Lecture 10
Pointers

Outline

- Computer Memory Structure
- Addressing Concept
- Introduction to Pointer
- Pointer Manipulation
- Summary

Computer Memory Revisited

- Computers store data in memory slots
- Each slot has an *unique address*
- Variables store their values like this:

Addr	**Content**	**Addr**	**Content**	**Addr**	**Content**	**Addr**	**Content**
1000	i: 37	1001	j: 46	1002	k: 58	1003	m: 74
1004	a[0]: ‘a’	1005	a[1]: ‘b’	1006	a[2]: ‘c’	1007	a[3]: ‘\0’
1008	ptr: 1001	1009	…	1010		1011	

Computer Memory Revisited

- *Altering the value of a variable is indeed changing the content of the memory*
  - e.g. i = 40; a[2] = ‘z’;

Addr	Content	Addr	Content	Addr	Content	Addr	Content
1000	i: 40	1001	j: 46	1002	k: 58	1003	m: 74
1004	a[0]: ‘a’	1005	a[1]: ‘b’	1006	a[2]: ‘z’	1007	a[3]: ‘\0’
1008	ptr: 1001	1009	…	1010		1011	

Addressing Concept

- Pointer stores the **address** of another entity
- It **refers** to a memory location
- The size of a pointer variable is **4 bytes**.

```
int i = 5;
int *ptr;                          /* declare a pointer variable */
ptr = &i;                          /* store address-of i to ptr */
printf("*ptr = %d\n", *ptr);       /* refer to referee of ptr */
```

Why do we need Pointer?

- Simply because it's there!
- It is used in some circumstances in C

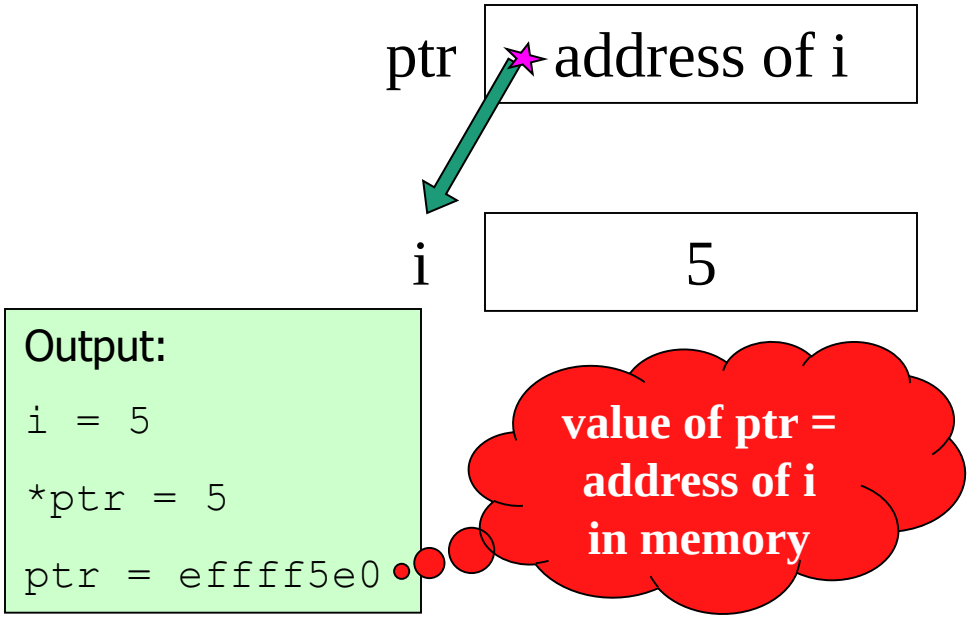
Remember this?

```
scanf("%d", &i);
```

What actually *ptr* is?

- **ptr** is a variable storing **an address**
- ptr is **NOT** storing the actual value of i

```
int i = 5;
int *ptr;
ptr = &i;
printf("i = %d\n", i);
printf("*ptr = %d\n", *ptr);
printf("ptr = %p\n", ptr);
```



Twin Operators

- &: Address-of operator
  - Get the *address* of an entity
    - e.g. `ptr = &j;`

Addr	Content	Addr	Content	Addr	Content	Addr	Content
1000	i: 40	1001	j: 33	1002	k: 58	1003	m: 74
1004	ptr: 1001	1005		1006		1007	

Twin Operators

- *: De-reference operator
  - Refer to the *content* of the referee
    - e.g. `*ptr = 99;`

Addr	Content	Addr	Content	Addr	Content	Addr	Content
1000	i: 40	1001	j: 99	1002	k: 58	1003	m: 74
1004	ptr: 1001	1005		1006		1007	

Example: Pass by Reference

- Modify behaviour in argument passing

```
void f(int j)
{
  j = 5;
}
void g()
{
  int i = 3;
  f(i);
}
```

i = 3

```
void f(int *ptr)
{
  *ptr = 5;
}
void g()
{
  int i = 3;
  f(&i);
}
```

i = 5

An Illustration

```
int i = 5, j = 10;
int *ptr;
int **pptr;
ptr = &i;
pptr = &ptr;
*ptr = 3;
**pptr = 7;
ptr = &j;
**pptr = 9;
*pptr = &i;
*ptr = -2;
```

Data Table			
Name	Type	Description	Value
i	int	integer variable	5
j	int	integer variable	10

An Illustration

```
int i = 5, j = 10;
int *ptr;     /* declare a pointer-to-integer variable */
int **pptr;
ptr = &i;
pptr = &ptr;
*ptr = 3;
**pptr = 7;
ptr = &j;
**pptr = 9;
*pptr = &i;
*ptr = -2;
```

Data Table			
Name	Type	Description	Value
i	int	integer variable	5
j	int	integer variable	10
ptr	int *	integer pointer variable	

An Illustration

```
int i = 5, j = 10;
int *ptr;
int **pptr;  /* declare a pointer-to-pointer-to-integer
variable */
ptr = &i;
pptr = &ptr;
*ptr = 3;
**pptr = 7;
ptr = &j;
**pptr = 9;
*pptr = &i;
*ptr = -2;
```

Data Table			
Name	Type	Description	Value
i	int	integer variable	5
j	int	integer variable	10
ptr	int *	integer pointer variable	
pptr	int **	integer pointer pointer variable	
		Double Indirection	

An Illustration

```
int i = 5, j = 10;
int *ptr;
int **pptr;
ptr = &i;     /* store address-of i to ptr */
pptr = &ptr;
*ptr = 3;
**pptr = 7;
ptr = &j;
**pptr = 9;
*pptr = &i;
*ptr = -2;
```

Data Table			
Name	Type	Description	Value
i	int	integer variable	5
j	int	integer variable	10
ptr	int *	integer pointer variable	address of i
pptr	int **	integer pointer pointer variable	
*ptr	int	de-reference of ptr	5

An Illustration

```
int i = 5, j = 10;
int *ptr;
int **pptr;
ptr = &i;
pptr = &ptr; /* store address-of ptr to pptr */
*ptr = 3;
**pptr = 7;
ptr = &j;
**pptr = 9;
*pptr = &i;
*ptr = -2;
```

Data Table			
Name	Type	Description	Value
i	int	integer variable	5
j	int	integer variable	10
ptr	int *	integer pointer variable	address of i
pptr	int **	integer pointer pointer variable	address of ptr
*pptr	int *	de-reference of pptr	value of ptr (address of i)

An Illustration

```
int i = 5, j = 10;
int *ptr;
int **pptr;
ptr = &i;
pptr = &ptr;
*ptr = 3;
**pptr = 7;
ptr = &j;
**pptr = 9;
*pptr = &i;
*ptr = -2;
```

Data Table			
Name	Type	Description	Value
i	int	integer variable	3
j	int	integer variable	10
ptr	int *	integer pointer variable	address of i
pptr	int **	integer pointer pointer variable	address of ptr
*ptr	int	de-reference of ptr	3

An Illustration

```
int i = 5, j = 10;
int *ptr;
int **pptr;
ptr = &i;
pptr = &ptr;
*ptr = 3;
**pptr = 7;
ptr = &j;
**pptr = 9;
*pptr = &i;
*ptr = -2;
```

Data Table			
Name	Type	Description	Value
i	int	integer variable	7
j	int	integer variable	10
ptr	int *	integer pointer variable	address of i
pptr	int **	integer pointer pointer variable	address of ptr
**pptr	int	de-reference of de-reference of pptr	7

An Illustration

```
int i = 5, j = 10;
int *ptr;
int **pptr;
ptr = &i;
pptr = &ptr;
*ptr = 3;
**pptr = 7;
ptr = &j;
**pptr = 9;
*pptr = &i;
*ptr = -2;
```

Data Table			
Name	Type	Description	Value
i	int	integer variable	7
j	int	integer variable	10
ptr	int *	integer pointer variable	address of j
pptr	int **	integer pointer pointer variable	address of ptr
*ptr	int	de-reference of ptr	10

An Illustration

```
int i = 5, j = 10;
int *ptr;
int **pptr;
ptr = &i;
pptr = &ptr;
*ptr = 3;
**pptr = 7;
ptr = &j;
**pptr = 9;
*pptr = &i;
*ptr = -2;
```

Data Table			
Name	Type	Description	Value
i	int	integer variable	7
j	int	integer variable	9
ptr	int *	integer pointer variable	address of j
pptr	int **	integer pointer pointer variable	address of ptr
**pptr	int	de-reference of de-reference of pptr	9

An Illustration

```
int i = 5, j = 10;
int *ptr;
int **pptr;
ptr = &i;
pptr = &ptr;
*ptr = 3;
**pptr = 7;
ptr = &j;
**pptr = 9;
*pptr = &i;
*ptr = -2;
```

Data Table			
Name	Type	Description	Value
i	int	integer variable	7
j	int	integer variable	9
ptr	int *	integer pointer variable	address of i
pptr	int **	integer pointer pointer variable	address of ptr
*pptr	int *	de-reference of pptr	value of ptr (address of i)

An Illustration

```
int i = 5, j = 10;
int *ptr;
int **pptr;
ptr = &i;
pptr = &ptr;
*ptr = 3;
**pptr = 7;
ptr = &j;
**pptr = 9;
*pptr = &i;
*ptr = -2;
```

Data Table			
Name	Type	Description	Value
i	int	integer variable	-2
j	int	integer variable	9
ptr	int *	integer pointer variable	address of i
pptr	int **	integer pointer pointer variable	address of ptr
*ptr	int	de-reference of ptr	-2

Pointer Arithmetic

- What's `ptr + 1`?

➔The next memory location!

- What's `ptr - 1`?

➔The previous memory location!

- What's `ptr * 2` and `ptr / 2`?

➔Invalid operations!!!

Pointer Arithmetic and Array

```
float a[4];
float *ptr;
ptr = &(a[2]);
*ptr = 3.14;
ptr++;
*ptr = 9.0;
ptr = ptr - 3;
*ptr = 6.0;
ptr += 2;
*ptr = 7.0;
```

Data Table			
Name	Type	Description	Value
a[0]	float	float array element (variable)	?
a[1]	float	float array element (variable)	?
a[2]	float	float array element (variable)	?
a[3]	float	float array element (variable)	?
ptr	float *	float pointer variable	
*ptr	float	de-reference of float pointer variable	?

Pointer Arithmetic and Array

```
float a[4];
float *ptr;
ptr = &(a[2]);
*ptr = 3.14;
ptr++;
*ptr = 9.0;
ptr = ptr - 3;
*ptr = 6.0;
ptr += 2;
*ptr = 7.0;
```

Data Table			
Name	Type	Description	Value
a[0]	float	float array element (variable)	?
a[1]	float	float array element (variable)	?
a[2]	float	float array element (variable)	?
a[3]	float	float array element (variable)	?
ptr	float *	float pointer variable	address of a[2]
*ptr	float	de-reference of float pointer variable	?

Pointer Arithmetic and Array

```
float a[4];
float *ptr;
ptr = &(a[2]);
*ptr = 3.14;
ptr++;
*ptr = 9.0;
ptr = ptr - 3;
*ptr = 6.0;
ptr += 2;
*ptr = 7.0;
```

Data Table			
Name	Type	Description	Value
a[0]	float	float array element (variable)	?
a[1]	float	float array element (variable)	?
a[2]	float	float array element (variable)	3.14
a[3]	float	float array element (variable)	?
ptr	float *	float pointer variable	address of a[2]
*ptr	float	de-reference of float pointer variable	3.14

Pointer Arithmetic and Array

```
float a[4];
float *ptr;
ptr = &(a[2]);
*ptr = 3.14;
ptr++;
*ptr = 9.0;
ptr = ptr - 3;
*ptr = 6.0;
ptr += 2;
*ptr = 7.0;
```

Data Table			
Name	Type	Description	Value
a[0]	float	float array element (variable)	?
a[1]	float	float array element (variable)	?
a[2]	float	float array element (variable)	3.14
a[3]	float	float array element (variable)	?
ptr	float *	float pointer variable	address of a[3]
*ptr	float	de-reference of float pointer variable	?

Pointer Arithmetic and Array

```
float a[4];
float *ptr;
ptr = &(a[2]);
*ptr = 3.14;
ptr++;
*ptr = 9.0;
ptr = ptr - 3;
*ptr = 6.0;
ptr += 2;
*ptr = 7.0;
```

Data Table			
Name	Type	Description	Value
a[0]	float	float array element (variable)	?
a[1]	float	float array element (variable)	?
a[2]	float	float array element (variable)	3.14
a[3]	float	float array element (variable)	9.0
ptr	float *	float pointer variable	address of a[3]
*ptr	float	de-reference of float pointer variable	9.0

Pointer Arithmetic and Array

```
float a[4];
float *ptr;
ptr = &(a[2]);
*ptr = 3.14;
ptr++;
*ptr = 9.0;
ptr = ptr - 3;
*ptr = 6.0;
ptr += 2;
*ptr = 7.0;
```

Data Table			
Name	Type	Description	Value
a[0]	float	float array element (variable)	?
a[1]	float	float array element (variable)	?
a[2]	float	float array element (variable)	3.14
a[3]	float	float array element (variable)	9.0
ptr	float *	float pointer variable	address of a[0]
*ptr	float	de-reference of float pointer variable	?

Pointer Arithmetic and Array

```
float a[4];
float *ptr;
ptr = &(a[2]);
*ptr = 3.14;
ptr++;
*ptr = 9.0;
ptr = ptr - 3;
*ptr = 6.0;
ptr += 2;
*ptr = 7.0;
```

Data Table			
Name	Type	Description	Value
a[0]	float	float array element (variable)	6.0
a[1]	float	float array element (variable)	?
a[2]	float	float array element (variable)	3.14
a[3]	float	float array element (variable)	9.0
ptr	float *	float pointer variable	address of a[0]
*ptr	float	de-reference of float pointer variable	6.0

Pointer Arithmetic and Array

```
float a[4];
float *ptr;
ptr = &(a[2]);
*ptr = 3.14;
ptr++;
*ptr = 9.0;
ptr = ptr - 3;
*ptr = 6.0;
ptr += 2;
*ptr = 7.0;
```

Data Table			
Name	Type	Description	Value
a[0]	float	float array element (variable)	6.0
a[1]	float	float array element (variable)	?
a[2]	float	float array element (variable)	3.14
a[3]	float	float array element (variable)	9.0
ptr	float *	float pointer variable	address of a[2]
*ptr	float	de-reference of float pointer variable	3.14

Pointer Arithmetic and Array

```
float a[4];
float *ptr;
ptr = &(a[2]);
*ptr = 3.14;
ptr++;
*ptr = 9.0;
ptr = ptr - 3;
*ptr = 6.0;
ptr += 2;
*ptr = 7.0;
```

Data Table			
Name	Type	Description	Value
a[0]	float	float array element (variable)	6.0
a[1]	float	float array element (variable)	?
a[2]	float	float array element (variable)	7.0
a[3]	float	float array element (variable)	9.0
ptr	float *	float pointer variable	address of a[2]
*ptr	float	de-reference of float pointer variable	7.0

Pointer Arithmetic and Array

```
float ar[4];
float *ptr;
ptr = &(ar[2]);
*ptr = 3.14;
ptr++;
*ptr = 9.0;
ptr = ptr - 3;
*ptr = 6.0;
ptr += 2;
*ptr = 7.0;
```

- Type of `a` is `float *`
- `ar[2]` ⇔ `*(ar + 2)`

  `ptr = &(ar[2])`

  ➔`ptr = &(*(ar + 2))`

  ➔`ptr = ar + 2`
- `ar` is a memory address *constant*
- `ptr` is a pointer *variable*

More Pointer Arithmetic

- What if `a` is a `double` array?
- A `double` *may* occupy more memory slots!
  - Given **`double *ptr = a;`**
  - What's **`ptr + 1`** then?

Addr	Content	Addr	Content	Addr	Content	Addr	Content
1000	a[0]: 37.9	1001	…	1002	…	1003	…
1004	a[1]: 1.23	1005	…	1006	…	1007	…
1008	a[2]: 3.14	1009	…	1010	…	1011	…

More Pointer Arithmetic

- Arithmetic operators + and –
  - *auto-adjust* the address offset
- According to the *type* of the pointer:
  - 1000 + sizeof(double) = 1000 + 8 = 1008

Addr	Content	Addr	Content	Addr	Content	Addr	Content
1000	a[0]: 37.9	1001	…	1002	…	1003	…
1004	a[1]: 1.23	1005	…	1006	…	1007	…
1008	a[2]: 3.14	1009	…	1010	…	1011	…

Advice and Precaution

- Pros
  - Efficiency
  - Convenience
- Cons
  - Error-prone
  - Difficult to debug

Summary

- A pointer stores the **address** (memory location) of another entity
- Address-of operator (&) **gets the address** of an entity
- De-reference operator (*) **makes a reference** to the referee of a pointer
- Pointer and array
- Pointer arithmetic