()`** the pointers must point to the beginning of the memory area allocated by **`malloc()`, `calloc()`,** or **`realloc()`** functions.

```
void sortByValue( float list[], int size )
{
    int i;
    float *tempArr;
.   .    .    .    .
if( (tempArr=calloc(size, sizeof(float))) == NULL)
 .   .    .    .    .
 tempArr += 4;
 .   .    .    .
 free( tempArr );
 .   .    .    .    .
 return;
}
```

Do not modify pointers returned by **`malloc()`**

WRONG
the pointer has been modified

UNIVERSITY OF WOLLONGONG IN DUBAI

# Caution

1. Freeing a memory area does not change the value of the pointer. After free() it still points to the same memory area.
2. After free() the memory content remains the same until this memory area is allocated again.

   However, you must not make any attempt to use data in the freed area

```
float *tempArr, tempData;
.   .    .    .    .
if( (tempArr=calloc(num, sizeof(float))) == NULL)
.   .    .    .    .
free( tempArr );
.   .    .    .
tempData = *tempArr;
.   .    .    .    .
```

WRONG!!!
The area has already been freed

UNIVERSITY OF WOLLONGONG IN DUBAI

# What does `free()` mean?

- A function call **`free()`** tells the memory manager, that the previously allocated memory does not need to be reserved any longer.
- The memory manager marks this memory area as free to be used for other purposes.
- Any consecutive **`malloc()`** or **`calloc()`** call may grab that memory area.

UNIVERSITY OF WOLLONGONG IN DUBAI

# Quiz

- What is wrong with this code?

```
int j, sum=0;
char codeList[7] = {25, -12, 24, 17, 59, 43, 1};
char *listPtr;

listPtr = codeList;
for(j=0; j<7; j++)
  sum += listPtr[j];

free( listPtr );
```

`listPtr` points to the same location as `codeList` that was allocated statically

You can't free statically allocated memory

UNIVERSITY OF WOLLONGONG IN DUBAI

# Dynamic Arrays

- Dynamic Array allocation eliminates the most awkward aspect for conventional arrays – How big the array size should be

```
#define NUM_OF_STUDENTS   50
float examMarks[NUM_OF_STUDENTS];
```

What if the actual number is different?

- Dynamic Arrays can reserve space according to the actual requirements.

*Example:*

```
int numOfStudents;
float *examMarks;

printf("Enter the number of students :");
scanf("%d", &numOfStudents);
examMarks = calloc( numOfStudents, sizeof(float) );
```

UNIVERSITY OF WOLLONGONG IN DUBAI

# The use of Dynamic Arrays

*Example:* A function **`createFlArray`** that allocates memory for a specified size array and initialise elements to 0.

```
float *createFlArray(int size)
{
   static float *array;

   array = (float*)calloc( size, sizeof(float) );
   return array;
}


float *priceList;

priceList = createFlArray(  120 );
if( priceList == NULL ) . . . /* error recovery */


priceList[5] = 35.50;         /* access through indexing */
*(priceList + 5) = 16.30;     /* access through a pointer */
```

UNIVERSITY OF WOLLONGONG IN DUBAI

# Dynamic Arrays

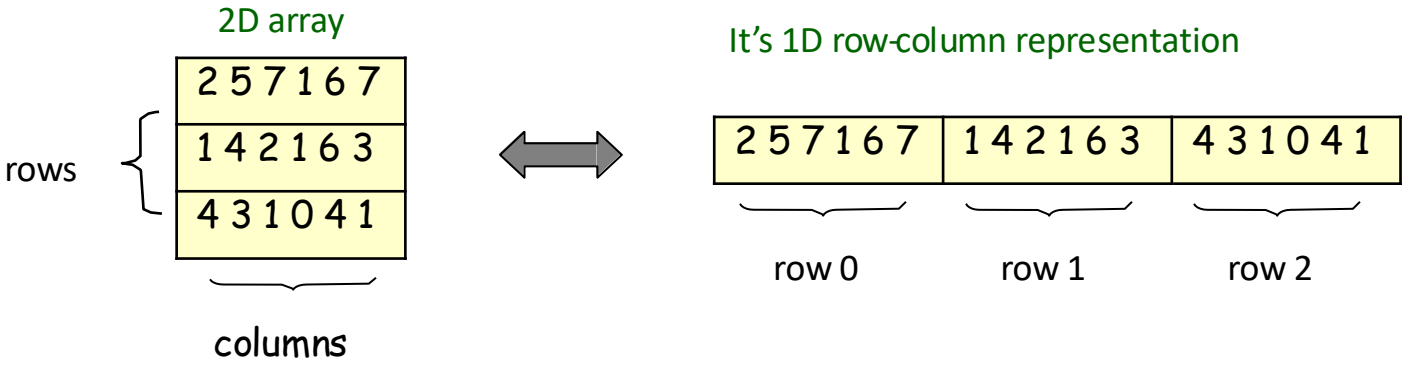
*Example:* A function that concatenates two strings

```
char firstName[] = "Peter";
char secondName[] = "Norton";
char *fullName;
/* function call */
fullName = addTwoStrings( firstName, secondName );
/* ----function definition---- */
char* addTwoStrings( char *firstStr, char *secondStr )
{
  static char *newStr;      /* a concatenated string */

  newStr=malloc( strlen(firstStr) + strlen(secondStr) + 2 );
  if(newStr == NULL ) return NULL; /*memory allocation error */
  strcpy( newStr, firstStr );
  strcat( newStr, " " );
  strcat( newStr, secondStr );
  return newStr;
  }
```

UNIVERSITY OF WOLLONGONG IN DUBAI

# Dynamic 2D Arrays

- You can allocate memory for 2D Arrays using their row-column decomposition into 1D Arrays




```
int *matrix;
matrix = calloc( nRows*nColumns, sizeof(int) );
 .   .   .   .   .
 *(matrix + rowIndex*nColumns + colIndex) = 35;
```

or → does the same as `matrix[rowIndex][colIndex] = 35;`

```
 matrix[ rowIndex*nColumns + colIndex ] = 35;
```

UNIVERSITY OF WOLLONGONG IN DUBAI

# Dynamic 2D Arrays

*Example:* A function **`createMatrix`** that allocates memory for a rows x columns number of elements of type int

```
int *createMatrix(    int rows,   int cols )
{
 static int *matrix;
      matrix  = (int*)calloc( rows*cols, sizeof(int) );
      return  matrix;}

int *matrix1, sum = 0;
/* an example of the function call */
matrix1 = createMatrix( 20, 30 );
if( matrix1 == NULL ) return      -1;
for( i=0; i<20; i++ )                /* row index */
   for( j=0; j<30; j++ )             /* column index */
    sum  += matrix1[i*30 + j];
  .    .    .    .
free( matrix1 );
```

A 2D array is stored in memory as a 1D array

UNIVERSITY OF WOLLONGONG IN DUBAI

# Dynamic Structures

- You can dynamically allocate memory to store structures

*Example:*

```
typedef struct
{   float value;
    int   tolerance;
} resistor;


resistor *ptrRs;      /* pointer to a structure */

ptrRs = malloc( sizeof(resistor) );
if( ptrRs == NULL ) { error recovery action }
.   .   .   .   .   .
ptrRs->value = 33.0; /*use -> to access members of a structure*/
ptrRs->tolerance = 5;
.   .   .   .   .   .
free(ptrRs);
```

UNIVERSITY OF WOLLONGONG IN DUBAI

# Dynamic Structures

- You can dynamically allocate memory to store nested structures

*Example:*

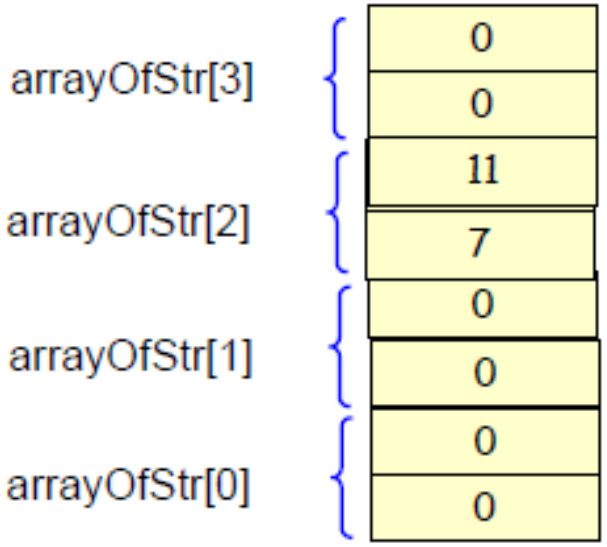
```
typedef struct
{   float value;
    char  tolerance;
} resistor;

typedef struct
{
    int pNumber;
    int quantity;
    resistor spec;
}part;
part *ptr;      /* pointer to a complex structure */

ptr = malloc( sizeof(part) );
.   .   .   .   .   .
ptr->pNumber = 3451;
ptr->spec.tolerance = 2; /* access a member of a nested struct */
```

UNIVERSITY OF WOLLONGONG IN DUBAI

# Array of Structures

*Example:*

```
typedef struct
{
    int   day;
    int   month;
} date;

date *arrayOfStr;

arrayOfStr = calloc( 4, sizeof(date) );
 .    .    .   .
```




UNIVERSITY OF WOLLONGONG IN DUBAI

# Memory Leakage

- A memory leak is a dynamically allocated block of memory that has no pointers pointing to it anywhere in the program.
- Since there are no pointers pointing to the blocks, the program cannot reference them and cannot free them.

```
void sortData( char *data, int size )
{
    char *tempArr;
    tempArr = malloc(size);
    . . . .
    return;
}
```

`free(tempArr);` is missing

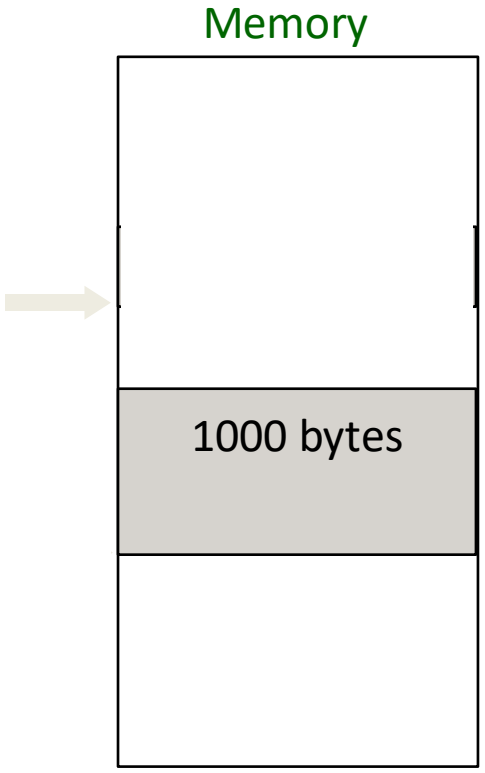
When the function returns, `tempArr` is discarded and after that the memory block cannot be freed

UNIVERSITY OF WOLLONGONG IN DUBAI

# Memory Leakage

- A memory leak can be created by an accidental modification of a pointer

```
    .   .   .   .   .
    char *ptr;
    ptr = malloc( 1000*sizeof(char) );
    .   .   .   .   .   .
    ptr = malloc( 50*sizeof(char) );
    .   .   .   .   .   .
    free( ptr );
    return 0;
}
```




UNIVERSITY OF WOLLONGONG IN DUBAI

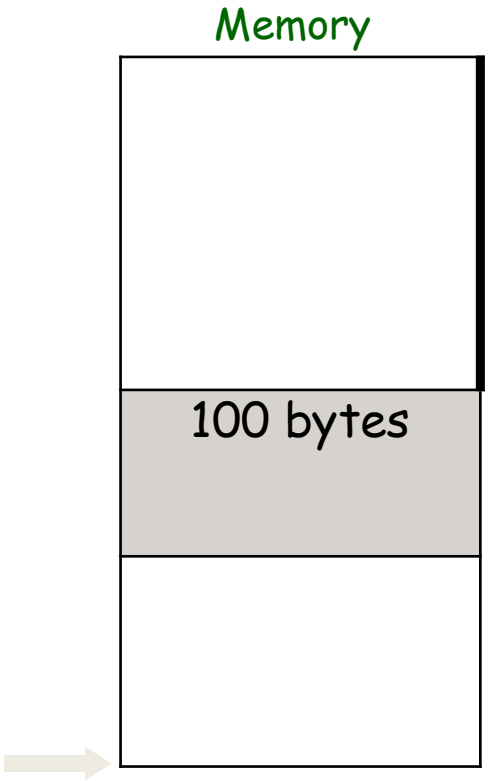
# Memory Leakage

- A memory leak can be created by an incorrectly used memory allocation function

```
.   .   .   .   .
char *ptr;
ptr = malloc( 100 );
.   .   .   .   .   .
ptr = realloc( ptr, 50000 );
.   .   .   .   .   .
```

Proper solution

```
tempPtr = realloc( ptr, 50000 );
if(tempPtr != NULL)
   ptr = tempPtr;
```




UNIVERSITY OF WOLLONGONG IN DUBAI

# Memory Leakage

- Writing a program that dynamically allocates memory, you are fully responsible for releasing that memory that is no longer required.
- Memory leaks result in increased memory consumption and generally result in memory fragmentation.
- This might slow down the performance of your program and the whole system.
- Conventional debugging techniques usually prove to be ineffective for locating the source of memory leaks.
- There are third-party libraries that can trace memory leaks when you debugging a program.

# Memory Manipulation Functions

**`string.h`** library provides a set of functions to support typical operations with memory

1. `memcpy(void *dest, void *src, int size);`

   Copies a block of `size` bytes from the source `*src` to the destination `*dest`

2. `int memcmp(void *loc1, void *loc2, int size);`

   Compares two blocks for a length of size bytes. If the returned value is 0, the memory content is identical

3. `memset(void *dest, char value, int size);`

   Sets `size` bytes of a memory block starting from `*dest` to the same `value`

# memset

*Example:*

```
#include <stdio.h>
#include <string.h>
#define SIZE 1024

char *buffer;
.   .    .    .    .
buffer=malloc(SIZE);
.   .    .    .    .
memset( buffer, 0xFF, SIZE );
.   .    .    .
free( buffer );
.   .    .    .    .
```

UNIVERSITY OF WOLLONGONG IN DUBAI

# memcmp

*Example:*

```
#define SIZE 1024

char *buffer1, *buffer2;
.  .   .   .   .
buffer1 = malloc(SIZE);
readData( buffer1, SIZE );
.  .   .   .   .
buffer2 = malloc(SIZE);
readData( buffer2, SIZE );
.  .   .   .  .  .
if( memcmp( buffer1, buffer2, SIZE ) != 0 )
  printf(" Content is not identical \n");
```

UNIVERSITY OF WOLLONGONG IN DUBAI

# Summary

- Dynamic memory allocation provides an efficient mechanism of utilizing computer resources.
- Some data structures cannot be implemented without dynamic memory allocation.
- The programming complexity of dynamically managed memory is very high.
- Bugs with memory allocation and freeing are very difficult to find.

# Example
# Allocation of Arrays with `calloc`

```
1.  #include <stdlib.h>  /* gives access to calloc */
2.  int scan_planet(planet_t *plnp);
3.
4.  int
5.  main(void)
6.  {
7.        char     *string1;
8.        int      *array_of_nums;
9.        planet_t *array_of_planets;
10.       int       str_siz, num_nums, num_planets, i;
11.       printf("Enter string length and string> ");
12.       scanf("%d", &str_siz);
13.       string1 = (char *)calloc(str_siz, sizeof (char));
14.       scanf("%s", string1);
15.
16.       printf("\nHow many numbers?> ");
17.       scanf("%d", &num_nums);
18.       array_of_nums = (int *)calloc(num_nums, sizeof (int));
19.       array_of_nums[0] = 5;
20.       for (i = 1; i < num_nums; ++i)
21.             array_of_nums[i] = array_of_nums[i - 1] * i;
22.
23.       printf("\nEnter number of planets and planet data> ");
24.       scanf("%d", &num_planets);
25.       array_of_planets = (planet_t *)calloc(num_planets,
26.                                             sizeof (planet_t));
27.       for (i = 0; i < num_planets; ++i)
28.             scan_planet(&array_of_planets[i]);
29.       . . .
30. }
```

```
Enter string length and string> 9 enormous

How many numbers?> 4

Enter number of planets and planet data> 2
Earth    12713.5 1   1.0 24.0
Jupiter 142800.0 4 11.9  9.925
```

```
typedef struct {
        char   name[STRSIZ];
        double diameter;
        int    moons;
        double orbit_time,
               rotation_time;

} planet_t;
```

UNIVERSITY OF WOLLONGONG IN DUBAI