System Software Experiment 1
Lecture 4 - Pointer

spring 2018

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungyunkwan University
http://csl.skku.edu
Inside our PC

Mainboard

CPU

RAM (Memory)

Device (HDD, SSD)
Memory Structure
Memory Structure

- The size of region on memory each data takes differs based on the type of data.
  - char variable: 1 byte
  - int variable: 4 byte

```c
int main(void)
{
    int i = 4;
    char c = 65;
}
```
Memory Structure

• The size of region on memory each data takes differs based on the type of data.
  • char variable: 1 byte
  • int variable: 4 byte

```c
int main(void)
{
    int i = 4;
    char c = 65;
}
```
Reference Operator &

• &: calculates the address of given variable.

```c
int main(void)
{
    int i = 4;
    printf("i: %d\n", i);
    printf("addr: %p\n", &i);
}
```

```
$ ./test_addr
4 1
```
Pointer

- A variable storing an address.
  - Size of pointer variable: 8 byte in 64bit
    (* In 32bit system, 4 byte *)

```c
int main(void)
{
    int i = 4;
    int *p = &i;
}
```
Pointers

• A variable storing an address.
  • Size of pointer variable: 8 byte in 64bit
    (* In 32bit system, 4 byte )

```c
int main(void)
{
    int i = 4;
    int *p = &i;
}
```
Dereference Operator *

• Returns the value stored in the address.

```c
int main(void) {
    int i = 4;
    int *p = &i;

    printf("%dn", *p);
    *p = 10;
    printf("%dn", *p);
    printf("%dn", i);
}
```

4
10
10
Dereference Operator *

• Returns the value stored in the address.

```c
int main(void)
{
    int i = 4;
    int *p = &i;

    printf("%d\n", *p);
    *p = 10;
    printf("%d\n", *p);
    printf("%d\n", i);
}
```

4
10
10
Pointer Data Type Match

• Make sure that the type of pointer and variable match with each other.

```c
int main(void)
{
    int i = 4;
    char *p = &i;
    p = &l;  \textcolor{red}{	extarrow{Warning}}
}
```
Pointer Arithmetics

- Addition and subtraction is possible against pointer variables
- The amount of change depends on the data type the address (inside the pointer) indicates.

### Pointer type ++ increment

<table>
<thead>
<tr>
<th>Pointer type</th>
<th>++ increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>unsigned long</td>
<td>8</td>
</tr>
</tbody>
</table>

```c
int main(void)
{
    int *p = (int *)1000;
    printf("%dn", p);
    p++;
    printf("%dn", p);
}
```

```
1000
1004
```
Pointer Arithmetics

• Addition and subtraction is possible against pointer variables
• The amount of change depends on the data type the address (inside the pointer) indicates.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*p++;</td>
<td>Increment to the address p stores.</td>
</tr>
<tr>
<td>(*p)+++;</td>
<td>Increment to the value p points.</td>
</tr>
<tr>
<td>p++;</td>
<td>Increment to the address p stores.</td>
</tr>
</tbody>
</table>
Pointer and Function

• Call-by-value
  • 값에 의한 호출
• Call-by-reference
  • 참조에 의한 호출
Call-by-value

• Propagate a value as a parameter.

```c
int sum(int a)
{
    ...
}

int main(void)
{
    int i = 4;
    sum(i);
}
```
Call-by-reference

• Propagate an address as a parameter (via pointer)

```c
int sum(int *p)
{
    ...
}

int main(void)
{
    int i = 4;
    sum(&i);
}
```
Swap Example (call-by-value)

- 변수 2개의 값을 바꾸는 작업을 함수로 작성

```c
int main(void)
{
    int a = 10, b = 20;

    printf("%d %d\n", a, b);
    swap(a, b);
    printf("%d %d\n", a, b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```

10 20
10 20
Swap Example (call-by-reference)

```c
int main(void)
{
    int a = 10, b = 20;
    printf("%d %d\n", a, b);
    swap(&a, &b);
    printf("%d %d\n", a, b);
}

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

10 20
20 10

Try to avoid using global variables as long as you can!
Memory Dynamic Allocation

- Memory allocation methods
  - Static: Declare certain variables and automatically release after the end of program.
    - Allocated during compile time.
  - Dynamic: Allocate memory as much as an object needs during runtime, release in any sequence.
    - Directly allocate the region using malloc()
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
    char *a = (char *)malloc(sizeof(char) * 4);

    printf("%p\n", a);
    *a = 4;
    *(a+1) = 5;
    printf("%p\n", a);
    printf("%p\n", a);
}
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
    char *a = (char *)malloc(sizeof(char) * 4);

    printf("%p\n", a);
    *a = 4;
    *(a+1) = 5;
    printf("%p\n", a++);
}

0x49
0x50
Memory Dynamic Alloc.

```c
int main(void)
{
    char *s;
    s = (char *)malloc(sizeof(char) * 4);
    scanf("%s", s); // "abcd"
    printf("%s", s);
}
```

sizeof(char) * 4 = 4 byte is allocated during runtime. The address of first allocated region is stored in s.
Memory Dynamic Alloc.

```c
int main(void)
{
    char *s;
    scanf("%s", s); // "abcd"
    printf("%s", s);
}
```

(char *) s still has no address information where to store the data.
Double Pointer

• A pointer pointing a pointer variable.

```c
int main(void)
{
    int a = 3;
    int *p = &a;
    int **pp = &p;

    printf("%d\n", *p);
    printf("%p\n", pp);
    printf("%p\n", *pp);
    printf("%d\n", **pp);
}
```
Exercise

For given string, the program starts printing out again from the middle.
For every iteration, Each character should change its location with the opposite character around the center one.
The length of string should be odd.

* Make sure you add library stdlib.h on your program!

#include <stdlib.h>