Introduction to C Programming

— Pointers —

Waseda University

Today's Topics

Understanding pointers

- Memory of computer, Addresses of variable.
- The swap function
- Pointers (address variables)
- The operator ampersand (&)
- The operator asterisk (*)
- Pointers and Arrays

Memory addresses

- Most modern computers are byte-addressable (each address identifies a single byte).
- 1byte is equal to 8bits.
- Addresses are assigned automatically when a variable is declared.

char	:	1byte	(8bits)
int	:	4bytes	(32bits)
double	:	8bytes	(64bits)

- A variable is assigned by specifying the address that points to that variable.
- Pointers contain the addresses of other variables.

Address	Contents	Example
0x00000001	0x32	Allocated to int a
0x00000002	0x4a	
0x00000003	0x2f	
0x00000004	0xaf	
0x00000005	0xd4	Allocated to double b (needs 8bytes)
0x00000006	0x29	
0x00000007	0x82	
0x00000008	0xcc	
0x00000009	0x3d	
0x0000000a	0x10	
0x0000000b	0x04	
0x0000000c	0x27	
0x0000000d	0x7d	char c
...	...	

Memory addresses

- Most modern computers are byte-addressable (each address identifies a single byte).
- 1byte is equal to 8bits.
- Addresses are assigned automatically when a variable is declared.

char	:	1byte	(8bits)
int	:	4bytes	(32bits)
double	:	8bytes	(64bits)

- A variable is assigned by specifying the address that points to that variable.
- Pointers contain the addresses of other variables.

Address	Contents	Example
0x00000001	0x32	Allocated to int a
0x00000002	0x4a	
0x00000003	0x2f	
0x00000004	0xaf	
0x00000005	0xd4	Allocated to double b (needs 8bytes)
0x00000006	0x29	
0x00000007	0x82	
0x00000008	0xcc	
0x00000009	0x3d	
0x0000000a	0x10	
0x0000000b	0x04	
0x0000000c	0x27	
0x0000000d	0x7d	char c
...	...	

Review (Scope of local variable)

- A scope of a local variable is limited to the function in which it is declared.
- When a value of a parameter is changed in a function, the input value is never changed in the main function.

```
#include <stdio.h>
void func(int x){    /* x is declared only in this function. */
  x=7;               /* The x below is different from this x. */
}
int main(void){
  int x=3;
  func(x);  /* Parameter x is 3 and copies 3 to the x above. */
  printf("x is %d.¥n", x);                /* Here "x is 3." */
  return 0;
}
```

Swap function

Swapping contents of two variables

```
void swap(int a, int b){
     int c;
     c = a;     a = b;      b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(x,y);         /* Swaps values of x and y. */
     :
}
```

- But x and y never change above code.
  (Values of x,y copy to above function but these are never changed.)
  - → If we know the pointer of this variable, these can be changed.
  - → We use pointers of x and y !
- Use operators ampersand“&” and asterisk “*”.
  (We already use “&” in the scanf function.)

Swap function

Swapping contents of two variables

```
void swap(int a, int b){
     int c;
     c = a;     a = b;      b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(x,y);         /* Swaps values of x and y. */
     :
}
```

- But `x` and `y` never change above code.
  (Values of `x`,`y` copy to above function but these are never changed.)
  → If we know the pointer of this variable, these can be changed.
  → We use pointers of `x` and `y` !
- Use operators ampersand "&" and asterisk "*".
  (We already use "&" in the scanf function.)

Ampersand &

The operator ampersand (&) returns an address of a variable.

&name_of_variable

```
int a=50;
printf("a=%d, hex=%x, address=%p ¥n",a ,a, &a);
```

- "%x" is called the hexadecimal conversion specification.
- "%p" is called the pointer conversion specification.
- Prints the address of a.

Address	Contents	
0x1234abc0 (Address of a)	0x32 (Value of a)	Allocated to int a
0x1234abc4	...	
...	...	

&a : Pointer to variable a (address of a)

Pointers (address variables)

A pointer contains an address of a variable.

Type *name_of_variable

```
int *p, a=50;
p = &a;
```

- Type is the same as the type of variables to which a pointer points.

Address	Contents	
0x1234abc0 (Address of a)	0x32 (Value of a)	Allocated to int a
0x1234abc4	0x1234abc0 (Address of a)	Allocated to int *p
0x1234abc8	...	
...	...	

p	: Pointer to an integer (may be a)
p=&a	: Assign the address of a to the pointer p

Pointers

- Define several pointers by the following:

  ```
  int *p, *q;
  ```

- Type of void indicates a generic pointer.
  (a pointer that can point to any type of variables)

  ```
  void *p;
  ```

- When a pointer points to nothing, it is called a NULL pointer.

  ```
  int *p=0;
  if (p==NULL) ...;
  ```

Asterisk *

- The operator asterisk (*) returns the object to which a pointer points. (ex. the expression *p indicates the variable a.)
- You can read or assign a value of variable by using pointers.

Address	Contents	
0x1234abc0 (Address of a)	0x32 (Value of a)	Allocated to int a
0x1234abc4	0x1234abc0 (Address of a)	Allocated to int *p
0x1234abc8	...	
...	...	

```
int a;     /* Define a variable "a" */
int *p;    /* Define a pointer */
p = &a;    /* Point to "a" (assign the address of "a" ) */
*p = 3;    /* Set "a" to 3 (indicates a=3;) */
```

Exercise 1

Confirm a result of this program.

```
#include<stdio.h>

int main(void){
    int a=5,b=10;
    int *p;
    p=&a;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);

    *p=b;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);
    return 0;
}
```

- Name of this program should be pointer.c.

Exercise 1

Confirm a result of this program.

```
int main(void){
    int a=5,b=10;
    int *p;
    p=&a;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);

    *p=b;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);
}
```

Outputs:

```
a=5, b=10, *p=5
a=10, b=10, *p=10
```

Exercise 1

Confirm a result of this program.

```
int main(void){
    int a=5,b=10;
    int *p;
    p=&a;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);

    *p=b;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);
}
```

Outputs:

```
a=5, b=10, *p=5
a=10, b=10, *p=10
```

Exercise 1

Confirm a result of this program.

```
int main(void){
    int a=5,b=10;
    int *p;
    p=&a;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);

    *p=b;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);
}
```

Outputs:

```
a=5, b=10, *p=5
a=10, b=10, *p=10
```

Exercise 1

Confirm a result of this program.

```
int main(void){
    int a=5,b=10;
    int *p;
    p=&a;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);

    *p=b;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);
}
```

Outputs:

```
a=5, b=10, *p=5
a=10, b=10, *p=10
```

Swap function

Swapping contents of two variables

```
void swap(int *a, int *b){   /* Get addresses of two variables */
     int c;
     c = *a;    *a =*b;      *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);                /* Assign addresses as parameters */
     :
}
```

	Address	Contents
x	(&x)	3 → 7
y	(&y)	7 → 3
a		&x
b		&y
c		3

```
Note:
  int a;     /* Define a variable "a" */
  int *p;    /* Define a pointer */
  p = &a;    /* Point to "a" */
  *p = 3;    /* Set "a" to 3 */
```

Swap function

Swapping contents of two variables

```
void swap(int *a, int *b){   /* Get addresses of two variables */
     int c;
     c = *a;    *a =*b;      *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);                /* Assign addresses as parameters */
     :
}
```

	Address	Contents
x	(&x)	3 → 7
y	(&y)	7 → 3
a		&x
b		&y
c		3

```
Note:
  int a;    /* Define a variable "a" */
  int *p;   /* Define a pointer */
  p = &a;   /* Point to "a" */
  *p = 3;   /* Set "a" to 3 */
```

Swap function

Swapping contents of two variables

```
void swap(int *a, int *b){   /* Get addresses of two variables */
     int c;
     c = *a;    *a =*b;      *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);                 /* Assign addresses as parameters */
     :
}
```

	Address	Contents
x	(&x)	3 → 7
y	(&y)	7 → 3
a		&x
b		&y
c		3

```
Note:
  int a;    /* Define a variable "a" */
  int *p;   /* Define a pointer */
  p = &a;   /* Point to "a" */
  *p = 3;   /* Set "a" to 3 */
```

Swap function

Swapping contents of two variables

```
void swap(int *a, int *b){   /* Get addresses of two variables */
     int c;
     c = *a;     *a =*b;       *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);                /* Assign addresses as parameters */
     :
}
```

	Address	Contents
x	(&x)	3 → 7
y	(&y)	7 → 3
a		&x
b		&y
c		3

```
Note:
  int a;    /* Define a variable "a" */
  int *p;   /* Define a pointer */
  p = &a;   /* Point to "a" */
  *p = 3;   /* Set "a" to 3 */
```

Swap function

Swapping contents of two variables

```
void swap(int *a, int *b){    /* Get addresses of two variables */
     int c;
     c = *a;     *a =*b;       *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);                 /* Assign addresses as parameters */
     :
}
```

	Address	Contents
x	(&x)	3 → 7
y	(&y)	7 → 3
a		&x
b		&y
c		3

```
Note:
  int a;     /* Define a variable "a" */
  int *p;    /* Define a pointer */
  p = &a;    /* Point to "a" */
  *p = 3;    /* Set "a" to 3 */
```

Swap function

Swapping contents of two variables

```
void swap(int *a, int *b){   /* Get addresses of two variables */
     int c;
     c = *a;    *a =*b;      *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);                 /* Assign addresses as parameters */
     :
}
```

	Address	Contents
x	(&x)	$3 \rightarrow 7$
y	(&y)	$7 \rightarrow 3$
a		&x
b		&y
c		3

```
Note:
  int a;    /* Define a variable "a" */
  int *p;   /* Define a pointer */
  p = &a;   /* Point to "a" */
  *p = 3;   /* Set "a" to 3 */
```

Exercise 2

Create a function "order" whose parameters are two pointers of integer. This function sorts two parameters in ascending order (use swap function).

- Name should be order.c.
- In the main function set $x = 7, y = 3$ and print addresses and values.
- Call the oder function (by order(&x,&y);) such that $x \leq y$.
- Print addresses and values of x and y, respectively.
- Output will be the followings:

```
x:  address = 0xbffff944, value = 7,
y:  address = 0xbffff940, value = 3,
x:  address = 0xbffff944, value = 3,
y:  address = 0xbffff940, value = 7,
```

Pointers and arrays

- The elements of an array are assigned to consecutive addresses (ex. int A[3]).
- The variable A is used as a pointer that points to A[0].
- *A indicates A[0].
- The pointer can be used to find each element of the array (ex. *(A+1) indicates A[1]).
- The address of A[i] is given by & (ex. `&A[i]`).

Address	Contents	Example
0x00000001	0x32	A[0]
0x00000002	0x4a	
0x00000003	0x2f	
0x00000004	0xaf	
0x00000005	0xd4	A[1]
0x00000006	0x29	
0x00000007	0x82	
0x00000008	0xcc	
0x00000009	0x3d	A[2]
0x0000000a	0x10	
0x0000000b	0x04	
0x0000000c	0x27	
...	...	

Pointers and arrays

- The elements of an array are assigned to consecutive addresses (ex. int A[3]).
- The variable A is used as a pointer that points to A[0].
- *A indicates A[0].
- The pointer can be used to find each element of the array (ex. *(A+1) indicates A[1]).
- The address of A[i] is given by & (ex. `&A[i]`).

Address	Contents	Example
0x00000001	0x32	A[0]
0x00000002	0x4a	
0x00000003	0x2f	
0x00000004	0xaf	
0x00000005	0xd4	A[1]
0x00000006	0x29	
0x00000007	0x82	
0x00000008	0xcc	
0x00000009	0x3d	A[2]
0x0000000a	0x10	
0x0000000b	0x04	
0x0000000c	0x27	
...	...	

Arrays as parameters

When we want to assign an array to parameters of a function, the pointer of the array is necessary.

Set 0 in all elements of array

```
void setZero( int *a, int size){
    int i;
    for(i=0; i<size; i++)
        a[i]=0;
}
int main(void){
    int A[10];
    setZero(A,10);              /* Not &A */
    ⋮
```

Summary

- What is pointer?

```
int a; /* Define a variable "a" */
int *p; /* Define a pointer */
p = &a; /* Point to "a" */
*p = 3; /* Set "a" to 3 */
```

- Assign pointers to parameters.

Change value of a

```
int a;
scanf("%d",&a); /* scanf("%d",a); is not working. */
```

- Pointers and arrays