Foundations of C Programming (Structured Programming)
- Dynamic Memory Allocation

Outline

- Static memory allocation and dynamic memory allocation
- Memory allocation functions

Memory Allocation

- Two ways
  - static memory allocation (静态内存分配)
    - Memory is allocated invisibly when a variable is declared
      - e.g., `int i;` four bytes are allocated to store the value of i.
  - dynamic memory allocation (动态内存分配)
    - Memory is allocated visibly by calling some allocation functions.

Problems with Static Memory Allocation

- Memory allocation determined at compiling time (编译时):
  - `int x;` // 4 bytes
  - `char y;` // 1 byte
  - `float a;` // 4 bytes
  - `double b;` // 8 bytes
  - `int a[10];` // 10*4 bytes
- The type of the variable determines how much memory the compiler (编译程序） allocates (分配)

Static Memory Allocation

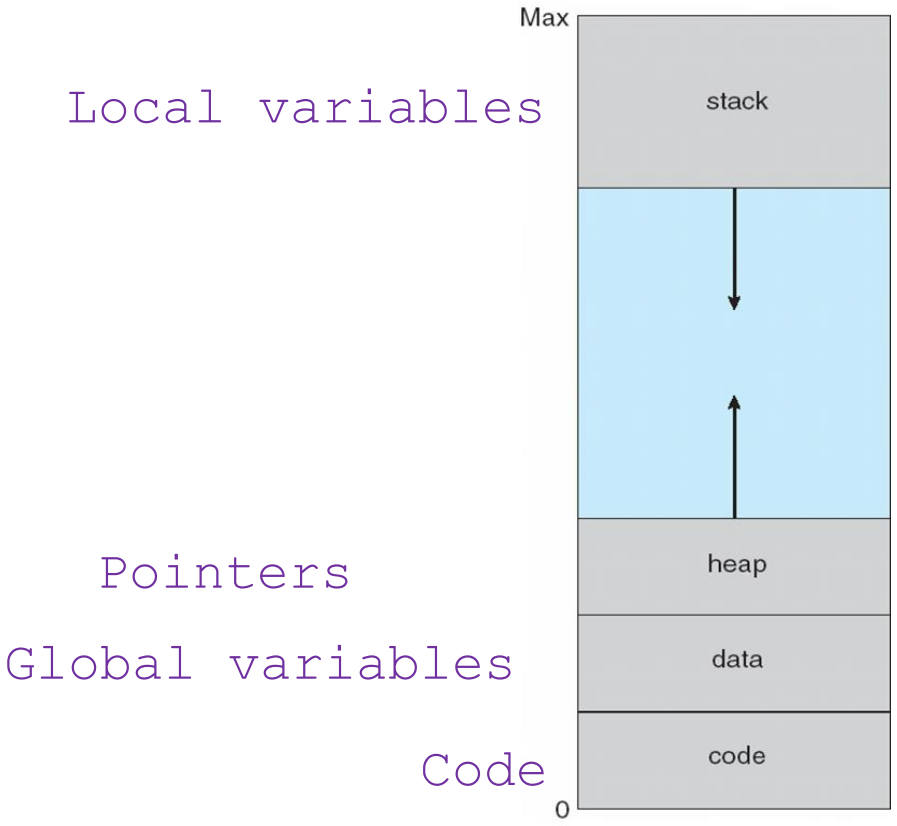
- If too much memory is allocated and then not used, there is a waste of memory.
- If not enough memory is allocated, the program is not able to handle the input data.

```
int grade[100];
int i = 0;
scanf("%d", &grade[i]);
while(grade[i] != -1)
{
  printf("%d\n", grade[i]);
  i++;
  scanf("%d", &grade[i]);
}
```

What if there are only 5 students?
What if there are 1000 students?

Dynamic Memory Allocation

- Dynamic memory allocation allocates memory at execution time or runtime (运行时) when needed, free the memory (释放内存) when the memory is not needed



Dynamic Memory Allocation Functions

- Two basic functions
  - `void *malloc(long size)`
    - Allocates a block of memory of specified size and returns a pointer of type void
    - We can assign it to a pointer of any type
    - A NULL pointer is returned if there is not enough space available
  - `void free(void *ptr)`
    - releases (释放) the used memory when it is no longer needed
- To use these functions, the stdlib.h header file must be included.

An Example: Allocate Memory for An Integer

```
int *ptr;
ptr = (int*)malloc(sizeof(int));
if (ptr == NULL)
   return;
*ptr = 23;
printf("Value stored is %d\n", *ptr);
free(ptr);
```

Example: Allocate Memory for A Number of Integers

```
int number, i;
int *ptr;

printf("How many integers would you like store? ");
scanf("%d", &number);
ptr = (int*)malloc(number * sizeof(int));
if(ptr == NULL) //Memory is not allocated successfully
   return;

for(i = 0; i < number; i++){ *(ptr + i) = i; }

for(i = number; i > 0; i--)
   printf("%d\n", *(ptr + (i - 1)));
free(ptr);
```

What is the output of this program if number is 5?

Example: Allocate Memory for A Structure

```
typedef struct{
    char name[15];
    int  id;
    char dept[5];
    char gender;
} studentRecord;

struct studentRecord *ps;

ps = (studentRecord*)malloc(sizeof(studentRecord));
strcpy(ps -> name, "Tom Hanks");
ps -> id = 12345;
strcpy(ps  -> dept, "COMP");
ps -> gender = 'M';

printf("student's name is %s, id is %d, department is %s,
    gender is %c \n", ps -> name, ps -> id, ps -> dept, ps ->
    gender);
free(ps);
```

More Functions

- Refer to the Internet for more memory functions.
  - void *calloc(int n, int elem-size)
    - allocates n blocks of storage, each of the same size (elem-size), and then sets all bytes to zero
    - A null pointer is returned if there is not enough space
  - void *realloc(void *ptr, int newsize)
    - allocates a new memory space of size newsize to the pointer variable ptr
    - A null pointer is returned if there is not enough space
    - The newsize maybe larger or smaller than the old size

Memory Leak

- Memory leak (内存泄漏) happens when memory is allocated (分配) but not released (释放)
- It will cause an application to gradually consume memory (消耗内存) then the available memory for other applications is reduced.

The allocated memory must be freed!!!

Summary

- Dynamic memory allocation can save memory when data size is un-determined
- Functions can be used to allocate and free memory