Pointers in C

DR. PAWAN KUMAR SINGH
DEPARTMENT OF INFORMATION TECHNOLOGY
JADAVPUR UNIVERSITY
KOLKATA

Addressing Concept

- Pointer stores the **address** of another entity
- It **refers** to a memory location

```
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);
```

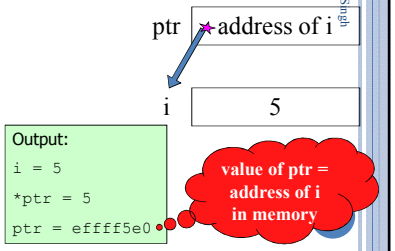

Operators used in Pointers

Dereferencing	Address
*	&
(Value of)	(Address of)

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);
```



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 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;
```

- Type of `a` is `float *`
- `a[2]` ←→ `*(a + 2)`
  - `ptr = &(a[2])`
  - → `ptr = &(*(a + 2))`
  - → `ptr = a + 2`
- `a` is a memory address *constant*
- `ptr` is a pointer *variable*

Advice and Precaution

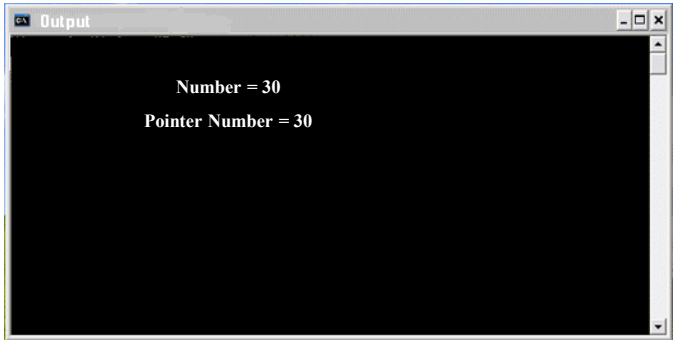
- Pros
  - Efficiency
  - Convenience
- Cons
  - Error-prone
  - Difficult to debug

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

Work your brain

```
void main()
{
int num=10;
int * pnum=NULL;
   pnum = &num;
   *pnum += 20;
   printf("\nNumber = %d", num);
   printf("\nPointer Number = %d", *pnum);
}
```

WORK YOUR BRAIN

```
int a[10] = {1,2,3,4,5,6,7,8,9,12} ,*p, *q , i;
p = &a[2];
q = &a[5];
i = *q - *p;
printf("The value of i is %d" i );
i = *p - *q;
printf("The value of i is %d" i );
a[2] = a[5] = 0;
i = *q - *p;
printf("The value of i is %d" i );
```


Output

```
The value of i is 3
The value of i is -3
The value of i is 0
```


WORK YOUR BRAIN

```
int a[10] = { 2,3,4,5,6,7,8,9,1,0 }, *p, *q;
p = &a[2];
q = p + 3;
p = q – 1;
p+ + ;
printf("The value of p and q are : %d , %d" ,*p,*q);
```


Output

```
The value of p and q are : 7 , 7
```

Work your brain

```
int main()
{
int x[2]={1,2},y[2]={3,4};
int small, big;
small=&x[0];
big=&y[0];
min_max(&small,&big);
printf("small=%d   big=%d",*small,*big);
return 0;
}
min_max(int *a,int *b)
{
a++;
b++;
return (*a,*b);
}
```


Arrays and pointers

- int array[ ] = {1,23,4,5,-100};
- The variable **arr** is a pointer pointing to the first element of the array .
- All other arrays are stored sequentially after that.

- array[1] = ?
- *(arr+1) = ?
- *arr + 1 = ?

- arr[3] = ?
- *(arr+3) = ?
- *arr + 3 = ?

Arrays and pointers

- int array[] = {1,23,4,5,-100};
- The variable "arr" is a pointer pointing to the first element of the array (array[1]).
- All other arrays are stored sequentially after that.

- array[1] = 23
- *(arr+1) = 4
- *arr + 1 = 24

- array[3] = 5
- *(arr+3) = -100
- *arr + 3 = 26

Arrays and pointers

- int array[ ] = {1,23,7,5,10,78,9,87};
- int *ptr = array + 2;

- array[1] = ?
- *(ptr+1) = ?
- *array + 3 = ?
- array[4] = ?
- *ptr + 1 = ?
- &array[0] = ?

Arrays and pointers

- int array[ ] = {1,23,7,5,10,78,9,87};
- int *ptr = array + 2;

- array[1] = 23
- *(ptr+1) = 5
- *array + 3 = 4
- array[4] = 10
- *ptr + 1 = 8
- &array[0] = address of array[0]

Arrays and pointers

- int arr[ ] = {1,2,3,4,5,6,7};

- arr[2] = ?

- *(arr + 2) = ?

- *(2 + arr) = ?

- 2[arr] = ?

Arrays and pointers

- int arr[ ] = {1,2,3,4,5,6,7};

- arr[2] = 3

- *(arr + 2) = 3

- *(2 + arr) = 3

- 2[arr] = 3