Week 5 Lecture 1

NWEN 241
Systems Programming

Jyoti Sahni

Jyoti.sahni@ecs.vuw.ac.nz

Admin stuff

- Assignment 2 released

- Term Test:
  - Date: 17:00 - 18:00, April 19 (Friday), week 6, after the mid-term break
    - Rooms for the test: HMLT205, KKLT303
    - Class split: TBA (at the course wiki)
  - Covers week 1 to week 6(lecture 1) lecture topics
  - Test is 45 minutes long, max marks: 45
  - Multiple choice and short answer questions
  - Take the weekly practice quiz to prepare for the test

Content

- Introduction Dynamic Memory Management
- `calloc()`
- `free()`
- `malloc()`
- `realloc()`
- Common Problems with Dynamic Memory

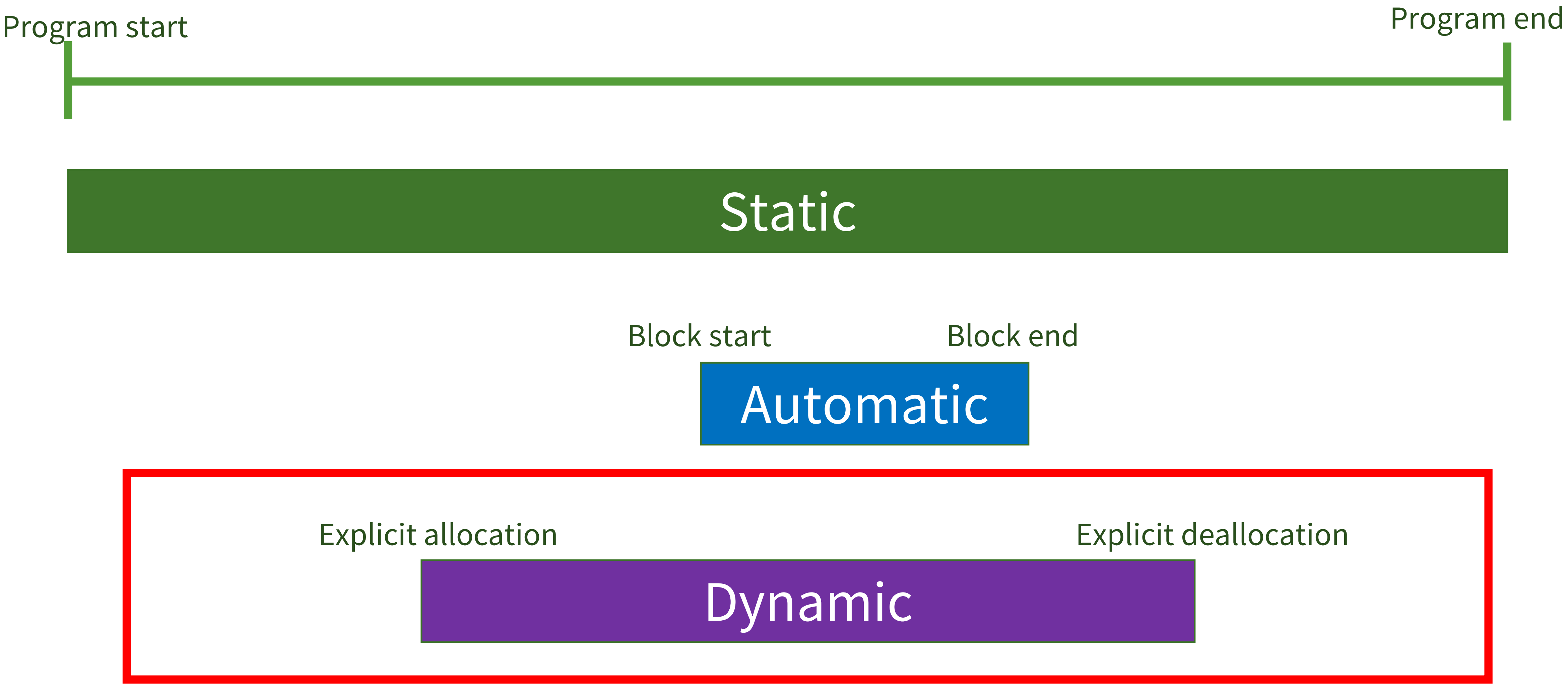
Recap: Usage of Pointers

1) Provide an alternative means of accessing information stored in arrays

2) Provide an alternative (and more efficient) means of passing parameters to functions

3) Enable dynamic data structures, that are built up from blocks of memory allocated from the heap at run time

Recap: Lifetime / Storage Duration



Why Allocate Memory Dynamically?

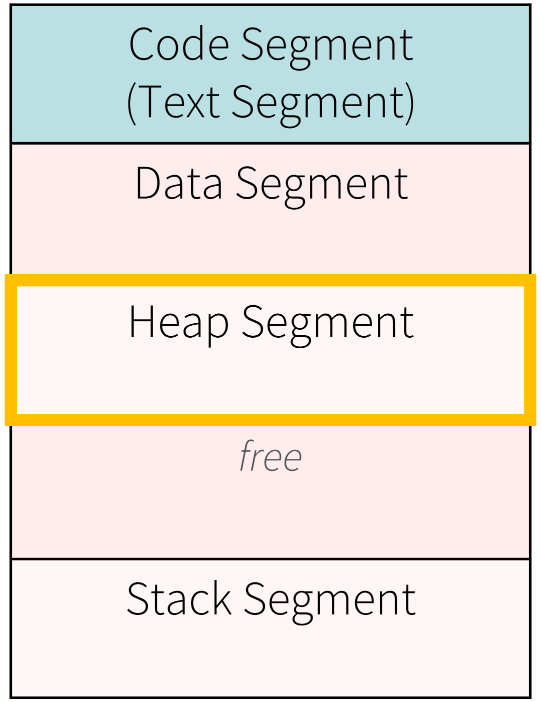
- **It may not be possible to know ahead of time the space needed by a variable (e.g., array) for storing data**

- With static allocation:
  - If predefined size is small, it may not be enough space to hold data, resulting in **program failure**
  - If predefined size is big, most of the space will not be used causing **waste or inefficiency**

Dynamic Memory Allocation

- **Allow the program to dynamically allocate memory for some variables (e.g. arrays) during the program execution**

- Approach:
  - Program has routines allowing user to request some amount of memory,
  - the user then uses this memory, and
  - returns it when they are done.
  - Memory is allocated in the *Heap Segment*



Dynamic Memory Management Functions

- `calloc` - allocate *array* of memory
- `malloc` - allocate *a single block* of memory
- `realloc` – extend or reduce the amount of space allocated previously
- `free` - free up a piece of memory that is no longer needed

⚠ Memory allocated dynamically does not go away at the end of functions, you MUST explicitly **free** it up

calloc – Allocate Memory for Array

- Function prototype:

```
void *calloc(size_t num, size_t esize)
```

  - size_t – special type used to indicate sizes, unsigned int
  - num – number of elements to be allocated in the array
  - esize – size (in bytes) of a single element to be allocated
    - to get the correct value, use `sizeof(<type>)`
    - memory of size `num*esize` is allocated
- `calloc` returns the address of the 1st byte of this memory
  - Cast the returned address to the appropriate type
- If not enough memory is available, `calloc` returns `NULL`

calloc Example

```
float *nums;
int a_size;
int idx;

printf("Read how many numbers:");
scanf("%d",&a_size);
nums = (float *)calloc(a_size, sizeof(float));

/* nums is now an array of floats of size a_size */
for (idx = 0; idx < a_size; idx++) {
  printf("Please enter number %d: ",idx+1);
  scanf("%f", nums+idx); /* read in the floats */
}

/* Calculate average, etc. */
```

calloc Example

```
float *nums;
…

nums = (float *)calloc(a_size, sizeof(float));
```

calloc Example

```
float *nums;
int a_size;
int idx;

printf("Read how many numbers:");
scanf("%d",&a_size);
nums = (float *)calloc(a_size, sizeof(float));

/* nums is now an array of floats of size a_size */
for (idx = 0; idx < a_size; idx++) {
  printf("Please enter number %d: ",idx+1);
  scanf("%f", nums+idx); /* read in the floats */
}

/* Calculate average, etc. */
```

Any potential issues with this code?

calloc Example

- Always check the return value of calloc, malloc or realloc!

```
float *nums;
int a_size;
int idx;

printf("Read how many numbers:");
scanf("%d",&a_size);
nums = (float *) calloc(a_size, sizeof(float));

if(nums == NULL) {
      /* exit or do some other stuff */
}
…
```

Next Lecture

- Dynamic memory allocation (cont.)