Short Notes on Dynamic Memory Allocation

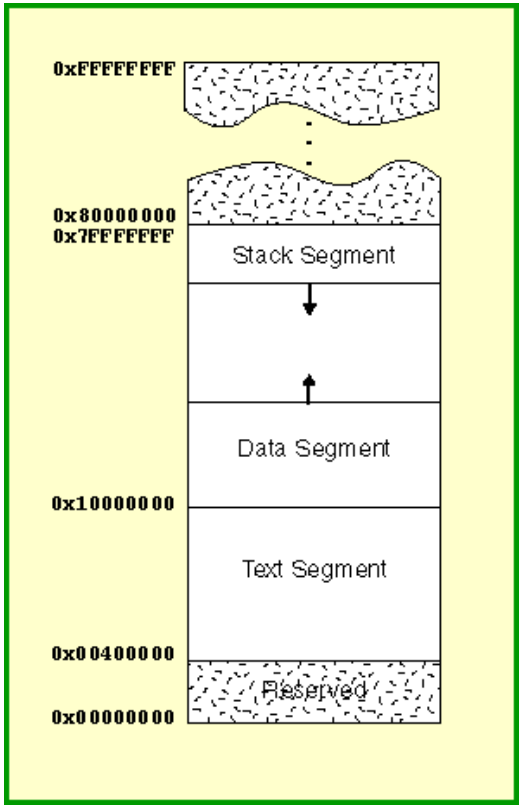
Dynamic Memory Allocation in C/C++

Motivation

```
 /*   a[100]  vs. *b or *c     */
Func(int array_size)
{
    double  a[100], *b, *c;
    b = (double *) malloc(array_size * sizeof(double));  /* allocation in C*/
    c = new double[array_size];  /* allocation in C++ */
}
```

- The size of the problem often can not be determined at “compile time”.
- Dynamic memory allocation is to allocate memory at “run time”.
- Dynamically allocated memory must be referred to by pointers.

Stack vs Heap



When a program is loaded into memory:

- Machine code is loaded into **text** segment
- **Stack** segment allocate memory for automatic variables within functions
- **Heap** segment is for dynamic memory allocation

Memory Allocation/Free Functions in C/C++

C:

- void *malloc(size_t number_of_bytes)
  -- allocate a contiguous portion of memory
  -- it returns a pointer of type void * that is the beginning place in memory of allocated portion of size number_of_bytes.
- void free(void * ptr);
  -- A block of memory previously allocated using a call to malloc, calloc or realloc is deallocated, making it available again for further allocations.

C++:

- "new" operator
  -- pointer = new type
  -- pointer = new type [number_of_elements]
  -- It returns a pointer to the beginning of the new block of memory allocated.
- "delete" operator
  -- delete pointer;
  -- delete [] pointer;

Example 1

```
Func() /* C++ version */
{
        double *ptr;
        ptr = new double;
        *ptr = -2.5;
}
Func_C()  /* C version */
{
         double *ptr;
         ptr = (double *) malloc(sizeof(double));
         ….
}
```

- **Illustration**

Name	Type	Contents	Address
ptr	double pointer	0x3D3B38	0x22FB66

Memory heap (free storage we can use)	
…	
0x3D3B38	-2.5
0x3D3B39	

Example 2

```
Func() /* C++ version */
{
        double *ptr, a[100];
        ptr = new double[10];   /* in C, use: ptr = (double *)malloc(sizeof(double)*10); */
        for(int i = 0; i < 10; i++)
            ptr[i] = -1.0*i;
        a[0] = *ptr;
        a[1] = *(ptr+1);   a[2] = *(ptr+2);
}
```

- **Illustration**

Name	Type	Contents	Address
ptr	double array pointer	0x3D3B38	0x22FB66

Memory heap (free storage we can use)	
…	
0x3D3B38	0.0
0x3D3B39	-1.0
…	

Example 3

- Static array of dynamically allocated vectors

```
Func() /* allocate a contiguous memory which we can use for 20 ×30 matrix */
{
      double *matrix[20];
      int   i, j;
      for(i = 0; i < 20; i++)
         matrix[i] = (double *) malloc(sizeof(double)*30);

      for(i = 0; i < 20; i++)
      {
           for(j = 0; j < 30; j++)
                matrix[i][j] =  (double)rand()/RAND_MAX;
      }

}
```

Example 4

- Dynamic array of dynamically allocated vectors

```
Func() /* allocate a contiguous memory which we can use for 20 ×30 matrix */
{
      double **matrix;
      int   i, j;

      matrix = (double **) malloc(20*sizeof(double*));
      for(i = 0; i < 20; i++)
         matrix[i] = (double *) malloc(sizeof(double)*30);

      for(i = 0; i < 20; i++)
      {
           for(j = 0; j < 30; j++)
                matrix[i][j] =  (double)rand()/RAND_MAX;
      }
}
```

Example 5

- Another way to allocate dynamic array of dynamically allocated vectors

```
Func() /* allocate a contiguous memory which we can use for 20 ×30 matrix */
{
      double **matrix;
      int   i, j;

      matrix = (double **) malloc(20*sizeof(double*));
      matrix[0[ = (double*)malloc(20*30*sizeof(double));

      for(i = 1; i < 20; i++)
         matrix[i] = matrix[i-1]+30;

      for(i = 0; i < 20; i++)
      {
           for(j = 0; j < 30; j++)
                matrix[i][j] =  (double)rand()/RAND_MAX;
      }
}
```

Release Dynamic Memory

```
Func()
{
    int *ptr, *p;
    ptr = new int[100];
    p = new int;
    delete[] ptr;
    delete  p;
}
```