Object oriented programming In C++ (ACE 1313)

Dynamic, file, Project #2

Professor 최학남

xncui@inha.ac.kr

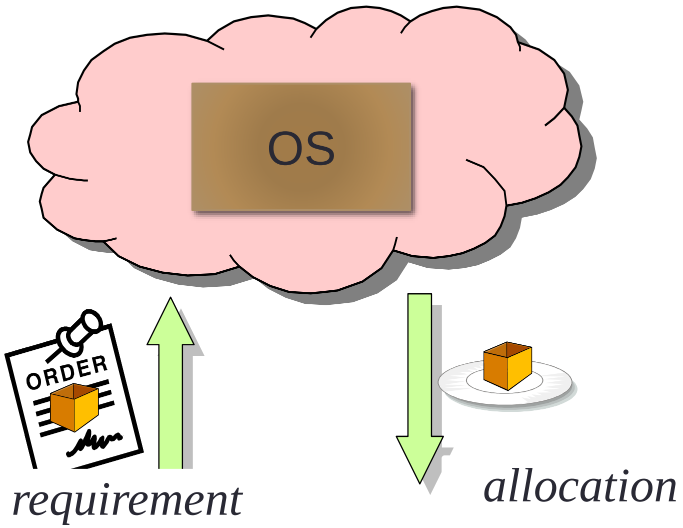
Office: high-tech 401

Contents

- Dynamic memory allocation
- File input and output
- Project#2

Dynamic memory allocation

- Memory allocation
  - ✓ Static
  - ✓ Dynamic
    - Dynamic allocation is the automatic allocation of memory in C/C++.



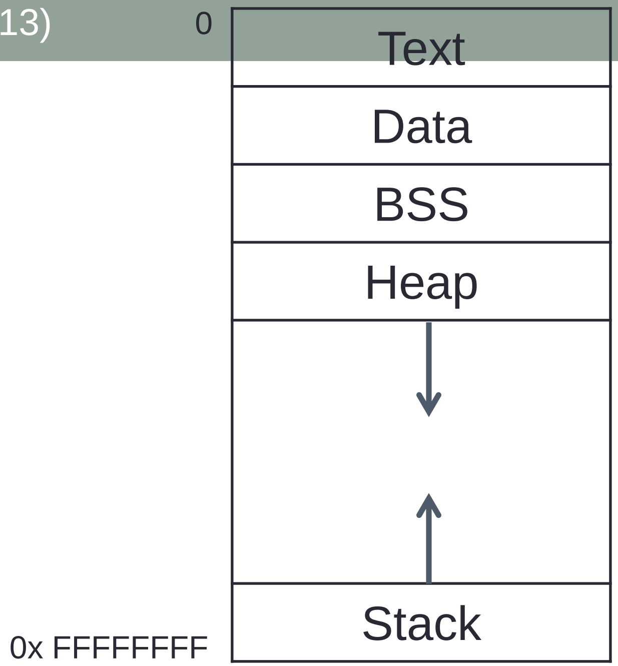
```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
 int *p;
 p = (int *)malloc( sizeof(int) );
 ...
}
```

Program

1. When a program executes, the operating system gives it a stack and a heap to work with.
2. The stack is where global variables, static variables, and functions and their locally defined variables reside.
3. The heap is a free section for the program to use for allocating memory at runtime.

Dynamic memory allocation

- How is memory organized?
  - ✓Text – code, constant data
  - ✓Data – initialized global and static variables
  - ✓BSS – Block Started by Symbol
  - ✓Heap – dynamic memory
    - Structures whose size varies dynamically (e.g. variable length arrays or strings).
    - Structures that are allocated dynamically (e.g. records in a linked list).
    - Structures created by a function call that must survive after the call returns.
  - ✓Stack – local variables
    - Local variables for functions, whose size can be determined at call time.
    - Information saved at function call and restored at function return:



0

Text

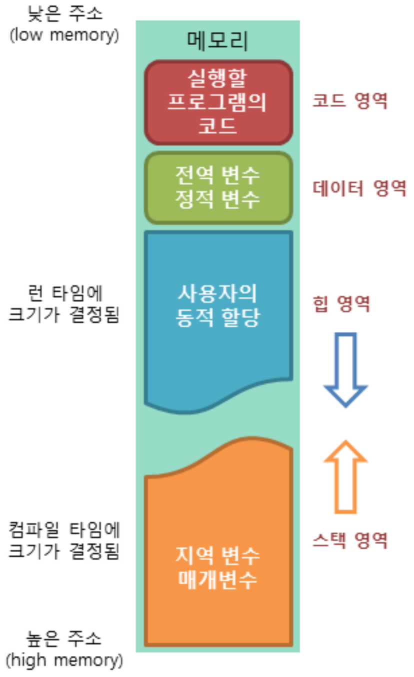
Data

BSS

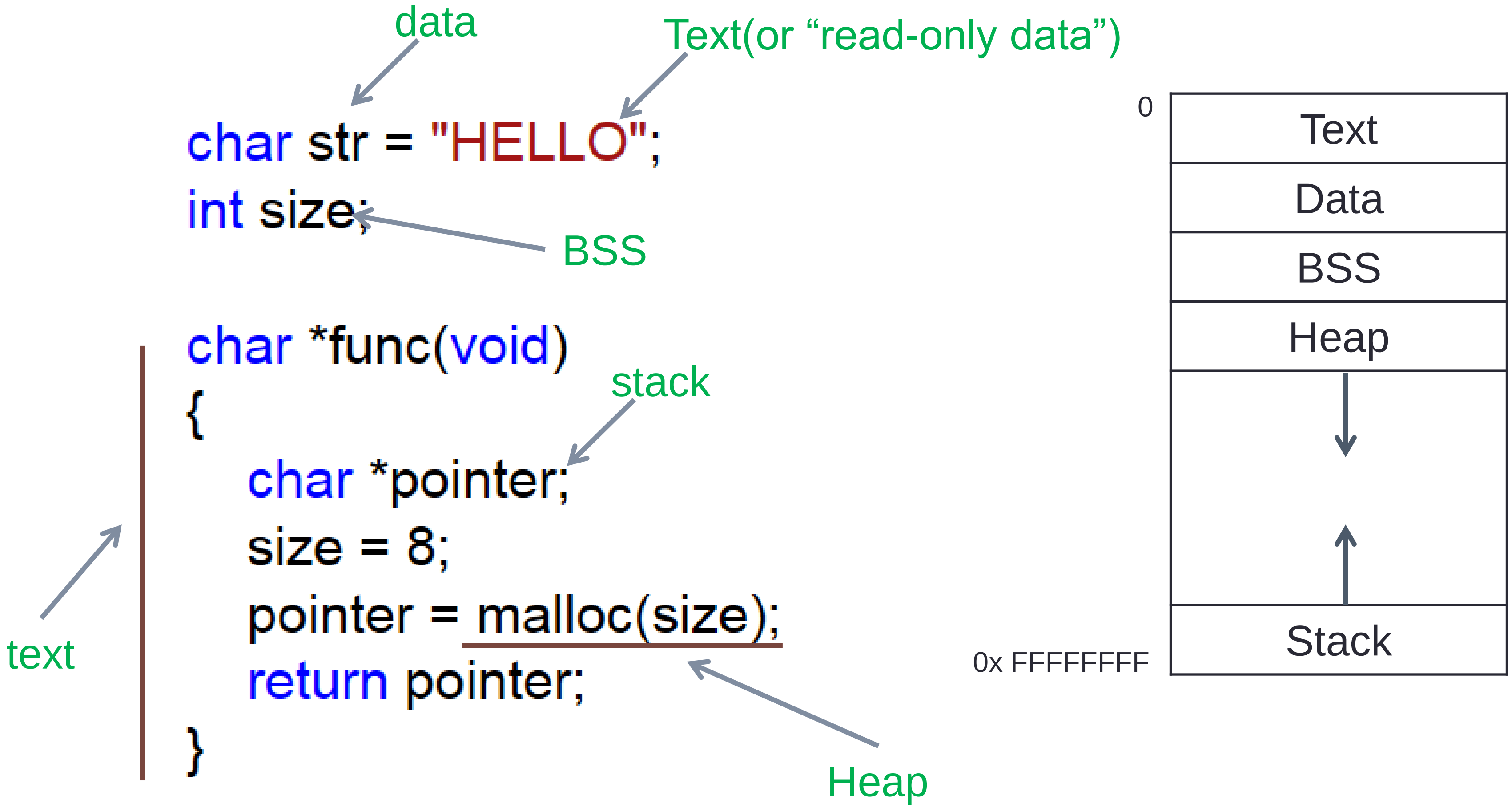
Heap

Stack

0x FFFFFFFF



Dynamic memory allocation

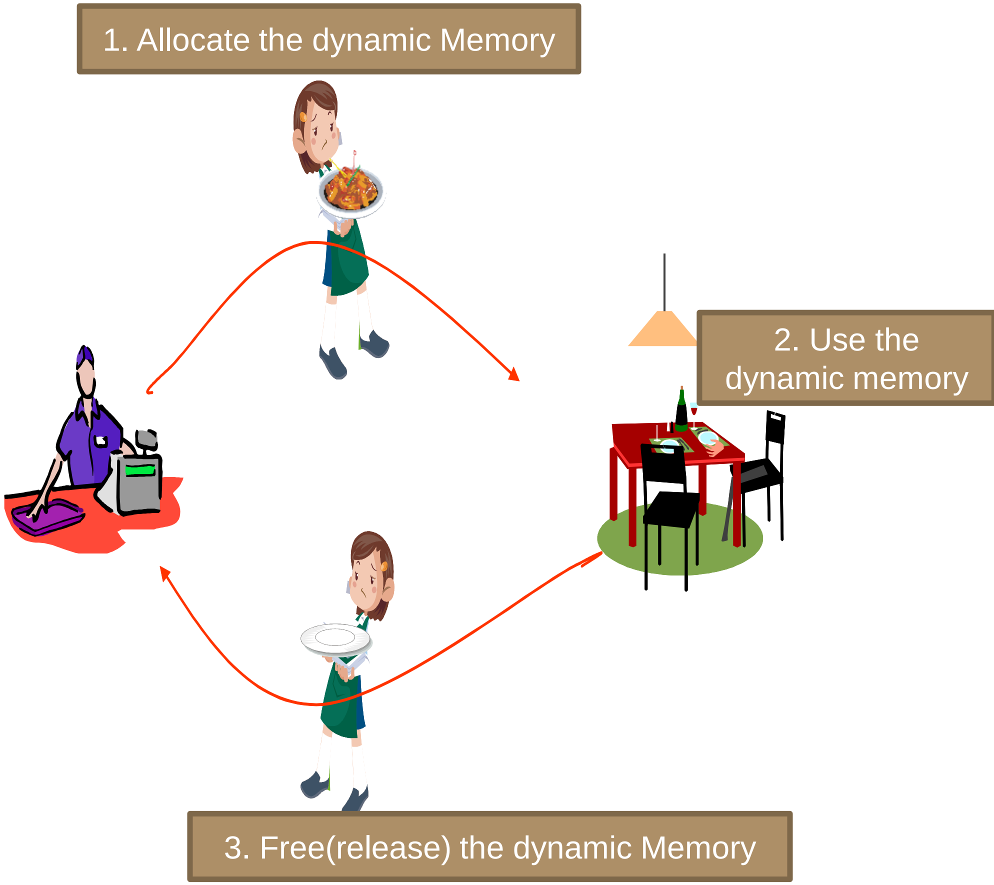


```
char str = "HELLO";
int size;

char *func(void)
{
    char *pointer;
    size = 8;
    pointer = malloc(size);
    return pointer;
}
```

Dynamic memory allocation

- Process of dynamic memory allocation

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  int *pi;

  pi = (int *)malloc(sizeof(int)); // ①

  if( pi == NULL )            // check the return value
 {
    printf(“dynamic memory allocation error\n");
    exit(1);
  }
  *pi = 100;   // ②
  printf("%d\n", *pi);
  free(pi);    // ③
  return 0;
}
```



1. Allocate the dynamic Memory

2. Use the dynamic memory

3. Free(release) the dynamic Memory

Dynamic memory allocation

Process of dynamic memory allocation

```
void *malloc(size_t size);
```

- malloc() : function for allocating the memory
- size: byte
- the malloc() function return the first block address
- If can not use the memory space, return the NULL value

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

- free() : free (or release) the memory
- ptr is memory address that point to the address of the malloc()

Dynamic memory allocation

- new, delete operator

```
int *p = new int[10];
…;
delete[ ] p;
```

EX #1: Dynamic memory allocation

```
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
        char *pc = NULL;

        pc = (char *)malloc( sizeof(char) );
        if( pc == NULL )
        {
                printf( “MEMORY ALLOCATION ERROR\n" );
                exit(1);
        }
        *pc = 'm';
        printf( "*pc = %c\n", *pc );
        free( pc );

        return 0;
}
```

EX #2: Dynamic memory allocation

```c
// 메모리 동적 할당
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
        char *pc = NULL;
        int i = 0;
        pc = (char *)malloc(100*sizeof(char));
        if( pc == NULL )
        {
                printf(“memory error\n");
                exit(1);
        }
        for(i=0;i<26;i++)
        {
                *(pc+i) = 'a'+i;
        }
        *(pc+i) = 0;   // add the NULL character

        printf("%s\n", pc);
        free(pc);
        return 0;
}
```

```
abcdefghijklmnopqrstuvwxyz
```

EX #3: Dynamic memory allocation

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        int *pi;

        pi = (int *)malloc(5 * sizeof(int));

        if(pi == NULL){
          printf("메모리 할당 오류\n") ;
          exit(1);
        }

        pi[0] = 100;            // *(pi+0) = 100;와 같다.
        pi[1] = 200;            // *(pi+1) = 200;와 같다.
        pi[2] = 300;            // *(pi+2) = 300;와 같다.
        pi[3] = 400;            // *(pi+3) = 400;와 같다.
        pi[4] = 500;            // *(pi+4) = 500;와 같다.

        free(pi);
        return 0;
}
```

EX #4: Dynamic memory allocation

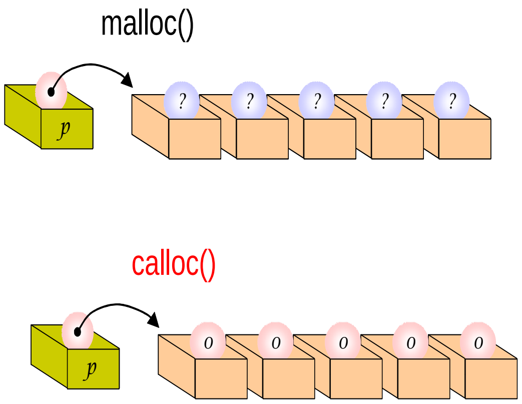
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Book {
        int number;
        char title[10];
};

int main(void)
{
        struct Book *p;

        p = (struct Book *)malloc(2 * sizeof(struct Book));

        if(p == NULL){
          printf(“memory error\n") ;
          exit(1);
        }

        p->number = 1;
        strcpy(p->title,"C/C++ Programming");

        (p+1)->number = 2;
        strcpy((p+1)->title,"Data Structure");

        free(p);
        return 0;
}
```

calloc() and realloc()

```
void *calloc(size_t n, size_t size);
```

- calloc(): allocation + zero initialization
- Ex:

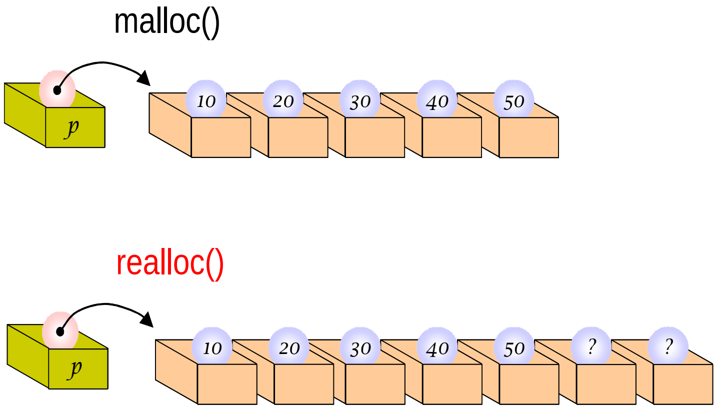
```
int *p;
p = (int *)calloc(5,sizeof(int));
```



```
void *realloc(void *memblock, size_t size);
```

- realloc() : resize the allocated memory size
- Ex:

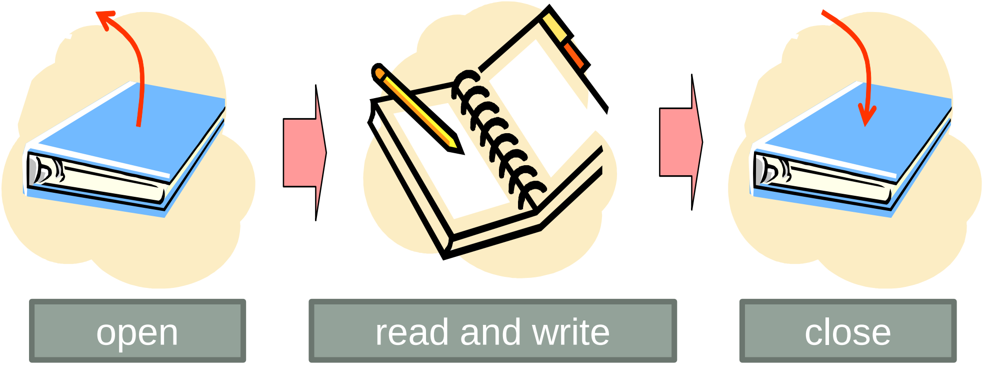
```
int *p;
p = (int *)malloc(5*sizeof(int));
p = (int *)realloc(p, 7 * sizeof(int));
```



File input and output

- File
  - ✓Text file
  - ✓Binary file
  - ✓Using *FILE* Structure pointer (file pointer) the access the file

- Must be use the following process



open

read and write

close

File open

```
FILE *fopen(const char *name, const char *mode)
```

mode	Description
“r”	Opens the file for reading
“w”	Opens the file as an empty file for writing. If the file exists, its contents are destroyed
“a“	Opens the file for writing at the end of the file (appending) without removing the EOF marker before writing new data to the file; this creates the file first if it doesn't exist
“r+“	Opens the file for both reading and writing. (The file must exist.)
“w+“	Opens the file as an empty file for both reading and writing. If the file exists, its contents are destroyed.
“a+“	Opens the file for reading and appending; the appending operation includes the removal of the EOF marker before new data is written to the file and the EOF marker is restored after writing is complete; creates the file first if it doesn't exist.
“b”	Binary mode to open the file

File open

“r”: 읽기 모드로 파일을 연다.

“w”: 쓰기 모드로 파일을 생성한다. 만약 파일이 존재하지 않으면 파일이 생성된다. 파일이 이미 존재하면 기존의 내용이 지워진다.

“a“: 추가 모드로 파일을 연다. 만약 똑같은 이름의 기존의 파일이 있으면 데이터가 파일의 끝에 추가된다. 파일이 없으면 새로운 파일을 만든다.

“r+”: 읽기와 쓰기 모드로 파일을 연다. 파일이 반드시 존재하여야 한다.

“w+”: 읽기와 쓰기 모드로 파일을 생성한다. 만약 파일이 존재하지 않으면 파일이 생성된다. 파일이 존재하면 새 데이터가 기존 파일의 데이터를 덮어 쓰게 된다.

“a+”: 읽기와 추가 모드로 파일을 연다. 만약 똑같은 이름의 기존의 파일이 있으면 데이터가 파일의 끝에 추가된다. 읽기는 어떤 위치에서나 가능하다. 파일이 없으면 새로운 파일을 만든다.

“b”: 이진 파일 모드로 파일을 연다.

Ex #5: file open

```c
// file open
#include <stdio.h>

int main(void)
{
		FILE *fp = NULL;

		fp = fopen("sample.txt", "w");

		if( fp == NULL )
				printf(“fail\n");
		else
				printf(“success\n");

		fclose(fp);

		return 0;
}
```

```
success
```

File close and remove

- File closing function

```
int fclose( FILE *stream );
```

- File removal function

```
int remove(const char *path)
```

```
#include <stdio.h>

int main( void )
{
        if( remove( "sample.txt" ) == -1 )
                printf( “can not remove the sample.txt.\n" );
        else
                printf( “remove the sample.txt.\n" );

        return 0;
}
```

File output

```
int fprintf( FILE *fp, const char *format, ...);
```

```
int i = 23;
float f = 1.2345;
FILE *fp;

fp = fopen("sample.txt", "w");

if( fp != NULL )
        fprintf(fp, "%10d %16.3f", i, f);

fclose(fp);
```

File input

```
int fscanf( FILE *fp,  const char *format, ...);
```

```
int i;
float f;
FILE *fp;

fp = fopen("sample.txt", "r");

if( fp != NULL )
        fscanf(fp, "%d %f", &i, &f);

fclose(fp):
```

EX #6: score

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        FILE *fp;
        char fname[100];
        int number, count = 0;
        char name[20];
        float score, total = 0.0;

        printf(“enter the socre file name:  ");
        scanf("%s", fname);

        // Open the socre.txt file with write mode.
        if( (fp = fopen(fname, "w")) == NULL )
        {
                fprintf(stderr," can not open the %s file.\n ", fname);
                exit(1);
        }
```

EX #6: score (cont.)

```
    // store the sid, name, score from the user input.
    while( 1 )
    {
        printf(“enter the sid, name, score:");
        scanf("%d", &number);
        if( number < 0 ) break;
        scanf("%s %f", name, &score);
        fprintf(fp, "%d %s %f\n", number, name, score);
    }
    fclose(fp);
    // open the score file with read mode.
    if( (fp = fopen(fname, "r")) == NULL )
    {
        fprintf(stderr,“can not open the %s file.\n", fname);
        exit(1);
    }
    // read the score information and compute the average score.
    while( 1)
    {
        fscanf(fp, "%d %s %f", &number, name, &score);
        if(feof( fp )) break;
        total += score;
        count++;
    }
    printf(“average = %f\n", total/count);
    fclose(fp);
    return 0;
}
```

Project #2: E-Library

- Design an electronic library system.

```
struct  book{
   int  id;         // book id
   char  name[30];  // book name. end with 0 to make a string
};

book  books[10];   // we have 10 books

struct  member{
   int  id;        // member id
   char  name[30];  // member name
};

member  members[5];   // 5 members
```

Project #2: E-Library (Cont.)

- menu:
  - ✓ quit, book show, book modify, book remove, add a book , member show, member modify, member remove, add a member, borrow, return.
  - ✓ Store the book and member information to the “book.txt” and “member.txt” file
  - ✓ Store the borrow and return information to the “manage.txt” file.