Loughborough University in London

C Programming

- Pointer

Dr. Hyun Lim

h.lim@lboro.ac.uk

Institute for Digital Technologies

Loughborough University London

In short, the **pointers** in C/C++ are just **memory addresses**.

Dereference operator * *Address-of* operator &

- For example,

```
int P = 10;
int *ptr;     // a pointer *ptr  or * ptr
ptr = &P;  // address of P
```

✓ *&P*: *address of P*.

✓ *ptr* is pointing to the location in memory of the variable *P*.
ptr has the *address of the location*.

✓ **ptr* gives the actual data, i.e. 10, stored at the location/ address.

Definition: A pointer is a **variable that holds the memory address** of another variable, where the actual data is stored.

Example

```
int P = 10;
int* ptr;  // a pointer *ptr
ptr = &P;
```

```
int* ptr = &P;
```

```
int *ptr = &P;
```

```
printf("\n P=%d \n Address of P==&P==%d\n",P, &P);
printf("\n ptr=%d \n *ptr=%d \n", ptr, *ptr);
```

Definition: A pointer is a **variable that holds memory address** of another variable, where the actual data is stored.

When do we use the operator *

- The * operator appears before a pointer variable in only two cases:
  - ✓ When declaring a pointer variable, e.g.

```
int *ptr;    //or, int* ptr
```

  - ✓ When de-referencing a pointer variable, e.g.

```
ptr = &P;
printf("%i", *ptr);
```

Example

```
int P = 10;
int* ptr;  // a pointer *ptr
ptr = &P;

printf("\n P=%d \n Address of P==&P==%d \n", P, &P);
printf("\n ptr=%d \n *ptr=%d \n", ptr, *ptr);

P = 20;
printf("\n P=%d \n Address of P==&P==%d \n", P, &P);
printf("\n ptr=%d \n *ptr=%d \n", ptr, *ptr);
```

```
P=10
Address of P==&P==18475964

ptr=18475964
*ptr=10

P=20
Address of P==&P==18475964

ptr=18475964
*ptr=20
```

The address does not change.
If we change the value stored in the variable which the pointe points to, the will return the new value.

"The address may vary according to your computer."

Example - *Keyboard input: scanf, scanf_s()*

scanf_s("*input data type***", *address of the variable*)** // new compilers
scanf() // for old C compilers, e.g., NewbieIDE, Code::Block, Xcode

input/ output

```
int main(void)
 {
   int page;
   printf("Page number: ");
   scanf_s("%d", &page);   // Visual Studio
    // or scanf  for old C compilers
   printf("Your page number: %d \n", page);
    // getchar();  // optional
   return 0;
  }
```

Initialising a Pointer

```
int *ptr;
```

```
ptr = &P;
```

- Once a pointer is declared, we may initialize it.
- This makes the pointer pointing to something.
- Point to nothing; it is dangerous.
- Uninitialised pointers will not cause a compiler error, but using an uninitialised pointer could result in unpredictable and potentially disastrous outcomes.
- Until a pointer holds an address of a variable, it isn't useful.
- ***C*** uses two pointer operators,

1. **Dereference** operator (*) – asterisk symbol
2. **Address-of** operator (&) – ampersand symbol, means to return the address of xx.

- When & operator is placed before the name of a variable, it will return the memory **address** of the variable instead of stored value.

```
int P;
```

```
int *ptr;
ptr = &P;
```

==

```
int *ptr = &P;
```

Summary

- *&A*: gives the (memory) **address** of (a variable) *A*.
- **Pointer** variable: let *int *ptr_A = &A*
  - *ptr_A* contains the **address**, on the other hand,
  - **ptr_A* gives the **actual value** of the data to which the pointer is pointing, i.e. *de-referencing*.

Pointer and Array

Array
- *Name of an array*

char my_ch_array **[**10**]**

name of an array

size

my_ch_array

h	o	w	a	r	E	y	O	u	\0
0	1	2	3	4	5	6	7	8	9

- name of an array
- the address of the element 0, i.e.
  my_ch_array == **&my_ch_array[0]**
- a pointer variable, i.e.
  *my_ch_array == *h*

*(my_ch_array+6) == *y*

Exercise

```
char author0[] = "Li"; // as we did it before
printf("\n %p \n", author0);
                                // %p for pointers
// name of an array == address of the element 0
printf("\n %c \t %c", *author0, *(author0 + 1));
// name of an array is a pointer variable
// *(name + n) will return the data stored in that location
```

Pointer and Post Increment

```
int P[ ] = {100, 120, 130, 135}; // a number array
int* ptr = P;
for ( int i=0; i < (4-1); /* do nothing*/ )
  {
    int i0 = i++;  // cf, ++i
    int dPi=*(ptr+i) - *(ptr+i0);
 // difference between two neighbouring elements
 // int dPi = P[i]-P[i0];
    printf("\n P[%d] - P[%d] =%d. \n", i, i0, dPi);
  }
```

Character, (Character) String String Array

```
int main()
 {
  char aa = 'a';      // Character
  printf("\n %c \n ", aa);

  char bb[] = "bbbb~ lalala~"; //(Character) String
  printf("\n %s \n ", bb);

  char *AA[] = {"BB","CC","DD"}; // String array
  printf("\n %s \n", AA[1]);

   // cf. int BB[] = {10,20,123}; // number array
   // printf("\n %d \n", BB[1]);
  return 0;
 }
```

String Array

```
int main()
 {
  int numb = 3;
  char* author[] = {"Li","Yang","Kim"}; // String Array
     // *author[numb]
  for (int i = 0; i < numb; i++)
    {
     printf("\nNr.[%d]\n", i);
     printf("Author: %s.\n", author[i]);
    }
  return 0;
 }
```

author[0] points to **L**
author[1] points to **Y**
author[2] points to **K**

```
char author0[] = "Li"; // as we did it before
char* author_a = author0;
printf("\n %s \n ", author_a); // %s requires a pointer variable
printf("%c", *(author0+1));
                // cf. %c, %d, %f require char, int, float variables
```

String & Keyboard input

```
#include<stdio.h>

int main(void)
{
   int nr=1;
   char name[20], title[40];     // char arrays
   int page;

   printf("Could you put book information details?\n");
   printf("Name(without space): ");     // No space
        scanf_s("%s", name, sizeof(name)); // sizeof(name)==20
   printf("Title(without space): ");
        scanf_s("%s", title, sizeof(title)); // sizeof(title)==40
   printf("Page number: ");
        scanf_s("%d", &page);
    // scsnf() for old C compilers, e.g., NewbieIDE, Code::Block, Clion, Xcode
   printf("\n\n\n[The details you've saved] \n");
   printf("No.[%d] \n", nr);          //%i integer number
   printf("Name: %s \n", name);   //%s character string
   printf("Title: %s \n", title);
   printf("Page number: %d \n", page);
    //while (!_kbhit());  // optional
return 0;
}
```

address

String & Keyboard input

// Input including spaces

```
printf("Name: ");
    scanf_s("%[^\n]s", name, sizeof(name));
rewind(stdin);
printf("Title: ");
    scanf_s("%[^\n]s", title, 40);
  //or, scanf("%[^\n]s", title);
printf("Page number: ");
    scanf_s("%d", &page);
```

- \n (enter) indicates the end of the input
- take an entire string including spaces until it gets \n (enter)

- Empty the stdin buffer holding \n

// *cf.* fgets(string, sizeof(string),stdin) will also read the whole line until '\n' is found.

String & Keyboard input

// cf. **fgets(string, sizeof(string),stdin)**

// instead of **scanf()**

```
printf("Name: ");
fgets(name,sizeof(name),stdin);
        // fgets() will also read the whole string until \n is found.
fflush(stdin) ;      // empty the stdin buffer holding
```