Introduction to Software Design

C08. Dynamic Allocation

Yoonsang Lee
Spring 2020

Alternative Assignment to Final Exam

- Time: 6/20(Sat) 10 am ~ 6/21(Sun) 10 am

- Please schedule your time in advance.

Alternative Assignment to Final Exam

- Programming assignment

- We will copy-check thoroughly. Do not try to cheat.
  - If A copies B's code, A and B will get 0 point.
  - **If A, B, C copies the same code from the internet**, they will all get 0 point.

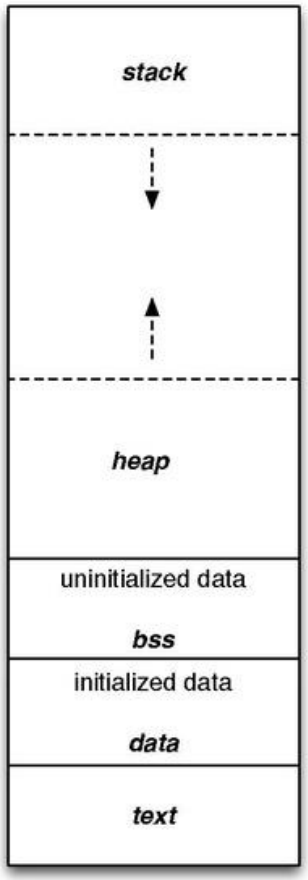
- Performing the assignment by proxy (대리 과제) is strictly prohibited.
  - If you hear about a proxy assignment case, please email me.

Topic Covered

- 프로그램이 사용하는 메모리 영역의 구성
- 프로그램 실행 시 메모리의 상태 변화
- 프로그램 실행 중에 데이터의 저장공간을 확보하는 방법
- -> **동적 할당**
- 구조체 변수의 동적 할당
- 배열의 동적 할당
- 구조체 배열의 동적 할당

Typical Memory Layout of C / C++ Programs

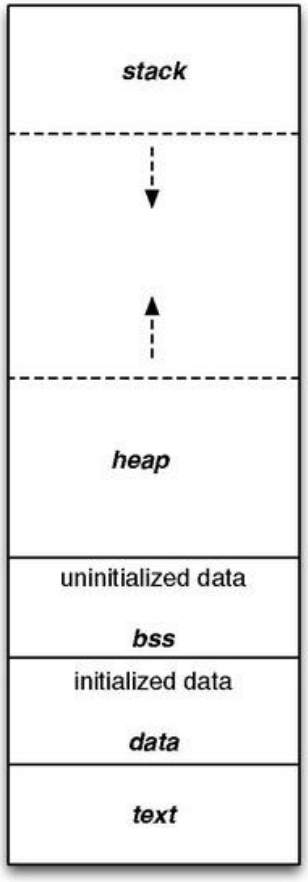
- When you run a C / C++ program, OS allocates memory space for the program like this:



Organized in several segments:

- Stack segment
- Heap segments
- BSS segments
- Data segments
- Text segments

Typical Memory Layout of C / C++ Programs



: local variables, function parameters, return address

: for dynamic memory allocation

(data segment)

: global variables, static variables

(code segment)

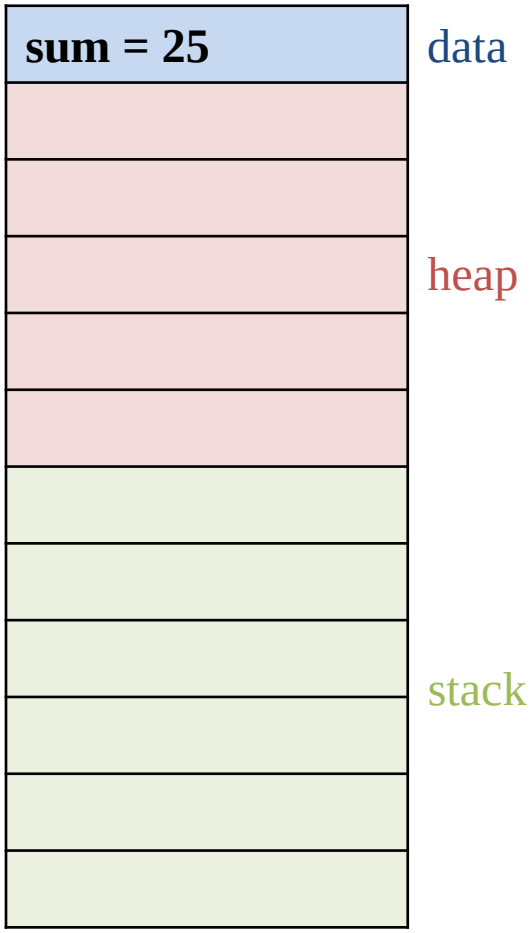
: compiled binary code of the program

- The reason of "typical" is, it's actually platform / implementation dependent (not a part of C/C++ specifications), but it generally used in most popular platforms.

Example - Memory Layout 1

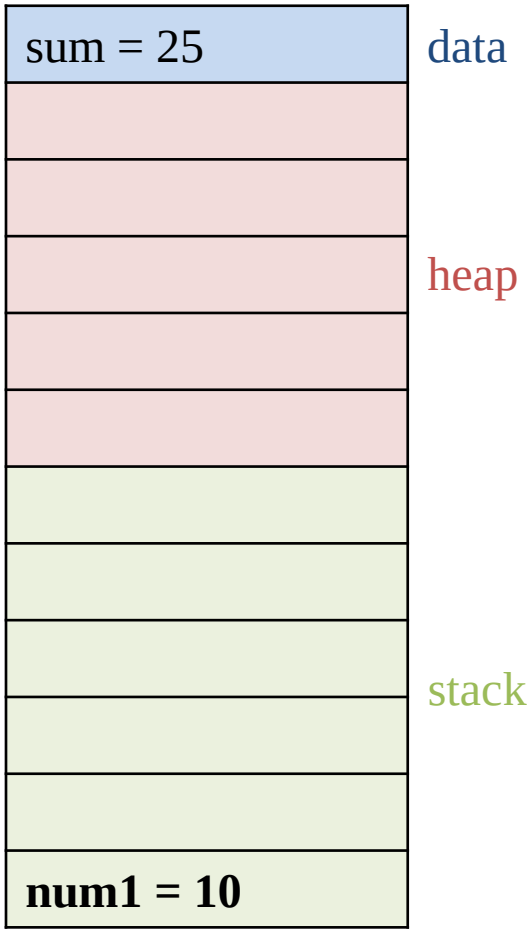
(Program starts)

```
int sum=25;

int main()
{
   int num1=10;
   func(num1);
   num1++;
   func(num1);
   return 0;
}

void func(int n)
{
   int num2=20;
}
```



Example - Memory Layout 2

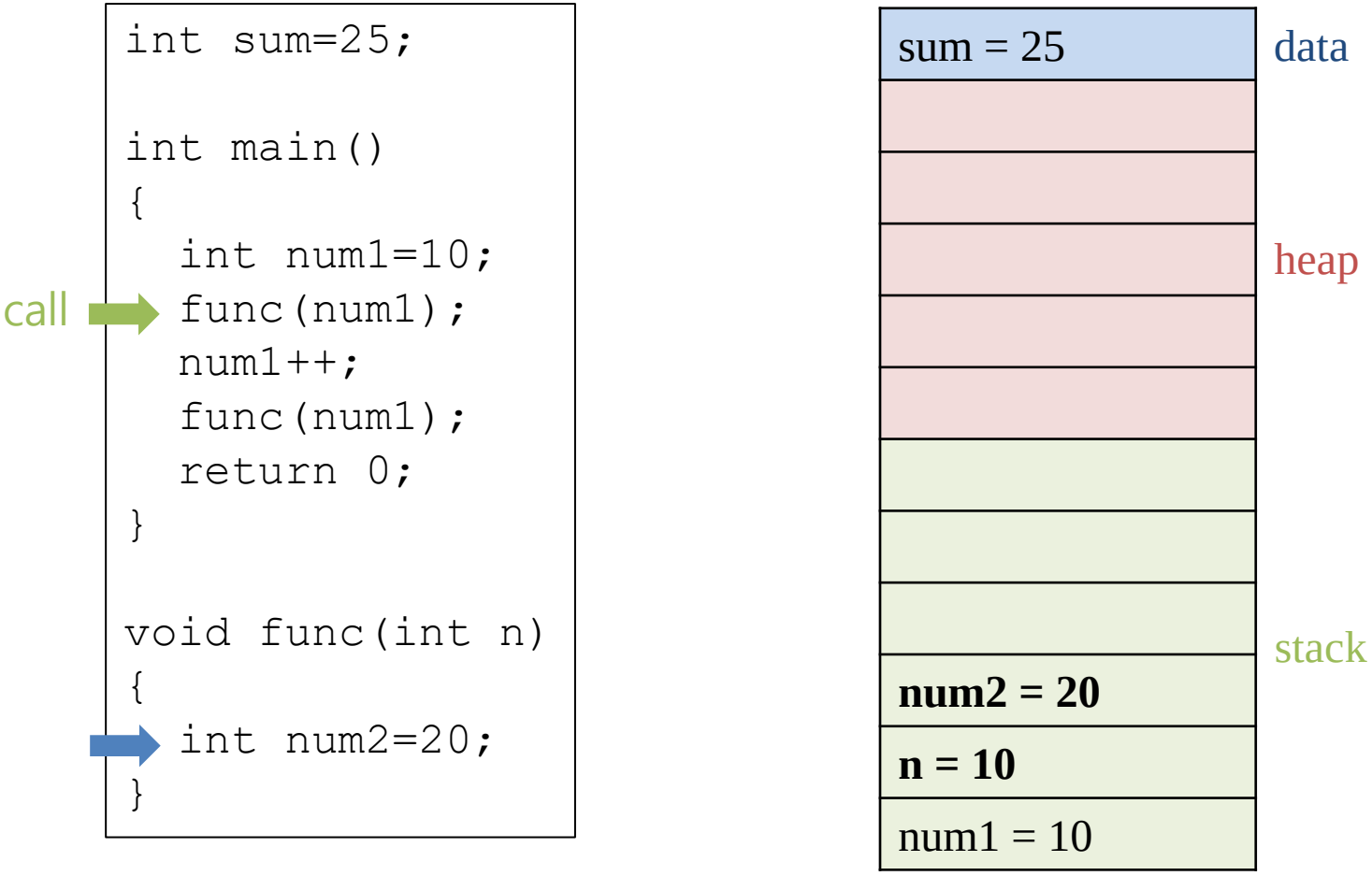
```
int sum=25;

int main()
{
  int num1=10;
  func(num1);
  num1++;
  func(num1);
  return 0;
}

void func(int n)
{
  int num2=20;
}
```



Example - Memory Layout 3

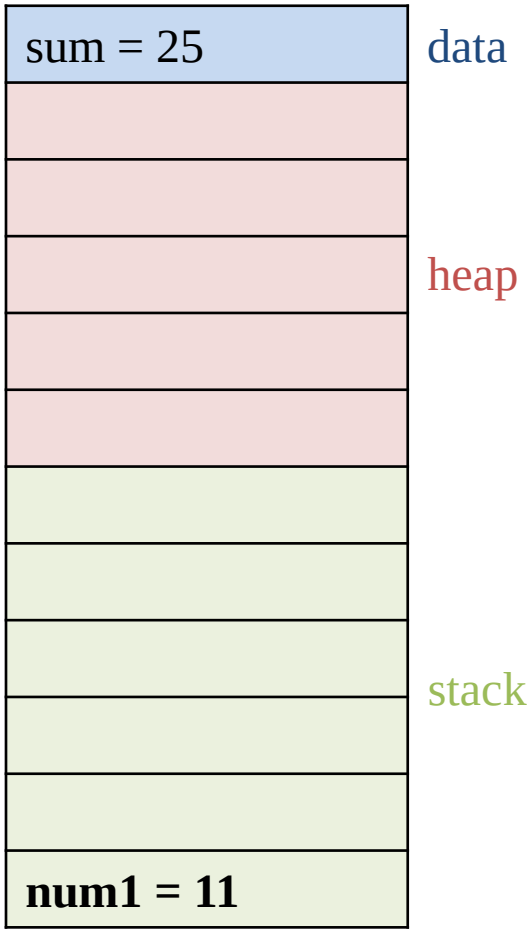
```
int sum=25;

int main()
{
   int num1=10;
   func(num1);
   num1++;
   func(num1);
   return 0;
}

void func(int n)
{
   int num2=20;
}
```

call



Example - Memory Layout 4

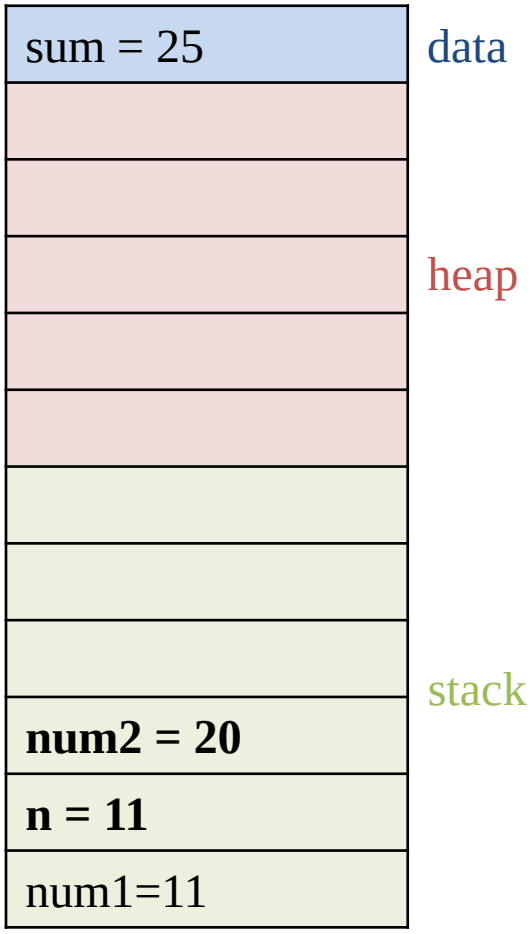
```
int sum=25;

int main()
{
   int num1=10;
   func(num1);
   num1++;
   func(num1);
   return 0;
}

void func(int n)
{
   int num2=20;
}
```



Example - Memory Layout 5

```
int sum=25;

int main()
{
   int num1=10;
   func(num1);
   num1++;
   func(num1);
   return 0;
}

void func(int n)
{
   int num2=20;
}
```

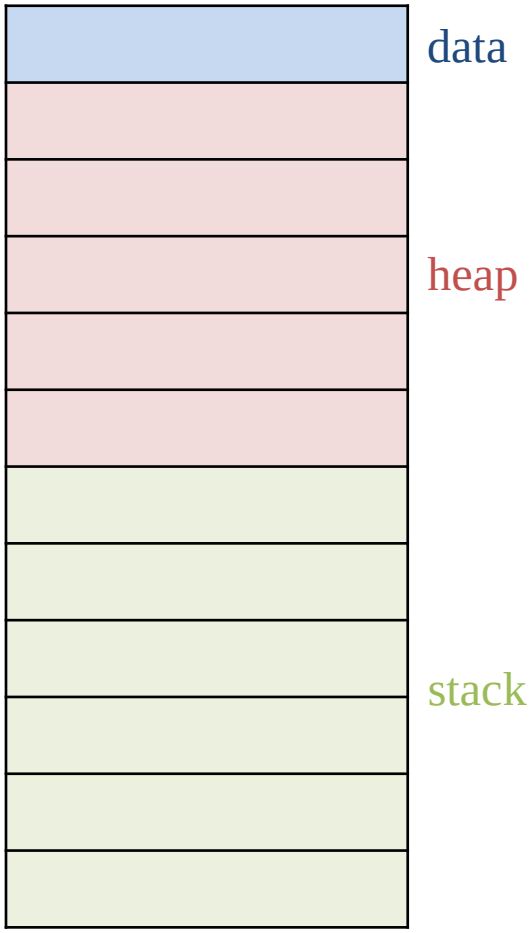
call



Example - Memory Layout 6

(Program ends)

```
int sum=25;

int main()
{
   int num1=10;
   func(num1);
   num1++;
   func(num1);
   return 0;
}

void func(int n)
{
   int num2=20;
}
```



‘스택(Stack)’의 의미

- 쌓아올린 더미. 접시를 쌓아 올린 모양을 생각해보자.

- **가장 나중에 쌓은 접시**가 가장 먼저 내려진다.

- **가장 나중에 스택 영역에 할당된 지역 변수**가 가장 먼저 소멸된다.

각 영역에 들어갈 데이터의 크기는 어떻게 알 수 있을까?

- Text segment : 프로그램 코드의 크기
- BSS/Data segment: 전역 변수, static 변수의 크기
- Stack segment : 각 함수별 지역변수 크기
  - (전체 stack size는 compiler optio으로 지정 가능)

- → 컴파일 시에 알 수 있다.

컴파일 시에 필요한 데이터의 크기를 알 수 없는 경우는?

- 만일 사용자로부터 scanf로 숫자를 입력 받아 해당 개수의 Point형 변수를 만들고 싶다면?

- **프로그램 코드를 작성할 때 처리할 데이터의 종류나 수를 미리 결정할 수 없다면?**

- → 실행 도중에 필요한 메모리 공간을 **동적으로** 확보해야 한다.

- → **동적 할당. 힙(heap) 영역**에 저장.

힙 영역의 메모리 공간 할당과 해제 : malloc(), free() 함수

```c
#include <stdlib.h>

void* malloc(size_t size);   // heap에 메모리 공간 할당

void free(void* ptr);        // heap에 할당된 메모리 공간 해제
```

```c
#include <stdlib.h>

int main()
{
    void* ptr1 = malloc(4);      // heap에 4바이트 할당
    void* ptr2 = malloc(12);     // heap에 12바이트 할당
    ...
    free(ptr1);      // ptr1이 가리키는 4바이트 해제
    free(ptr2);      // ptr2가 가리키는 12바이트 해제
    ...
}
```

void형 포인터 (void*)

```
void* ptr;
```

- void 포인터 : 형(type) 정보가 없는 포인터.
- 어떤 종류의 주소 값도 저장이 가능
- *, ->, + 같은 포인터 연산 불가능

```
int n = 10;
double d = 2.1;
void* ptr;

ptr = &n;
ptr = &d;
```

이렇게 하는 것이 가능

```
int main(void)
{
    int num=20;
    void * ptr=&num;
    *ptr=20;    // 컴파일 에러!
    ptr++;    // 컴파일 에러!
    . . . .
}
```

형 정보가 존재하지 않으므로!!

malloc 함수의 반환형이 void형 포인터 인 이유

```
// malloc 함수의 호출 예
void* ptr1 = malloc(sizeof(int));
void* ptr2 = malloc(sizeof(double));
void* ptr3 = malloc(sizeof(int)*7);
void* ptr4 = malloc(sizeof(double)*9);
```

```
// sizeof 연산 후 실행되는 코드
void* ptr1 = malloc(4);
void* ptr2 = malloc(8);
void* ptr3 = malloc(28);
void* ptr4 = malloc(72);
```

- malloc 함수는 지정된 크기의 메모리 공간을 할당할 뿐, 어떻게 쓰일지는 알지 못한다.
- 리턴된 포인터를 제대로 쓰려면 아래와 같이 **형 변환**을 해야한다.

```
// malloc 함수의 가장 모범적인 호출 예
int* ptr1 = (int*)malloc(sizeof(int));
double* ptr2 = (double*)malloc(sizeof(double));
int* ptr3 = (int*)malloc(sizeof(int)*7);
double* ptr4 = (double*)malloc(sizeof(double)*9);
```

C Example

```c
#include <stdio.h>
#include <stdlib.h> // malloc, free 함수 사용을 위한 헤더 파일
int main()
{
    int* pi;
    double* pd;
    pi = (int*)malloc(sizeof(int));
    pd = (double*)malloc(sizeof(double));

    // 할당된 메모리에 값을 쓴 적이 없으므로 쓰레기 값이 출력됨
    printf("%d %f\n", *pi, *pd);

    // 할당된 메모리 공간에 값을 대입
    *pi = 10;
    *pd = 3.4;
    // 아래처럼 scanf로 할당된 메모리 공간에 값을 채울 수도 있다.
    scanf("%d %lf", pi, pd);

    // 새로 대입한 값 출력
    printf("%d %f\n", *pi, *pd);

    // malloc()으로 할당된 메모리의 사용이 끝나면 반드시
    // free()를 호출해서 해제해야 한다.
    free(pi);
    free(pd);
    return 0;
}
```

- Python에서는 모든 변수(혹은 객체)가 동적으로 할당된다고 보면 된다.
- 동적으로 할당된 객체의 해제는 Python이 garbage collector(GC)를 통해 알아서 수행한다 (프로그래머가 신경쓸 필요가 없다)

힙 영역으로의 접근 방법

- 힙 영역(동적으로 할당된 메모리 공간)은 **오직 포인터로만 접근이 가능함!**

```
pi = (int*)malloc(sizeof(int));
*pi = 10;
```

- 우리가 포인터를 열심히 배웠던 이유 중 하나.

free()를 호출하지 않으면?

- → 할당된 메모리 공간은 **시스템의 메모리라는 중요한 자원을 계속 차지**하고 있게 된다.

```
void test()
{
    int* pi = (int*)malloc(sizeof(int));
    *pi = 10;
}
// 이 함수의 호출이 끝나면 지역변수 pi는 소멸되지만,
malloc()에 의해 할당된 4바이트는 그대로 남아있게 된다
(pi가 소멸되면 해당 4바이트에 접근할 수 있는 방법이
없어지므로 다른 용도로 사용 불가).
```

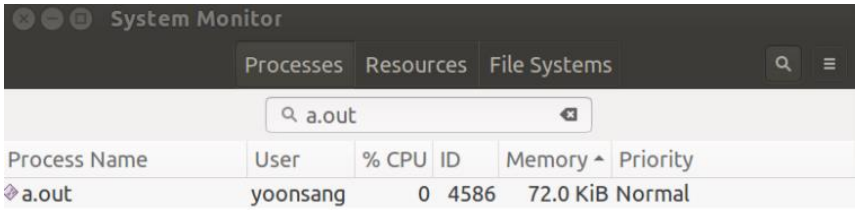
- 이를 **메모리 누수(memory leak)**이라고 부른다.

C Example

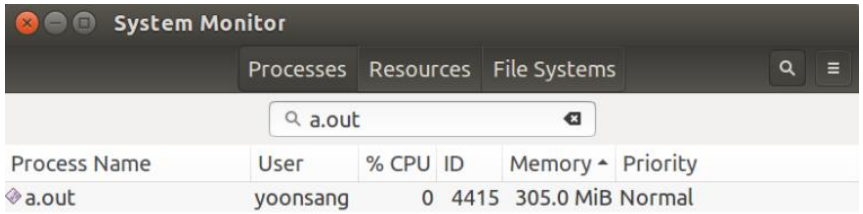
```c
#include <stdio.h>
#include <stdlib.h>

void test()
{
    int* pi = (int*)malloc(4);
    *pi = 10;
    //free(pi);
}

int main()
{
    int i, n;
    for(i=0; i<10000000; i++)
        test();

    // 사용자의 입력을 기다리느라 잠시 멈춘 사이에
    // Ubuntu system monitor를 실행하자 (Windows에서는 작업관리자)
    scanf("%d", &n);

    return 0;
}
```

C Example

free()를 호출한 경우



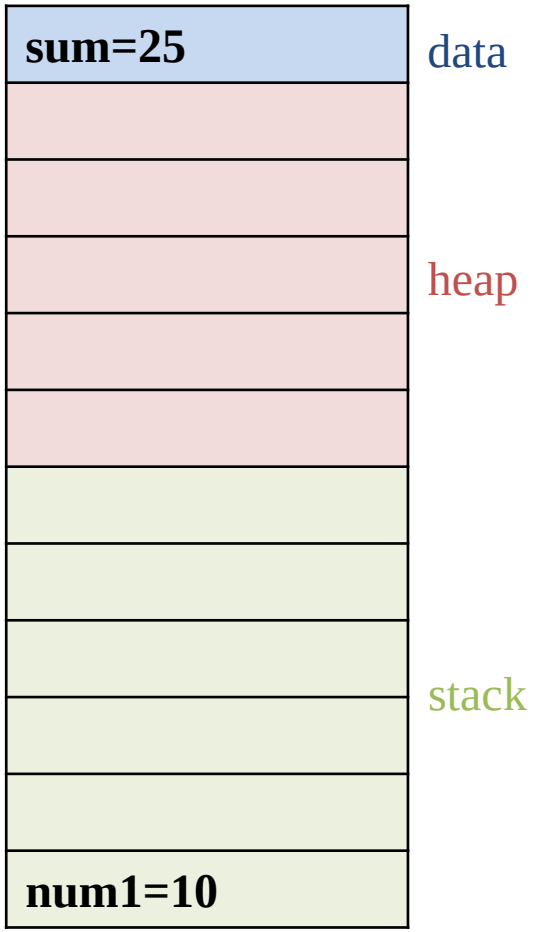
free()를 호출하지 않은 경우



(참고) 실습코드의 내용상 이론적으로는 4*10000000 = 약40MB 정도의 메모리를 힙 영역에 할당해야 하지만, 실제로는 관련 정보를 기록하기 위한 추가적인 공간을 더 할당한다. 그리고 위의 수치는 힙 이외의 다른 영역 및 기타 필요한 요소들이 포함된 수치이다.

Memory Layout

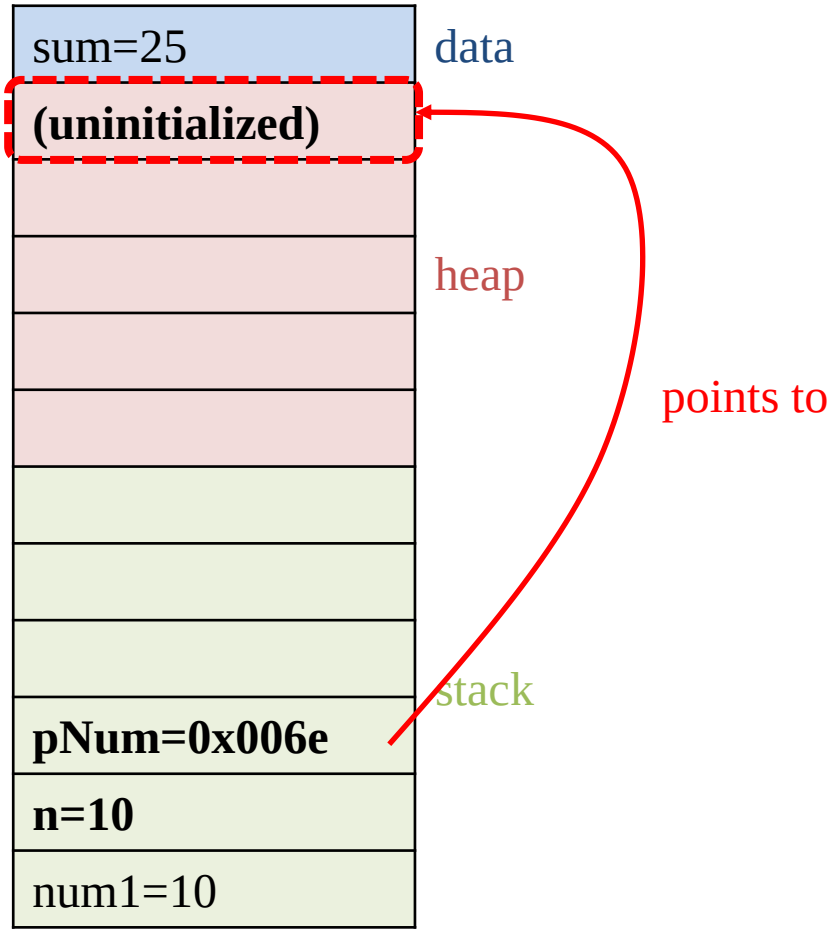
```
int sum=25;

int main(void)
{
  int num1=10;
  fct(num1);
  num1++;
  fct(num1);
  return 0;
}

void fct(int n)
{
  int* pNum = (int*)malloc(4);
  *pNum = n;
  free(pNum);
}
```



Memory Layout

```
int sum=25;

int main(void)
{
   int num1=10;
   fct(num1);
   num1++;
   fct(num1);
   return 0;
}

void fct(int n)
{
   int* pNum = (int*)malloc(4);
   *pNum = n;
   free(pNum);
}
```

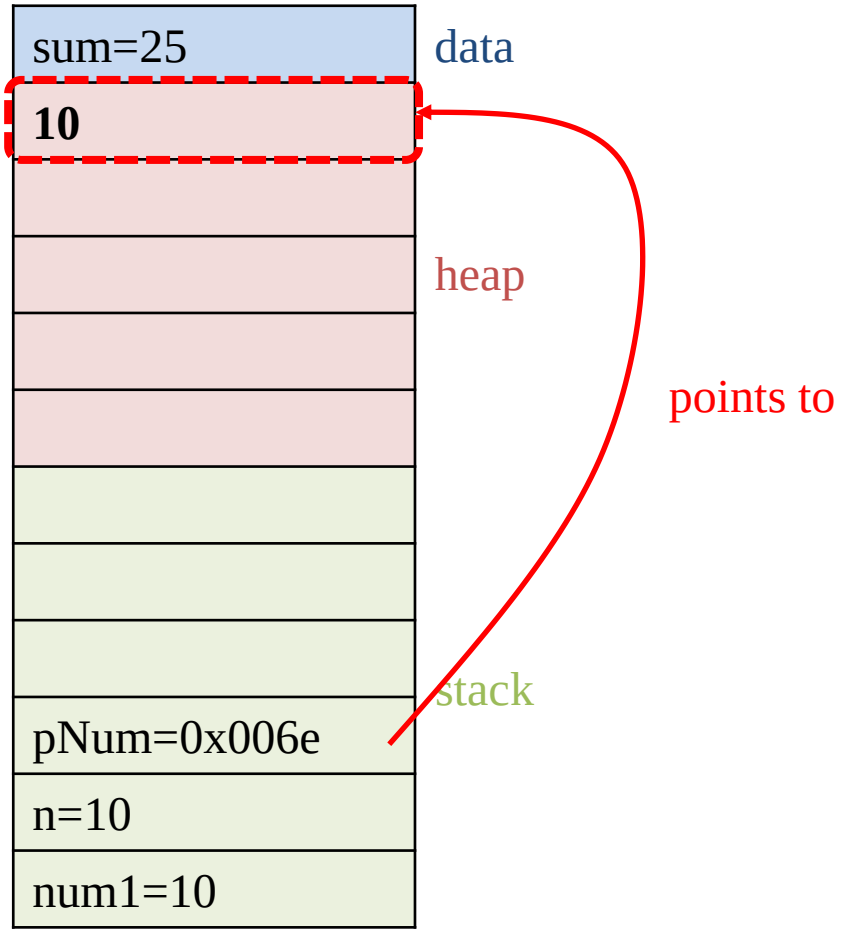
call



Memory Layout

```
int sum=25;

int main(void)
{
   int num1=10;
   fct(num1);
   num1++;
   fct(num1);
   return 0;
}

void fct(int n)
{
   int* pNum = (int*)malloc(4);
   *pNum = n;
   free(pNum);
}
```

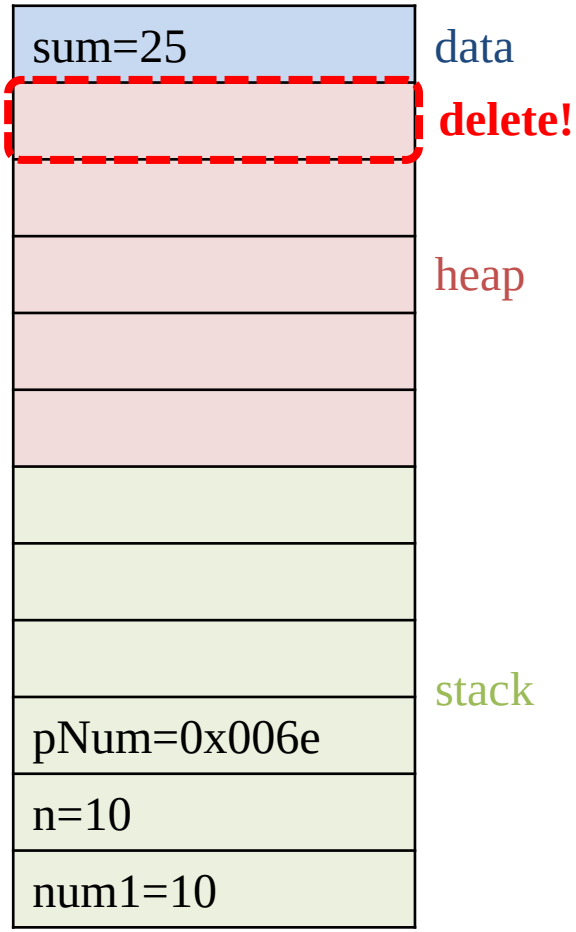
call



Memory Layout

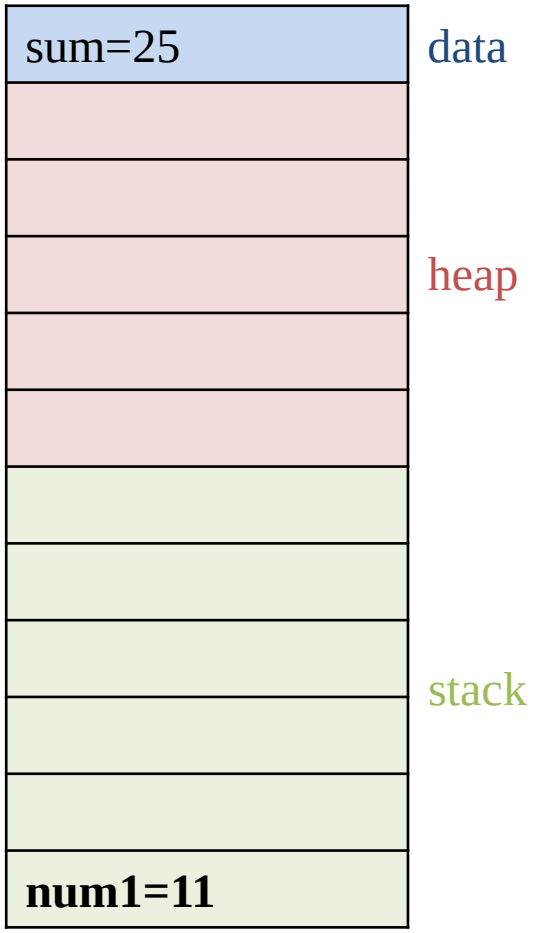
```
int sum=25;

int main(void)
{
   int num1=10;
   fct(num1);
   num1++;
   fct(num1);
   return 0;
}

void fct(int n)
{
   int* pNum = (int*)malloc(4);
   *pNum = n;
   free(pNum);
}
```

call



Memory Layout

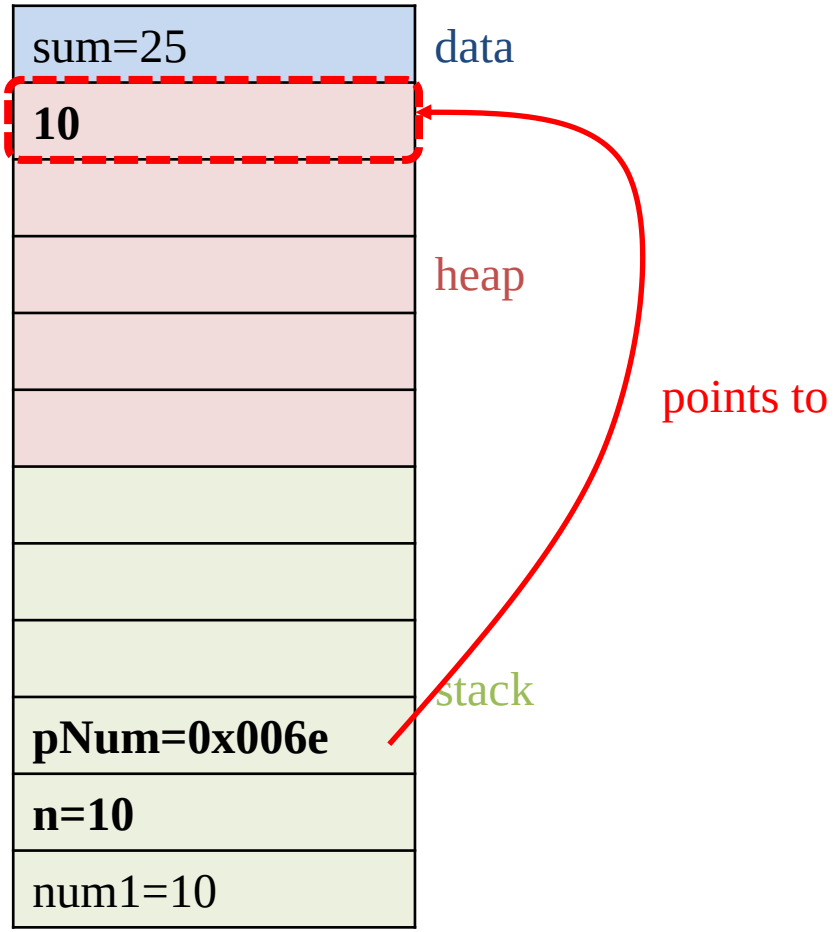
```
int sum=25;

int main(void)
{
  int num1=10;
  fct(num1);
  num1++;
  fct(num1);
  return 0;
}

void fct(int n)
{
  int* pNum = (int*)malloc(4);
  *pNum = n;
  free(pNum);
}
```



When a memory leak occurs

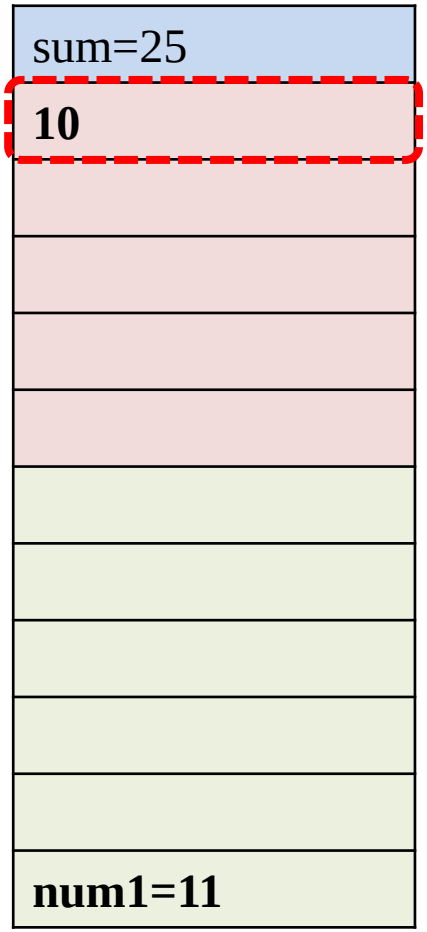
```
int sum=25;

int main(void)
{
   int num1=10;
   fct(num1);
   num1++;
   fct(num1);
   return 0;
}

void fct(int n)
{
   int* pNum = (int*)malloc(4);
   *pNum = n;
   //free(pNum);
}
```

call



When a memory leak occurs

```
int sum=25;

int main(void)
{
   int num1=10;
   fct(num1);
   num1++;
   fct(num1);
   return 0;
}

void fct(int n)
{
   int* pNum = (int*)malloc(4);
   *pNum = n;
   //free(pNum);
}
```



Quiz #1

- Go to https://www.slido.com/
- Join #isd-hyu
- Click “Poll”

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked as “attendance”.

free()에 대해…

- free()하지 않은 메모리는 영원히 할당된 채로 남아 있나요?
- -> 아니다. 해당 프로그램이 종료하면 운영체제로 반환된다.

- 그렇다면 작은 메모리 공간은 굳이 free()하지 않아도 괜찮지 않나요?
- -> 아니다. 만일 365일, 24시간 돌아가야 하는 서버 프로그램이라면?
- 지속적으로 메모리 누수가 발생되어 결국 시스템이 다운될 것.

명심할 것 : malloc()과 free()는 항상 쌍(pair)으로 사용!

```
int* pi = (int*)malloc(4);

// 필요한 작업들을 수행…

free(pi);
```

- malloc()이 한 번 등장하면 반드시 free()도 있어야 한다!

- malloc()과 free()를 서로 다른 함수에서 호출하면 안 된다!
  - free()를 호출하지 않는 실수를 할 가능성이 높아짐.

malloc()의 리턴값

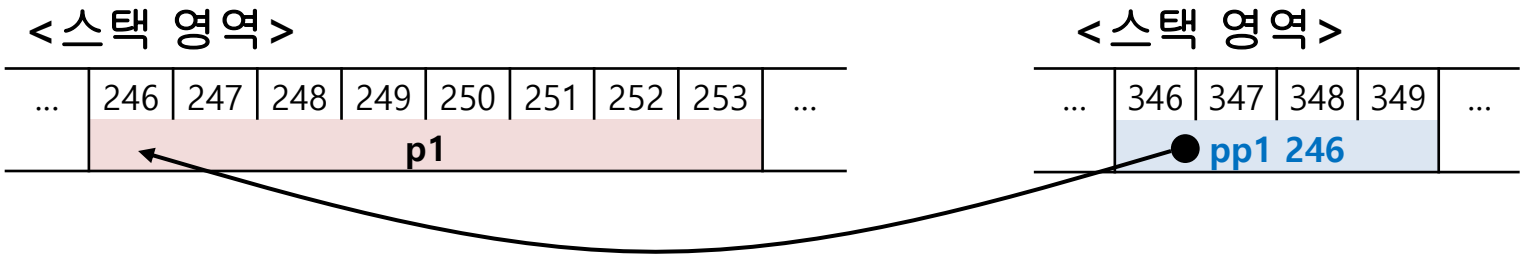
- malloc() 함수는 성공 시 할당된 메모리의 주소 값, 실패 시 NULL 을 리턴한다.
- 원칙적으로는 malloc()함수의 리턴값이 NULL인지 항상 확인해야 한다.
  - 만일 NULL이라면 적절한 예외 처리 코드가 필요
- 실제로는 메모리 사용량이 아주 크지 않은 이상 메모리 할당에 실패하는 경우는 거의 없다.
- 하지만 대용량 데이터를 다루는 프로그램에서는 당연히 실패하는 경우에 대한 예외 처리 코드가 필요하다.

구조체 변수의 동적 할당

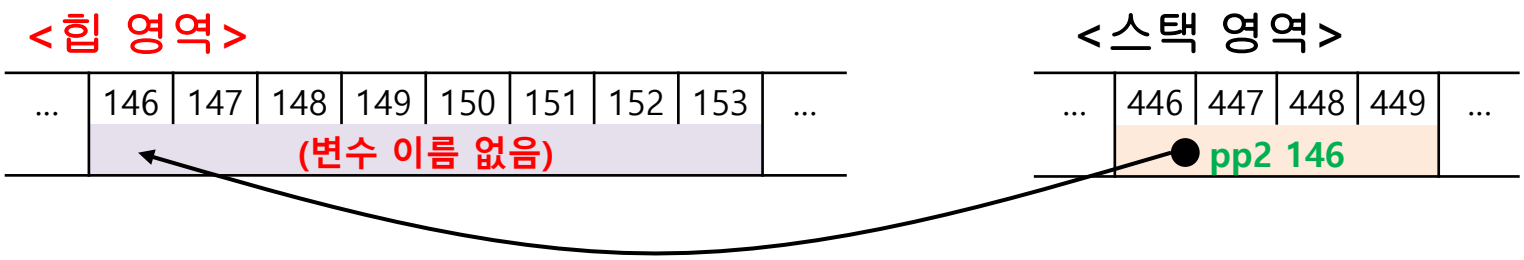
```
Point p1;
Point* pp1 = &p1;
```

(32bit system
이라고 가정)

```
typedef struct
{
    int xpos;
    int ypos;
}Point;
```




```
Point* pp2 = (Point*)malloc(sizeof(Point));
```



구조체 변수의 동적 할당

```
Point p1;
Point* pp1 = &p1;
```

```
Point* pp2 = (Point*)malloc(sizeof(Point));
```

- pp1은 스택에 할당된 8바이트 공간(지역변수 p1)을 가리키는 포인터
- pp2는 힙에 할당된 8바이트 공간(malloc으로 할당)을 가리키는 포인터
- pp1, pp2 모두 **하나의 Point형 변수를 저장할 수 있는 공간**을 가리키는 것은 마찬가지이다!

C Example

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

typedef struct
{
    int xpos;
    int ypos;
} Point;

int main()
{
    Point p1;
    Point* pp1 = &p1;

    Point* pp2 = (Point*)malloc(sizeof(Point));

    pp1->xpos = 1;
    pp1->ypos = 2;
    printf("%d %d\n", pp1->xpos, pp1->ypos);

    pp2->xpos = 1;
    pp2->ypos = 2;
    printf("%d %d\n", pp2->xpos, pp2->ypos);

    free(pp2);

    return 0;
}
```

Quiz #2

- Go to https://www.slido.com/
- Join #isd-hyu
- Click “Poll”

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked as “attendance”.

배열의 동적 할당

- 사용자가 scanf로 입력한 길이만큼의 배열을 만들고 싶다면?

- **int형 데이터 5개**를 저장하는 배열을 동적 할당
- -> **int*** pi = **(int*)**malloc(**5*sizeof(int)**)

- 배열을 동적 할당하는 함수가 따로 있는 것이 아니라, **malloc()으로 배열의 크기만큼 메모리를 확보**하고 **리턴된 포인터 pi를 통해 배열을 다루듯이 코딩**하면 된다.

C Example

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

int main()
{
    int n, i;
    int* pi;

    // 배열의 길이를 입력받음
    scanf("%d", &n);

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

    for(i=0; i<n; ++i)
        // 배열의 길이만큼 정수를 입력 받음
        // 아래 둘 중 어느 것을 해도 무방
        scanf("%d", &pi[i]);
        //scanf("%d", pi+i);

    for(i=0; i<n; ++i)
        // 배열의 각 요소를 출력
        // 아래 둘 중 어느 것을 해도 무방
        printf("%d ", pi[i]);
        //printf("%d ", *(pi+i));

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

구조체 배열의 동적할당

- 앞의 배열의 동적 할당과 같은 방법

- **Point형 데이터 5개**를 저장하는 배열을 동적 할당
- -> **Point*** pi = **(Point*)**malloc(**5*sizeof(Point)**)

- 동적 할당한 배열의 각 요소(pi[i])의 각 멤버에 접근하여 읽거나 쓰기를 하면 됨.

C Example

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

typedef struct
{
    int xpos;
    int ypos;
} Point;

int main()
{
    // 배열의 길이를 입력받음
    int n;
    scanf("%d", &n);

    Point* pi = (Point*)malloc(n * sizeof(Point));

    for (int i = 0; i<n; ++i)
        // 배열의 길이만큼 정수를 입력 받음
        // 아래 둘 중 어느 것을 해도 무방
        scanf("%d %d", &pi[i].xpos, &pi[i].ypos);
        //scanf("%d %d", &(pi+i)->xpos, &(pi+i)->ypos);

    for (int i = 0; i<n; ++i)
        // 배열의 각 요소를 출력
        // 아래 둘 중 어느 것을 해도 무방
        printf("%d %d\n", pi[i].xpos, pi[i].ypos);
        //printf("%d %d\n", (pi+i)->xpos, (pi+i)->ypos);

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

Quiz #3

- Go to https://www.slido.com/
- Join #isd-hyu
- Click “Poll”

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked as “attendance”.

Course Wrap up

한 학기 동안 다룬 주제들

P01. Hello World
P02. Guess the Number, Jokes
P03. Dragon Realm
P04. Hangman
C01. C Basics
C02. Data Representation
C03. Functions
C04. Array, Pointer
C05. String Functions, Text Games in C, Const
C06. Parameter Passing, Const Pointer & String, Struct
C07. More about Array & Pointer, Preprocessor
C08. Dynamic Allocation

Python & C

- 하나의 주제를 여러가지 방법으로 다뤄보는 것은 그 주제를 보다 깊이 있게 이해할 수 있게 해줌.

- “프로그래밍”을 두 가지 “프로그래밍 언어” (Python, C)를 이용해 해봄

- 이를 통해:
  - Programming language의 구성 요소
  - Program의 작동 방식
  - Programming, Coding
- 에 대해 보다 깊이 있게 이해하는 기회가 되었기를 바람

학기를 마치며

- 바라는 점 딱 한 가지:
- **재미있게** 프로그래밍을 하면 좋겠습니다.

- 지금까지는 시간에 쫓기며 과제 문제 풀이를 위한 프로그래밍을 해왔겠지만, 앞으로는 스스로 만들어 보고 싶은 프로그램을 작성해보면서, 진정한 프로그래밍의 재미를 느낄 수 있기를 바랍니다.
  - 이를 위해서 별도의 라이브러리(library) 혹은 패키지(package)를 사용하는 방법을 꼭 익히기 바랍니다.

- 예: 간단한 게임 작성
  - C: SDL library
    - http://lazyfoo.net/tutorials/SDL/index.php
  - Python: PySDL2 package
    - https://pysdl2.readthedocs.io/en/rel_0_9_6/tutorial/index.html

이번 주 실습

- 6/11일 목요일에만 실습이 있습니다.

Thanks for being a great class!