

NWEN 241

Storage Class & Dynamic Memory Allocation

Qiang Fu

School of Engineering and Computer Science
Victoria University of Wellington



This Lecture

- Overview of storage class
- Dynamic memory allocation
 - calloc, malloc, realloc, free

15/05/2015

2

Storage Class

- Every object is stored in memory
- But, not everyone is treated the same way
- There are two storage classes:
 - Automatic (automatic objects)
 - Static (static objects)

15/05/2015

3

Storage Class

- Automatic objects
 - Created when declared within a block if no storage specification is mentioned (`int i; float f;`)
 - Memory/storage is allocated when declared
 - Local to the block
 - Memory is deallocated when the execution of the block is finished (the object no longer exists)

15/05/2015

4

Storage Class

- Automatic objects
 - Revisit the “Blocks” in a previous lecture

```
int main(void)
{ int i = 0, x = 0;

    for (int i=-4; i < 4; i++) /* Only for C99. i is re-declared. */
    { x += i;
    }

    while (i < 2*4)
    { x += i;
      i++;
    }

    do
    { x += i;
      i++;
    } while (i < 3*4);

    return 0;
}
```

15/05/2015

NWEN 241: Systems Programming

5

Storage Class

- Automatic objects
 - Revisit the “Blocks” in a previous lecture

```
int main(void)
{ int i = 0, x = 0;

    for (int i=-4; i < 4; i++) /* i will be used by the */
                                /* while and do-while loops, */
                                /* but not the for loop */
                                /* Only for C99. i is re-declared. */
                                /* only valid within this block. */
    { x += i;
    }

    while (i < 2*4)
    { x += i;
      i++;
    }

    do
    { x += i;
      i++;
    } while (i < 3*4);

    return 0;
}
```

15/05/2015

NWEN 241: Systems Programming

6

Storage Class

- Static objects
 - Created when declared outside all blocks
 - Created when declared with the keyword **static** within a block
 - May be local to a block or external to all blocks
 - Memory/storage and the value in the memory are retained even after the execution of the block

```
int i = 0, x = 0;
int main(void)
{ int i = 0, x = 0;
  for (; i < 4; i++)
    f();
  return 0;
}
void f()
{ static i = 0, x = 0;
  x += i;
}
```

15/05/2015

7

Static Memory Allocation

- Compile time (data segment / stack)
 - Data segment: the area of memory used for static objects
 - Stack: the area of memory used for automatic objects

15/05/2015

8

Dynamic Memory Allocation

- Run time (heap)
 - Heap: the area of memory that we (programmers) may explicitly ask for memory space to store objects
 - In return, we need to explicitly free the memory if we do not need it anymore

Dynamic Memory Allocation

- Do we need explicitly to ask for storage
 - Case 1:

```
float *ppi; *ppi = 3.14;  
/* memory has been allocated to ppi, */  
/* but ... */
```

Dynamic Memory Allocation

- Do we need explicitly to ask for storage
 - Case 1:

```
float *ppi; *ppi = 3.14;  
/* memory has been allocated to ppi, */  
/* but not explicitly to *ppi */
```

Dynamic Memory Allocation

- Do we need explicitly to ask for storage
 - Case 2:

```
char source[] = "this is an array";  
  
char target[] = "";  
  
strcpy(source, target);
```

Dynamic Memory Allocation

- Do we need explicitly to ask for storage
 - Case 2:

```
char source[] = "this is an array";

char target[] = "";

strcpy(source, target);

/* target[] does not seem to have enough */
/* space to hold source[] */
```

15/05/2015

13

Dynamic Memory Allocation

- Do we need explicitly to ask for storage
 - Case 3:

```
/* Let us create an array that we do not
 * know its size prior to program execution
 */
```

```
char a[];    /* is this correct? */
```

- Case 4: dynamic data structures

15/05/2015

14

Dynamic Memory Allocation

- Functions that allow us to request storage

```
– calloc()
typedef unsigned size_t;
void *calloc(size_t n, size_t el_size);
/* calloc() allocates contiguous space in memory */
/* for an array of n elements. the size of the */
/* element is el_size bytes. the memory space is */
/* initialised with all bits set to zero */

/* calloc() returns a ...??? */
```

15/05/2015

15

Dynamic Memory Allocation

- Functions that allow us to request storage

```
– calloc()
typedef unsigned size_t;
void *calloc(size_t n, size_t el_size);
/* calloc() allocates contiguous space in memory */
/* for an array of n elements. the size of the */
/* element is el_size bytes. the memory space is */
/* initialised with all bits set to zero */

/* calloc() returns a pointer to void */

int *pa, n;
pa = calloc(n, sizeof int);
```

15/05/2015

16

Dynamic Memory Allocation

- Functions that allow us to request storage

```
- calloc()
typedef unsigned size_t;
void *calloc(size_t n, size_t el_size);
/* calloc() allocates contiguous space in memory */
/* for an array of n elements. the size of the */
/* element is el_size bytes. the memory space is */
/* initialised with all bits set to zero */

/* calloc() returns a pointer to void */

int *pa, n;
pa = calloc(n, sizeof int);
pa = (int*)calloc(n, sizeof int); /*not necessary */
```

15/05/2015

17

Dynamic Memory Allocation

- Functions that allow us to request storage

```
- malloc()

void *malloc(size_t size);
/* malloc() allocates a block of space in memory. */
/* the size of the memory block is size bytes. */
/* the memory space is NOT initialised. */

/* malloc() returns a pointer to no type (void). */
/* the ptr points to the base of the memory block */

int *pa;
int n;
pa = malloc(n * sizeof int);
```

15/05/2015

18

Dynamic Memory Allocation

- Functions that allow us to request storage

```
- realloc()

void *realloc(void *ptr, size_t size);
/* realloc() changes the size of the memory block */
/* pointed to by ptr to size bytes. Any new space */
/* is NOT initialised. */

/* if possible the same base addr will be returned */

int *pa;
int n, m;
pa = malloc(n * sizeof int);
pa = realloc(pa, m * sizeof int); /* problematic? */
```

15/05/2015

19

Dynamic Memory Allocation

- Functions that allow us to request storage

```
- realloc()

void *realloc(void *ptr, size_t size);
/* realloc() changes the size of the memory block */
/* pointed to by ptr to size bytes. Any new space */
/* is NOT initialised. */

/* if possible the same base addr will be returned */

int *pa;
int n, m;
pa = malloc(n * sizeof int);
pa = realloc(pa, m * sizeof int); /* NULL could be */
/* returned */
```

15/05/2015

20

Dynamic Memory Allocation

- Functions that allow us to request storage
 - realloc()

```
void *realloc(void *ptr, size_t size);
/* realloc() changes the size of the memory block */
/* pointed to by ptr to size bytes. Any new space */
/* is NOT initialised. */

/* if possible the same base addr will be returned */

int *pa, *tmp;
int n, m;
pa = malloc(n * sizeof int);
tmp = realloc(pa, m * sizeof int);
...      /* Check if tmp is NULL */
```

15/05/2015

21

Dynamic Memory Allocation

- Functions that allow us to request storage
 - realloc()

```
void *realloc(void *ptr, size_t size);
/* realloc() changes the size of the memory block */
/* pointed to by ptr to size bytes. Any new space */
/* is NOT initialised. */

/* if possible the same base addr will be returned */
```

Case 1: Request failed

NULL is returned.

Original allocated space left untouched.

15/05/2015

22

Dynamic Memory Allocation

- Functions that allow us to request storage
 - realloc()

```
void *realloc(void *ptr, size_t size);
/* realloc() changes the size of the memory block */
/* pointed to by ptr to size bytes. Any new space */
/* is NOT initialised. */

/* if possible the same base addr will be returned */
```

Case 2: Enough contiguous space right after the original space

Expand the original space with contiguous space.

The same base address is returned.

The original/old content is left untouched.

15/05/2015

23

Dynamic Memory Allocation

- Functions that allow us to request storage
 - realloc()

```
void *realloc(void *ptr, size_t size);
/* realloc() changes the size of the memory block */
/* pointed to by ptr to size bytes. Any new space */
/* is NOT initialised. */

/* if possible the same base addr will be returned */
```

Case 3: No enough contiguous space to expand

Allocate memory space from a different location.

A new/different base address is returned.

The original/old content is copied to the new space.

The old/original space is freed.

15/05/2015

24

Dynamic Memory Allocation

- Functions that allow us to request storage
 - Successful memory request
 - The base address of the allocated memory space is returned
 - Unsuccessful memory request
 - NULL is returned
 - To avoid dereferencing a NULL pointer, do this:

```
if (pa == 0)
    abort(); /* memory request failed! */
```

Dynamic Memory Allocation

- Free the allocated memory
 - Remember the memory allocated to our request is from the heap
 - If we do not free the allocated memory, no one does (C does NOT have garbage collection as Java does)

```
void free(void *ptr);
/* free() deallocates the memory space */
/* pointed by ptr. ptr must be the base */
/* addr of the memory space previously */
/* allocated by calloc(), malloc() or
realloc(). */
```

Dynamic Memory Allocation

- Free the allocated memory
 - Remember the memory allocated to our request is from the heap
 - If we do not free the allocated memory, no one does (C does NOT have garbage collection as Java does)

```
void free(void *ptr);
/* it is the memory space pointed by ptr */
/* that is freed, not ptr. ptr is still */
/* there with original value - base addr */
```

Dynamic Memory Allocation

- An example – string copy

```
void strcpy(char *, char *);

int main(void)
{ char source[] = "this is an array";
  /* char target[] = "this is another array"; */
  char *target = malloc(sizeof(source));
  strcpy(source, target);
  ...
  return 0;
}

void strcpy(char *s, char *t)
{ while (*t++ = *s++);
}
```

Dynamic Memory Allocation

- Another example – string concatenation

```
int main(void)
{ char source1[] = "this is ";
  char source2[] = "a pointer";

  /* can we do: strcat(source1, source2); */

  ...
  return 0;
}
```

15/05/2015

29

Dynamic Memory Allocation

- Another example – string concatenation

```
int main(void)
{ char source1[] = "this is ";
  char source2[] = "a pointer";

  /* can we do: strcat(source1, source2); */
  /* source1 does not have enough space to hold */
  /* both source1 and source2 */

  ...
  return 0;
}
```

15/05/2015

30

Dynamic Memory Allocation

- Another example – string concatenation

```
int main(void)
{ char source1[] = "this is ";
  char source2[] = "a pointer";

  char *target;
  target = malloc(sizeof(source1)+sizeof(source2));
  /* fill this gap ... */
  /* fill this gap ... */

  ...
  return 0;
}
```

15/05/2015

31

Dynamic Memory Allocation

- Another example – string concatenation

```
int main(void)
{ char source1[] = "this is ";
  char source2[] = "a pointer";

  char *target;
  target = malloc(sizeof(source1)+sizeof(source2));
  strcat(target, source1); /* problematic? */

  strcat(target, source2);

  ...
  return 0;
}
```

15/05/2015

32

Dynamic Memory Allocation

- Another example – string concatenation

```
int main(void)
{ char source1[] = "this is ";
  char source2[] = "a pointer";

  char *target;
  target = malloc(sizeof(source1)+sizeof(source2));
  strcat(target, source1);
  /* '\0' could be in the allocated memory block */
  strcat(target, source2);

  ...
  return 0;
}
```

15/05/2015

33

Dynamic Memory Allocation

- Another example – string concatenation

```
int main(void)
{ char source1[] = "this is ";
  char source2[] = "a pointer";

  char *target;
  target = malloc(sizeof(source1)+sizeof(source2));
  strcpy(target, source1);
  /* or ... */
  strcat(target, source2);

  ...
  return 0;
}
```

15/05/2015

34

Next Lecture

- Dynamic data structures

15/05/2015

35