Chapter 5) Pointers and Strings

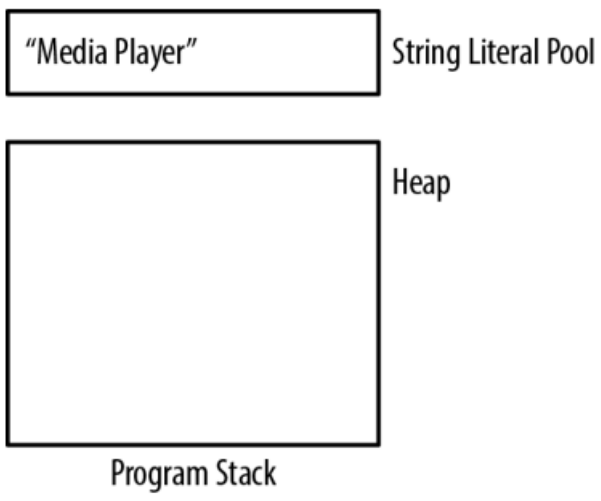
정의 : string은 '\0'으로 끝나는 char의 배열을 의미한다.

📌**주의**

- NULL과 '\0'은 다르다 NULL은 ((void*)0)을 의미하고, '\0'은 아스키코드값이 0인 char문자이다.

String Declaration

Case1) Literal string



- 리터럴 문자열은 `Data` 영역에 `read-only-segment`에 할당된다.
- literals는 literal pool에 선언된다.
- 리터럴 String은 `read-only memory`에 allocated된다. (immutable함)
- global, static, local모두 상관이 없다 (Scope가 없음)

⭐**주의**

- literal string은 몇몇의 compiler에 의해 변경이 가능한 경우가 있다. (권장되지는 않음)
- 저런 행위를 방지하기 위해서는 `const` 키워드를 사용한다.

```
char *tabHeader = "Sound";
*tabHeader = 'L';
printf("%s\n", tabHeader); // Display "Lound"
```

```
const char *tabHeader = "Sound";
```

Case2) arrays of char

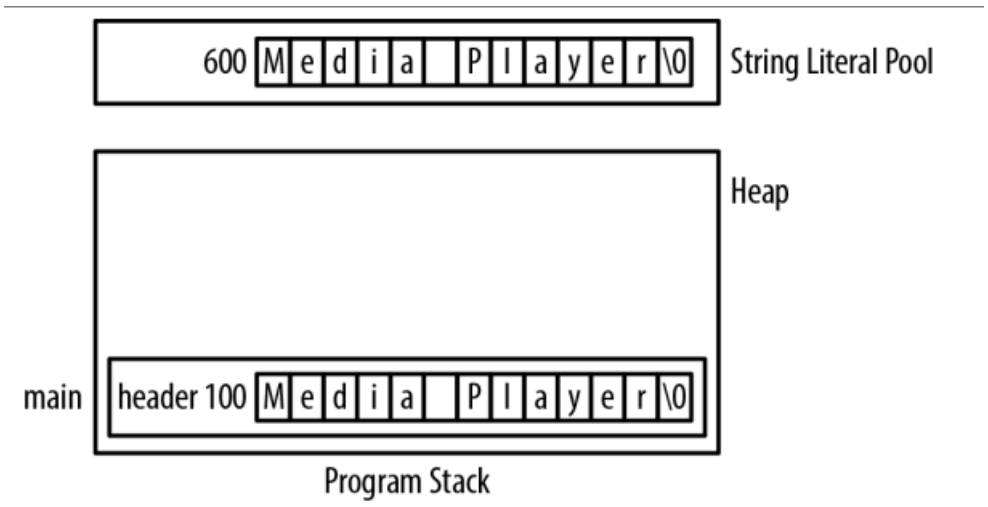
How to Initialize?

1. 초기화 연산자 = 사용
    - 컴파일 시점에 정해짐
    - 리터럴 "Media player"의 복사본으로 초기화가 됨.

```
char header[] = "Media player";
```

2. 선언 후 strcpy를 사용
    - 런타임 시점에 정해짐
    - 문자열 리터럴 "Media player"가 header배열로 복사됨.

```
char header[13];
strcpy(header, "Media Player");
```



📌주의 🙊

```
char header2[];
header2 = "Media Player"; 안됨!!
```

Case3) pointer

1) Dynamic allocation을 사용하는 방법

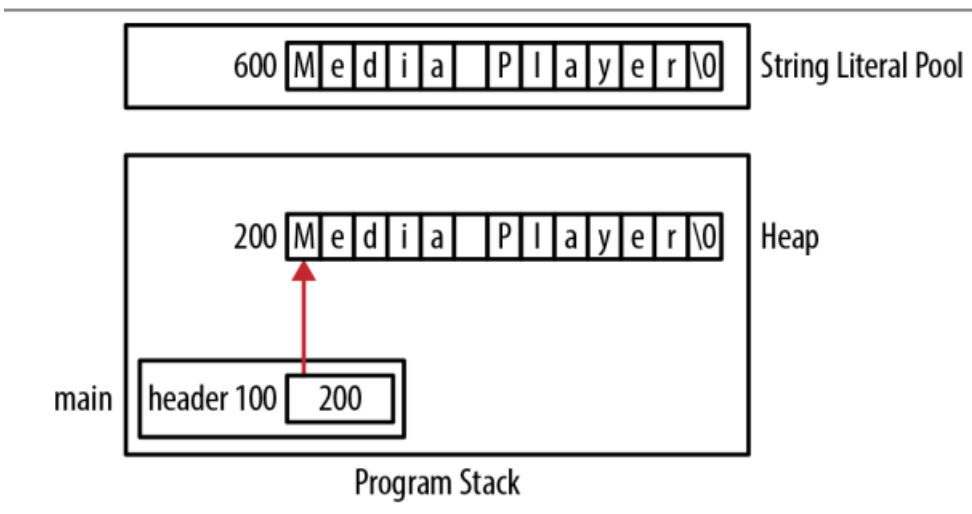
```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char *header = (char*)malloc(sizeof(char) * strlen("Media Player") + 1);
    strcpy(header, "Media Player");
    return (0);
}
```

질문 : literal 문자열이 같은 것이 2개가 있는데, 같은 메모리에 있는가??

리터럴 문자열의 메모리 저장

1. **리터럴 문자열의 저장**:
   - C 프로그램에서 리터럴 문자열은 **읽기 전용 데이터 세그먼트**(read-only data segment)에 저장됩니다. 이 영역은 문자열 리터럴을 수정할 수 없는 메모리 영역으로, 프로그램이 실행되는 동안 동일한 리터럴 문자열이 메모리에서 하나만 유지됩니다.
2. **리터럴 문자열의 중복 저장**:
   - 문자열 리터럴 `"Media Player"`는 프로그램의 메모리에 저장될 때, 하나의 메모리 위치에 저장됩니다. 이는 동일한 문자열 리터럴이 여러 번 사용되더라도 메모리에서 중복 저장되지 않음을 의미합니다.
3. **리터럴 문자열의 참조**:
   - 문자열 리터럴이 여러 번 사용되더라도, 각 사용 위치는 동일한 메모리 주소를 참조합니다. 예를 들어, `strlen("Media Player")`와 `strcpy("Media Player")`에서 사용된 `"Media Player"`는 메모리 내에서 동일한 위치를 가리킵니다. 두 호출 모두 문자열 리터럴의 동일한 메모리 주소를 참조합니다.

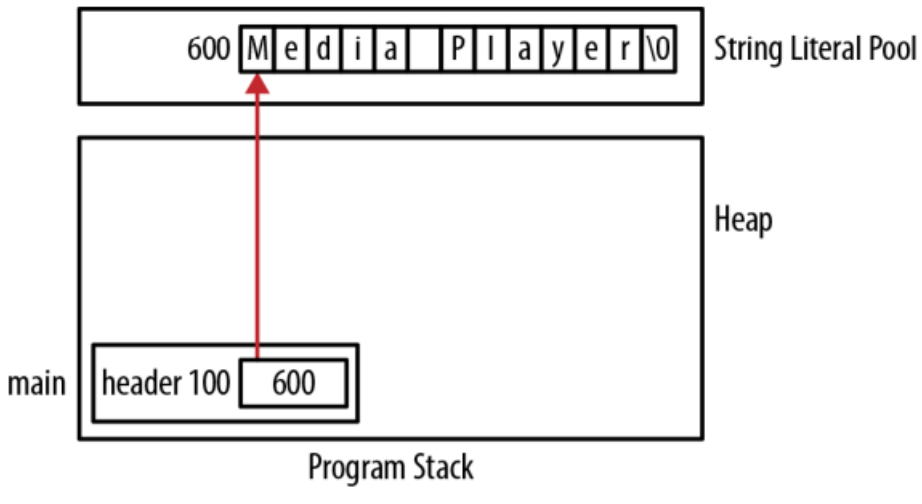


TIP 문자열의 길이를 정할 때

1. NUL terminate를 항상 생각해라
2. `sizeof` operator를 사용하지 마라, `strlen` 함수를 사용해라.
   - sizeof연산자는 문자열의 길이가 아닌, 변수의 크기를 return할 수 있다.

2) char포인터를 사용해서 String Literal Pool에 있는 literal 문자열 자체를 pointing하는 방법.

```
char *header = "Media Player";
```



많이 하는 실수!!

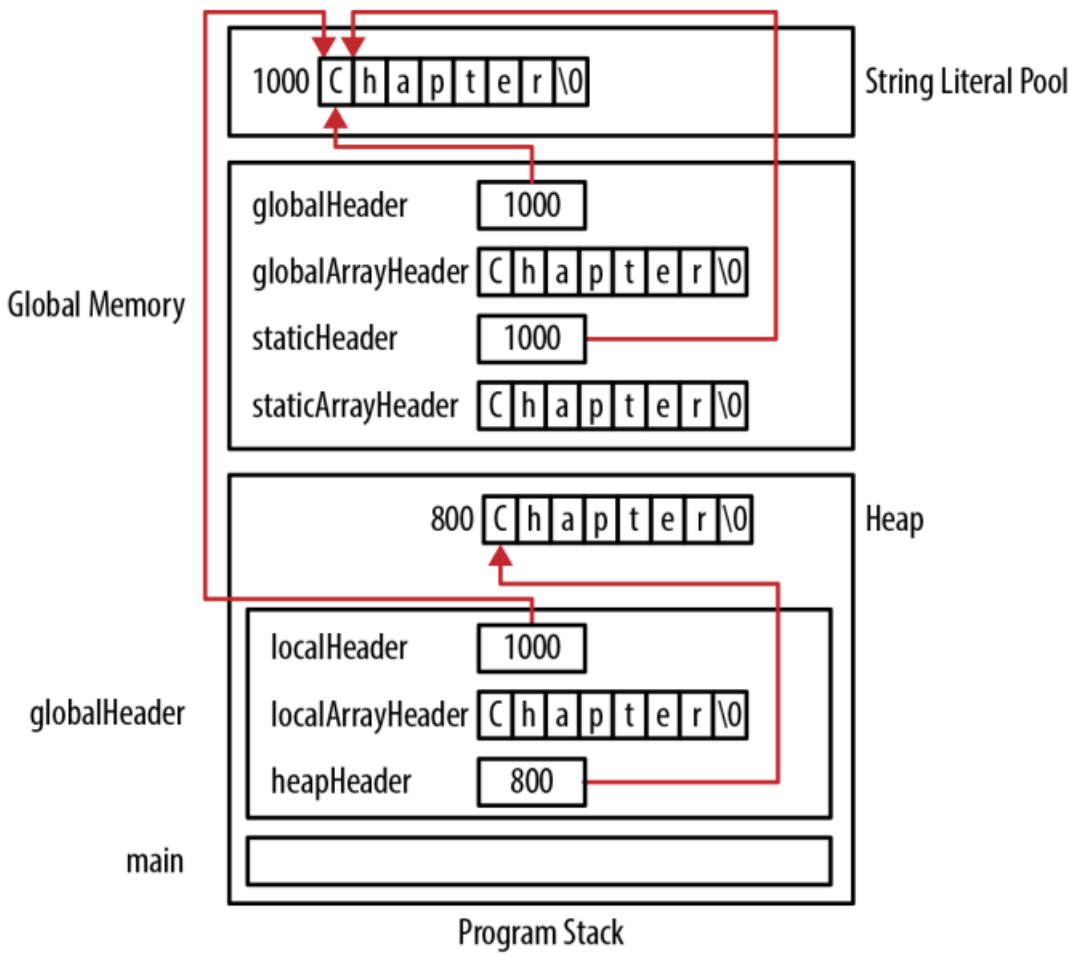
```
char *ptr = '+'; // illegal
```

- char pointer에 char literal을 넣으려고 하면 안된다. character literal의 type은 `int` 이기 때문이다.
- 그리고 오른쪽은 char이기 때문에 애초에 안된다

📌 Summary

```
char *globalHeader = "Chapter";
char globalArrayHeader[] = "Chapter";

void displayHeader()
{
    static char* staticHeader = "Chapter";
    char* localHeader = "Chapter";
    static char staticArrayHeader[] = "Chapter";
    char localArrayHeader[] = "Chapter";
    char *heapHeader = (char*)malloc(strlen("Chapter") + 1);
    strcpy(heapHeader, "Chapter");
}
```

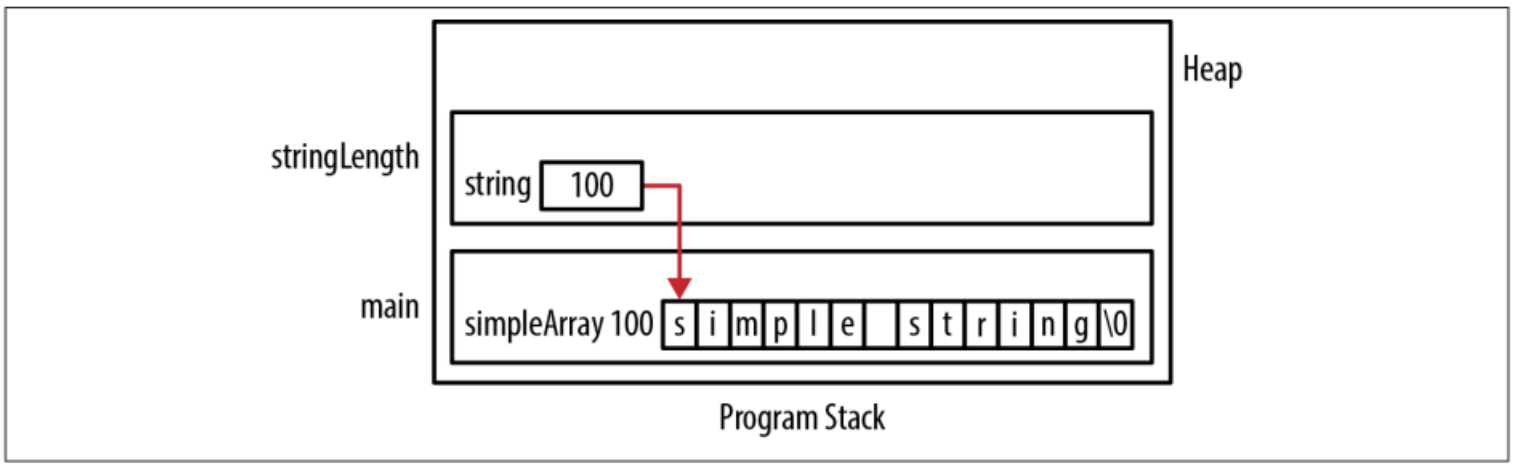


Passing Strings

함수 parameter로 string을 pass하는 3가지의 방법

```
stringLength(simpleArray);
stringLength(&simpleArray);
stringLength(&simpleArray[0]);
```

1. 배열의 이름을 사용하는 방법 : 배열의 이름은 포인터로 사용될 수 있다!
2. 명시적으로 배열의 이름에 주소연산자(&)를 사용하는 방법(비추천)
3. 첫번째 배열의 값의 주소연산자를 사용하는 것



Passing technique "Const Char"

만약 배열을 수정하고 싶지 않다면 `const` 키워드를 꼭 사용하는 습관을 들이자.

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

int get_len(const char* str)
{
    size_t len = 0;
    while(*str)
    {
        len++;
        str++;
    }
    return len;
}
```

Returning String

Case 1) return literal string

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

char *return_str()
{
    return "Hello World";
}
int main()
{
    char *ptr = return_str();
    printf("%s\n", ptr);
    //*ptr = 'a';
    //printf("%s", ptr);
    return (0);
}
```

Case 2) return malloc pointer ⭐

```c
#include <stdio.h>

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

char *make_str(const int size)
{
    char *str = (char*)malloc(sizeof(char) * size + 1);
    for(int i =0; i < 4; i++)
    {
        str[i] = 'a';
    }
    str[4] = '\0';
    return (str);
```

```
}


int main()
{
    char *ptr = make_str(4);
    return (0);
}
```

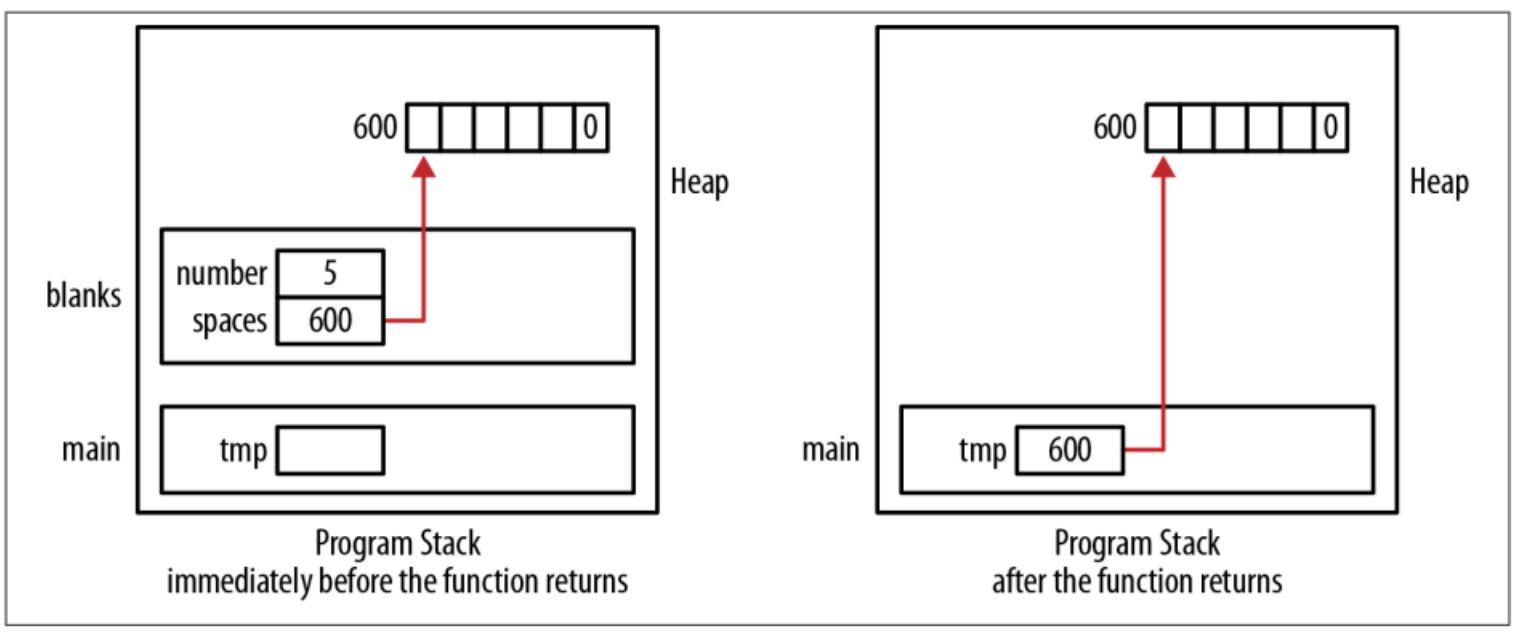


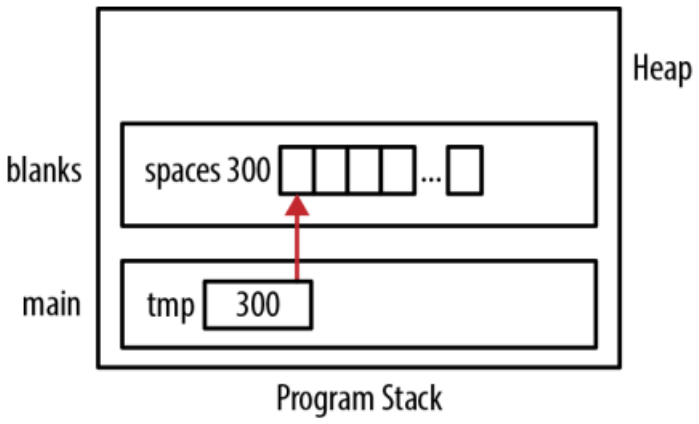
Figure 5-14. Returning dynamically allocated string

- Function에서 Heap에 memory를 allocate를 한 뒤에, 그 시작 주소를 return한다.
- 많이 사용하는 technique이다.

Warning (Return local string)

```
#define MAX_TAB_LENGTH 32

char* blanks(int number) {
    char spaces[MAX_TAB_LENGTH];
    int i;
    for (i = 0; i < number && i < MAX_TAB_LENGTH; i++) {
        spaces[i] = ' ';
    }
    spaces[i] = '\0';
    return spaces;
}
```



- memory area가 corrupt될 가능성이 있다.
- 컴파일러가 Error처리를 한다.