計算機程式設計

Computer Programming

Pointers

Instructor: 林英嘉

2024/11/04

W9 Slido: #4031666

GitHub repo

Outline

- Introduction to Pointers
- Practical Properties of Pointers
- Pointers and functions
- Pointers and arrays (11/11)

Introduction to Pointers

[Definition & Declaration] Pointers

- A pointer is a **variable** that is used to store the memory address of another variable.
- A pointer also has a data type, which is the type of the pointed variable.
- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

[Declaration] Pointers

- Pointer declaration: Data_type * Variable_name

  Data_type * is a data type for pointers

- When a pointer variable is declared, its name must be preceded by an **asterisk**:

```
int *iptr; // A pointer points to an int variable (整數指標)

float *fptr; // A pointer points to a float variable (浮點數指標)

double *dfptr; // A pointer points to a double variable (倍準浮點數指標)
```

[Declaration] Make a pointer point to a variable

To build the relationship between a pointer and a variable, we can perform the following operations:

```
1 int main(void){
2     int *iptr; // Declaration of a pointer
3     int i = 10;
4     iptr = &i;
5 }
```

[Declaration] Make a pointer point to a variable

To build the relationship between a pointer and a variable, we can perform the following operations:

```
1 int main(void){
2     int *iptr; // Declaration of a pointer
3     int i = 10;
4     iptr = &i;
5 }
```

Line 4:

[Declaration] Make a pointer point to a variable

To build the relationship between a pointer and a variable, we can perform the following operations:

```
1 int main(void){
2     int *iptr; // Declaration of a pointer
3     int i = 10;
4     iptr = &i;
5 }
```

Line 4:

(1) Use ampersand (&) to get the address of i

variable i

10

0x7fffffffdd6c

variable iptr

0x7fffffffdd70

[Declaration] Make a pointer point to a variable

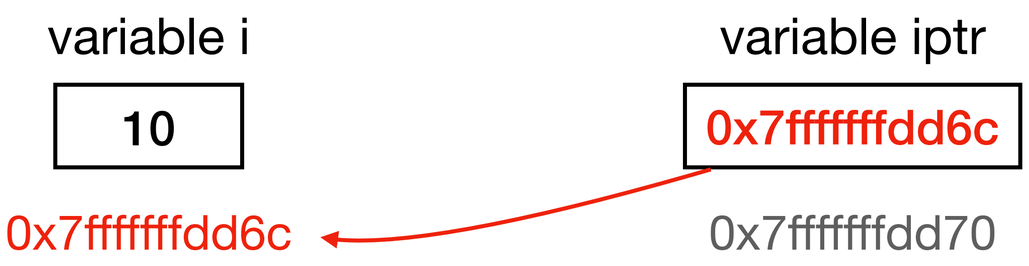
To build the relationship between a pointer and a variable, we can perform the following operations:

```
1 int main(void){
2     int *iptr; // Declaration of a pointer
3     int i = 10;
4     iptr = &i;
5 }
```

Line 4:

(1) Use ampersand (&) to get the address of i

(2) Assigning the address of **i** to the pointer



[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
Variable		
Stored value		
Memory address itself		

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
Variable	`int i = 10;`	
Stored value		
Memory address itself		

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
	`int i = 10;`	
Variable	Variable i	
Stored value		
Memory address itself		

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
	`int i = 10;`	
Variable	Variable i	
Stored value	10	
Memory address itself		

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
	`int i = 10;`	
Variable	Variable i	
Stored value	10	
Memory address itself	0x7fffffffdd6c	

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

Standard variable

```
int i = 10;
```

Pointer variable

```
int *iptr;
iptr = &i;
```

Variable	Variable i
Stored value	10
Memory address itself	0x7fffffffdd6c

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
	`int i = 10;`	`int *iptr;` `iptr = &i;`
Variable	Variable i	Variable iptr
Stored value	10	
Memory address itself	0x7fffffffdd6c	

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
	`int i = 10;`	`int *iptr;` `iptr = &i;`
Variable	Variable i	Variable iptr
Stored value	10	
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
	`int i = 10;`	`int *iptr;` `iptr = &i;`
Variable	Variable i	Variable iptr
Stored value	10	0x7fffffffdd6c
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

[Illustration] Standard variable vs. Pointer variable

- **Variable** comparison:
  - Standard variable: stores a specific data value directly
  - Pointer variable: stores the memory address of another variable

	Standard variable	Pointer variable
	`int i = 10;`	`int *iptr;` `iptr = &i;`
Variable	Variable i	Variable iptr
Stored value	10	0x7fffffffdd6c
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

A pointer and its pointed variable

C-course-materials/06-Pointers/pointer_relationship.c

```
#include <stdio.h>
int main(void){
    int *iptr;
    int i = 10;
    iptr = &i;
    printf(“%d\n”, *iptr); // 取得指標所指向的值
    printf("變數i的位址：%p\n", &i);
    printf("指標指向的位址：%p\n", iptr);
    printf("指標本身的位址：%p\n", &iptr);
}
```

[Important Notes] Declaration of Pointers

- Pointer variables can appear in declarations along with other variables:

```
int i, j, a[10], b[20], *p, *q;
```

- C requires that every pointer variable point only to objects of a particular type

```
int *iptr; // A pointer points only to an int variable

float *fptr; // A pointer points only to a float variable

double *dfptr; // A pointer points only to a double variable
```

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

	Standard variable	Pointer variable
Variable		
Stored value		
Memory address itself		

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

Standard variable

```
int i = 10;
```

Pointer variable

Variable

Stored value

Memory address itself

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

Standard variable

```
int i = 10;
```

Pointer variable

Variable

Variable i

Stored value

Memory address itself

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

	Standard variable	Pointer variable
	`int i = 10;`	
Variable	Variable i	
Stored value	10	
Memory address itself		

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

	Standard variable	Pointer variable
	`int i = 10;`	
Variable	Variable i	
Stored value	10	
Memory address itself	0x7fffffffdd6c	

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

Standard variable

```
int i = 10;
```

Pointer variable

```
int *iptr;
iptr = &i;
```

```
int *iptr = &i;
```

Variable	Variable i
Stored value	10
Memory address itself	0x7fffffffdd6c

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

	Standard variable	Pointer variable
	`int i = 10;`	`int *iptr;` `iptr = &i;`
Variable	Variable i	Variable iptr
Stored value	10	
Memory address itself	0x7fffffffdd6c	

`int *iptr = &i;`

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

	Standard variable	Pointer variable
	`int i = 10;`	`int *iptr;` `iptr = &i;`
Variable	Variable i	Variable iptr
Stored value	10	
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

`int *iptr = &i;`

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

	Standard variable	Pointer variable
	`int i = 10;`	`int *iptr;` `iptr = &i;`
Variable	Variable i	Variable iptr
Stored value	10	0x7fffffffdd6c
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

`int *iptr = &i;`

[Declaration] Make a pointer point to a variable in one line

```
int main(void){
    int i = 10;
    int *iptr = &i; // Declaration of a pointer
}
```

	Standard variable	Pointer variable
	`int i = 10;`	`int *iptr;` `iptr = &i;`
Variable	Variable i	Variable iptr
Stored value	10	0x7fffffffdd6c
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

`int *iptr = &i;`

[Definition] Asterisk (*)

- In C, there are two main functions of an asterisk:

  1. Declaration of a pointer variable

[Definition] Asterisk (*)

- In C, there are two main functions of an asterisk:

1. Declaration of a pointer variable

Standard variable: `int p;`

Pointer variable: `int *p;`

[Definition] Asterisk (*)

- In C, there are two main functions of an asterisk:

1. Declaration of a pointer variable

Standard variable: `int p;`

Pointer variable: `int *p;`

2. Dereference (解除參照)

[Definition] Asterisk (*)

- In C, there are two main functions of an asterisk:

1. Declaration of a pointer variable

Standard variable: `int p;`

Pointer variable: `int *p;`

2. Dereference (解除參照)

We can use an asterisk to a pointer (*p) to obtain the value of the pointed variable. Here, the asterisk is an **indirection** operator (間接運算子).

Use an Asterisk (*) for Dereference

C-course-materials/06-Pointers/dereference.c

2. Dereference (解除參照)

We can use an asterisk to obtain the value of the pointed variable (取值).

```
#include <stdio.h>
int main(void){
    int *p;
    int i = 10;
    p = &i;
    printf("The value of i is: %d", *p);
}
```

Use an Asterisk (*) for Dereference

C-course-materials/06-Pointers/dereference.c

2. Dereference (解除參照)

We can use an asterisk to obtain the value of the pointed variable (取值).

```
#include <stdio.h>
int main(void){
    int *p;
    int i = 10;
    p = &i;
    printf("The value of i is: %d", *p);
}
```

Assign the address of i to p

Use an Asterisk (*) for Dereference

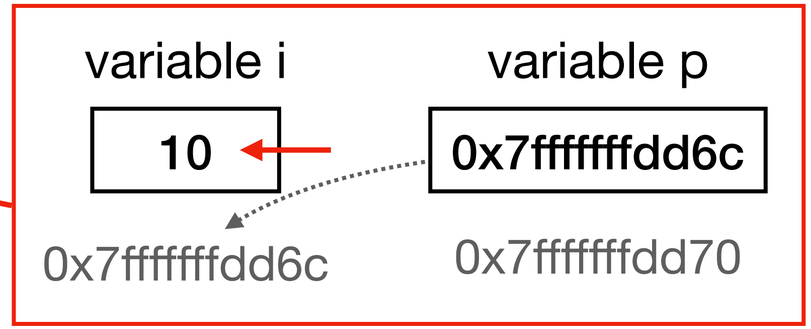
C-course-materials/06-Pointers/dereference.c

2. Dereference (解除參照)

We can use an asterisk to obtain the value of the pointed variable (取值).

```
#include <stdio.h>
int main(void){
    int *p;
    int i = 10;
    p = &i;
    printf("The value of i is: %d", *p);
}
```

Assign the address of i to p



Dereference and Assignment with a Pointer

C-course-materials/06-Pointers/asterisk.c

```c
#include <stdio.h>
int main(void){
    int i = 10;
    int *p = &i;
    int deref = *p; // dereferencing
    printf("%d\n", deref);

    *p = 20; // assigning a new value
    printf("%d", *p);
}
```

Dereference and Assignment with a Pointer

C-course-materials/06-Pointers/asterisk.c

```c
#include <stdio.h>
int main(void){
    int i = 10;
    int *p = &i;
    int deref = *p; // dereferencing
    printf("%d\n", deref);

    *p = 20; // assigning a new value
    printf("%d", *p);
}
```

(Data type)
integer = integer

Dereference and Assignment with a Pointer

C-course-materials/06-Pointers/asterisk.c

```c
#include <stdio.h>
int main(void){
    int i = 10;
    int *p = &i;
    int deref = *p; // dereferencing
    printf("%d\n", deref);

    *p = 20; // assigning a new value
    printf("%d", *p);
}
```

(Data type)
integer = integer

Dereference and Assignment with a Pointer

C-course-materials/06-Pointers/asterisk.c

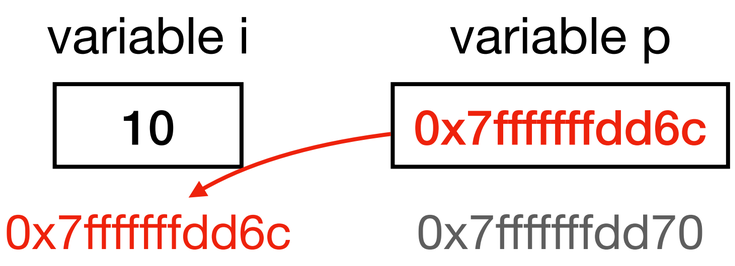
```
#include <stdio.h>
int main(void){
    int i = 10;
    int *p = &i;
    int deref = *p; // dereferencing
    printf("%d\n", deref);

    *p = 20; // assigning a new value
    printf("%d", *p);
}
```

1. Declaration of a pointer variable

[Illustration] Comparison of Pointer Operations

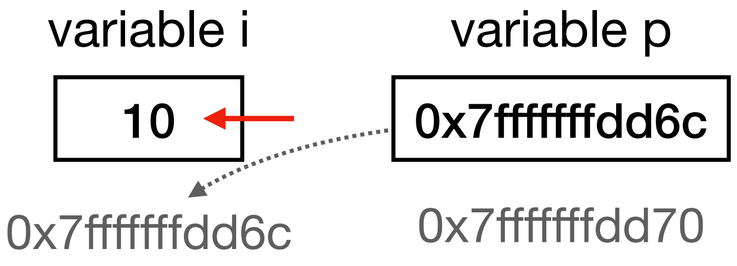
```
int i = 10;
```

```
int *p = &i;
```



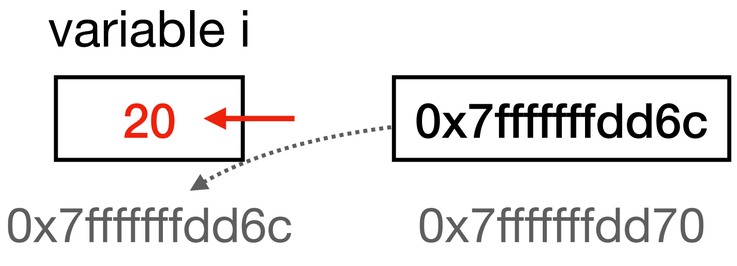
An integer pointer ‘p’ is set to the address of ‘i’.

```
printf("%d", *p);
```



Print the value of the variable pointed by ‘p’.

```
*p = 20;
```

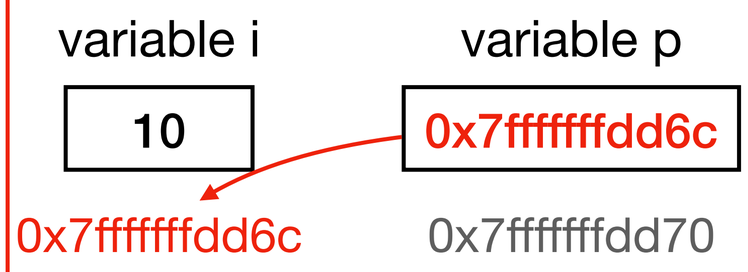


Set the value of the variable pointed by ‘p’ as 20.

[Illustration] Comparison of Pointer Operations

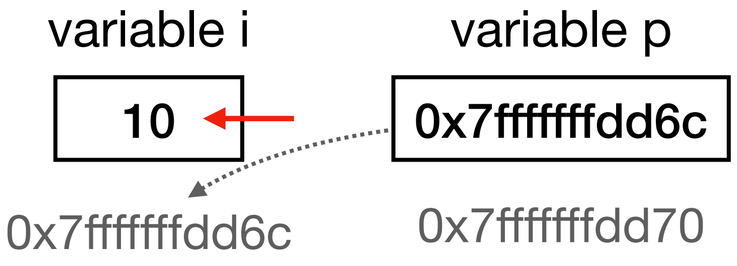
```
int i = 10;
```

```
int *p = &i;
```



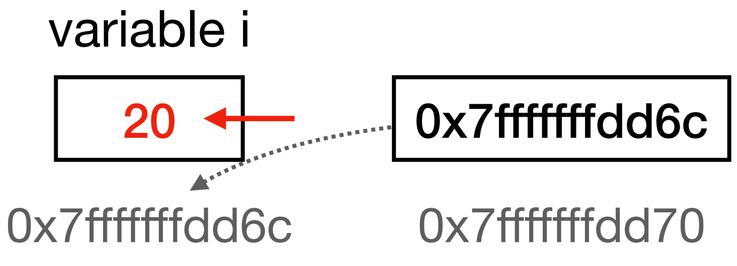
An integer pointer ‘p’ is set to the address of ‘i’.

```
printf("%d", *p);
```



Print the value of the variable pointed by ‘p’.

```
*p = 20;
```

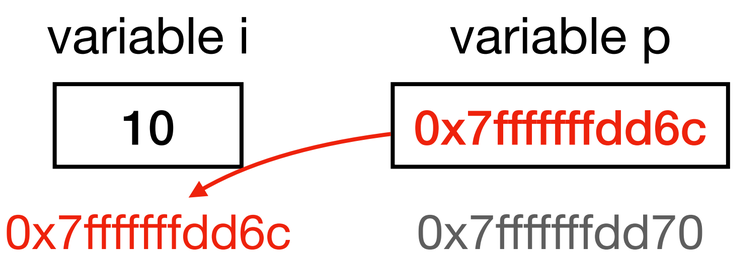


Set the value of the variable pointed by ‘p’ as 20.

[Illustration] Comparison of Pointer Operations

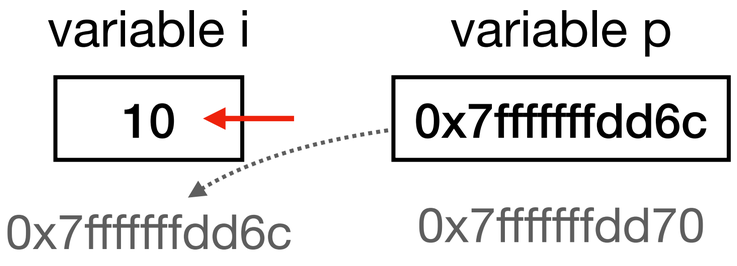
```
int i = 10;
```

```
int *p = &i;
```



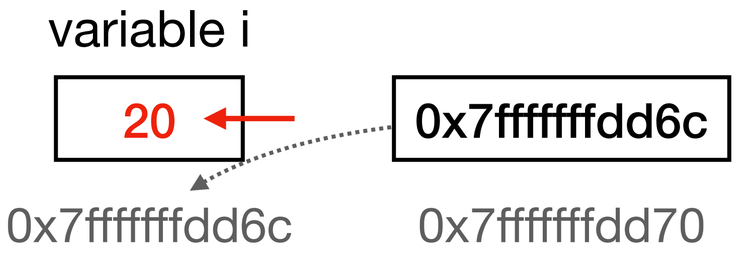
An integer pointer ‘p’ is set to the address of ‘i’.

```
printf("%d", *p);
```



Print the value of the variable pointed by ‘p’.

```
*p = 20;
```

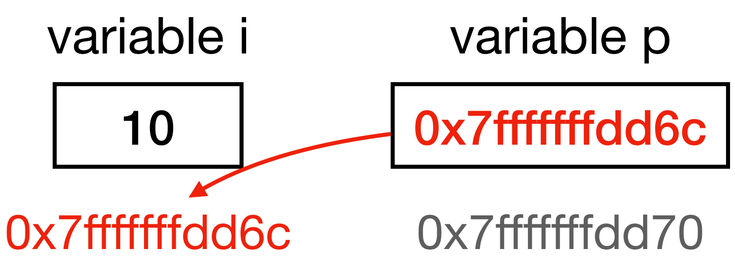


Set the value of the variable pointed by ‘p’ as 20.

[Illustration] Comparison of Pointer Operations

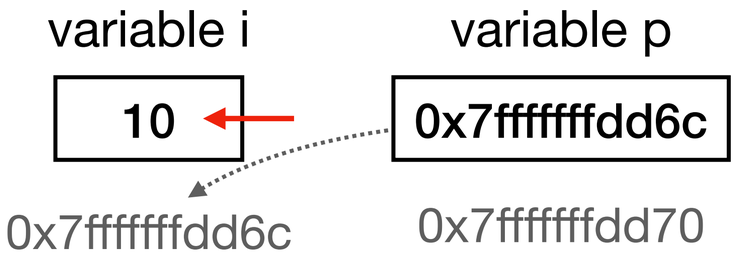
```
int i = 10;
```

```
int *p = &i;
```



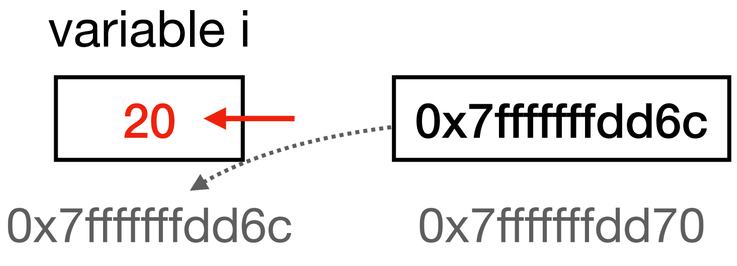
An integer pointer ‘p’ is set to the address of ‘i’.

```
printf("%d", *p);
```



Print the value of the variable pointed by ‘p’.

```
*p = 20;
```



Set the value of the variable pointed by ‘p’ as 20.

[Usage] Summary of & and *

- C provides a pair of operators designed specifically for use with pointers.
  - (取址) To find the address of a variable, we use the & (address) operator.
  - (取值) To gain access to the object that a pointer points to, we use the * (indirection) operator.

[Usage] Summary of stored values

	Standard variable	Pointer variable
Declaration	`int i = 10;`	`int *iptr= &i;`
Variable	Variable i	Variable iptr
Stored value	10	0x7fffffffdd6c
Memory address itself	0x7fffffffdd6c	0x7fffffffdd70

	i	&i (位置運算子)	iptr	&iptr (位置運算子)	*iptr (間接運算子)
Value	10	0x7fffffffdd6c	0x7fffffffdd6c	0x7fffffffdd70	10

Same

Pointer Sizes

C-course-materials/06-Pointers/pointer_sizes.c

```c
#include <stdio.h>
int main(void){
    int i = 10;
    float f = 10.0;
    double d = 10.0;
    int *iptr = &i;
    float *fptr = &f;
    double *dptr = &d;
    printf("Size of the int pointer p is: %d\n", sizeof(iptr));
    printf("Size of the value of the pointed varaiable is: %d\n", sizeof(*iptr));
    printf("Size of the float pointer p is: %d\n", sizeof(fptr));
    printf("Size of the value of the pointed varaiable is: %d\n", sizeof(*fptr));
    printf("Size of the double pointer p is: %d\n", sizeof(dptr));
    printf("Size of the value of the pointed varaiable is: %d", sizeof(*dptr));
}
```

On a 64-bit system, a pointer has a size of 8 bytes for all data types (1 byte = 8 bits.)

This is because that a pointer stores a **memory address** of its pointed variable, and

a memory address is **64 bits wide** on a 64-bit system.

Practical Properties of Pointers

Example1: Exchange Pointers

C-course-materials/06-Pointers/pointer_ops.c

We cannot modify the address of a variable, but we can redirect a pointer.

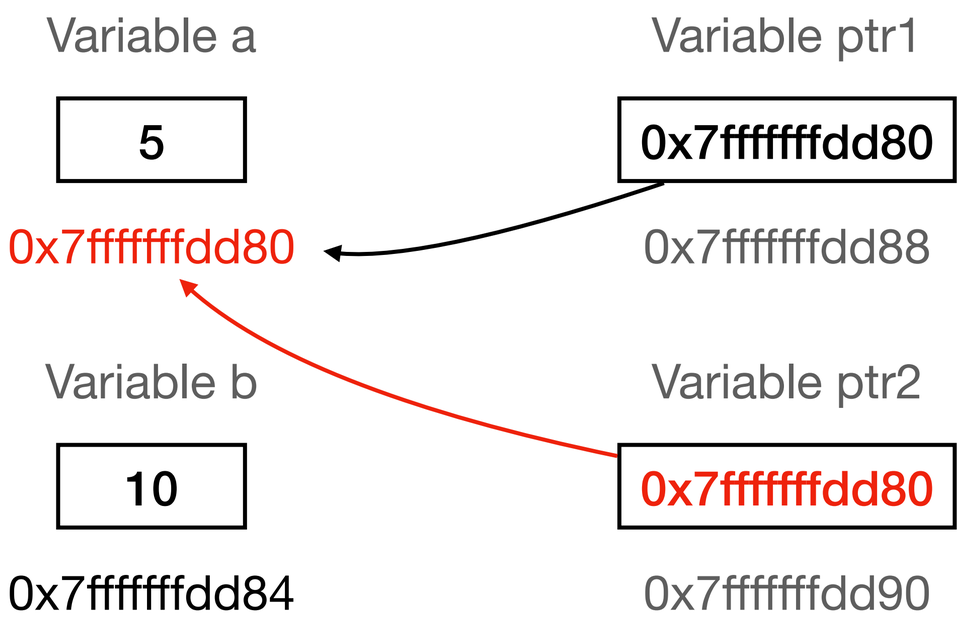
```
int a = 5, b = 10;
printf("&a: %p\n", &a);
printf("&b: %p\n", &b);

int *ptr1, *ptr2;
ptr1 = &a;
ptr2 = &b;
printf("*ptr1: %d\n", *ptr1);
printf("ptr1: %p\n", ptr1);
printf("&ptr1: %p\n", &ptr1);
printf("*ptr2: %d\n", *ptr2);
printf("ptr2: %p\n", ptr2);
printf("&ptr2: %p\n", &ptr2);
ptr2 = ptr1;
printf("*ptr2: %d\n", *ptr2);
printf("ptr2: %p\n", ptr2);
printf("&ptr2: %p\n", &ptr2);
```

Example1: Exchange Pointers

C-course-materials/06-Pointers/pointer_ops.c

We cannot modify the address of a variable, but we can redirect a pointer.

```
int a = 5, b = 10;
printf("&a: %p\n", &a);
printf("&b: %p\n", &b);

int *ptr1, *ptr2;
ptr1 = &a;
ptr2 = &b;
printf("*ptr1: %d\n", *ptr1);
printf("ptr1: %p\n", ptr1);
printf("&ptr1: %p\n", &ptr1);
printf("*ptr2: %d\n", *ptr2);
printf("ptr2: %p\n", ptr2);
printf("&ptr2: %p\n", &ptr2);
ptr2 = ptr1;
printf("*ptr2: %d\n", *ptr2);
printf("ptr2: %p\n", ptr2);
printf("&ptr2: %p\n", &ptr2);
```



Example1: Exchange Pointers

C-course-materials/06-Pointers/pointer_ops.c

Output:

```
&a: 0x7fffffffdd80
&b: 0x7fffffffdd84
*ptr1: 5
ptr1: 0x7fffffffdd80
&ptr1: 0x7fffffffdd88
*ptr2: 10
ptr2: 0x7fffffffdd84
&ptr2: 0x7fffffffdd90
*ptr2: 5
ptr2: 0x7fffffffdd80
&ptr2: 0x7fffffffdd90
```

Example2: Overwrite Values of Pointed Variables

C-course-materials/06-Pointers/overwrite_values.c
C-course-materials/06-Pointers/overwrite_values_pointers.c

Using Pointer

```c
#include <stdio.h>
int main(void){
    int a = 5, b = 10;
    int *ptr1 = &a;
    *ptr1 = b;
    printf("a: %d\n", a);
    printf("*ptr1: %d", *ptr1);
}
```

Output:

```
a: 10
*ptr1: 10
```

Not Using Pointer

```c
#include <stdio.h>
int main(void){
    int a = 5, b = 10;
    int new_a;
    new_a = b;
    printf("a: %d\n", a);
    printf("new_a: %d", new_a);
}
```

Output:

```
a: 5
new_a: 10
```

A pointer only points to a variable with the same type

C-course-materials/06-Pointers/same_type.c

```c
#include <stdio.h>
int main(void){
    int i = 10;
    float float_var = 3.14;
    int *iptr = &i;
    iptr = &float_var; // Warning: Incompatible types
    printf("%d\n", *iptr);
    printf("%f\n", *iptr);
}
```

A pointer only points to a variable with the same type

C-course-materials/06-Pointers/same_type.c

```c
#include <stdio.h>
int main(void){
    int i = 10;
    float float_var = 3.14;
    int *iptr = &i;
    iptr = &float_var; // Warning: Incompatible types
    printf("%d\n", *iptr);
    printf("%f\n", *iptr);
}
```

Redirection failed.

Dereferences show wrong results.

[Important Notes] Properties of Pointers

- Two pointers can point the same variable.
- A pointer can be redirected to point other variables.
  - The memory address of a variable itself cannot be changed
- Generally, a pointer with a specific data type can only point to the variable with the same data type

Complicated Pointer Operations

C-course-materials/06-Pointers/complicated_pointer_ops.c

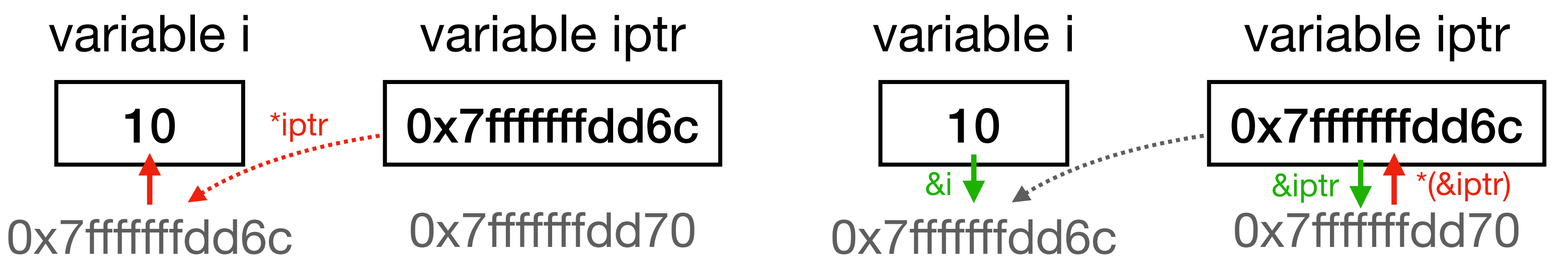
```c
#include <stdio.h>
int main(void){
    int i = 10;
    int *iptr = &i;
    // Operation 1
    printf("&(*iptr) = %p\n", &(*iptr));
    // Operation 2
    printf("*(&iptr) = %p\n", *(&iptr));
    // Operation 3
    printf("*(*(&iptr)) = %d\n", *(*(&iptr)));
    // Operation 4
    printf("*(&(*iptr)) = %d\n", *(&(*iptr)));
    // Operation 5
    printf("&(*(&iptr)) = %p\n", &(*(&iptr)));
}
```

[Illustration] Legends and Notations

C-course-materials/06-Pointers/complicated_pointer_ops.c

```
int i = 10;
int *iptr = &i;
```

Red arrow(s)	Dereference (*)
Green arrow	Get address (&)
←- - - -	Pointing path



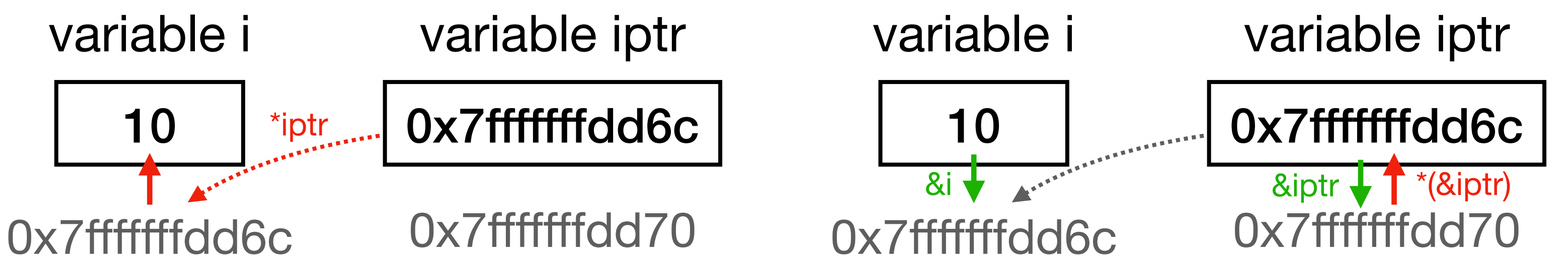
[Illustration] Legends and Notations

C-course-materials/06-Pointers/complicated_pointer_ops.c

```
int i = 10;
int *iptr = &i;
```

Red arrow(s)	Dereference (*)
Green arrow	Get address (&)
←- - - -	Pointing path

上樓取值，下樓取址
指標遇* (e.g.,*iptr) 走pointing



[Illustration] Operation 1

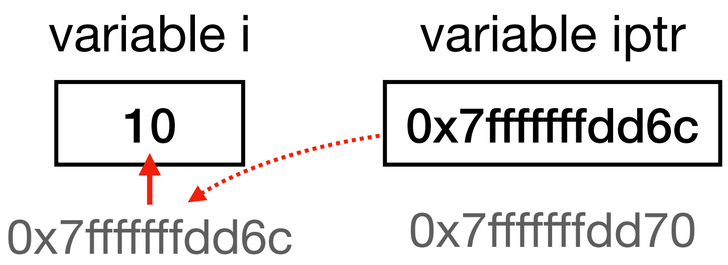
C-course-materials/06-Pointers/complicated_pointer_ops.c

```
printf("&(*iptr) = %p\n", &(*iptr));
```

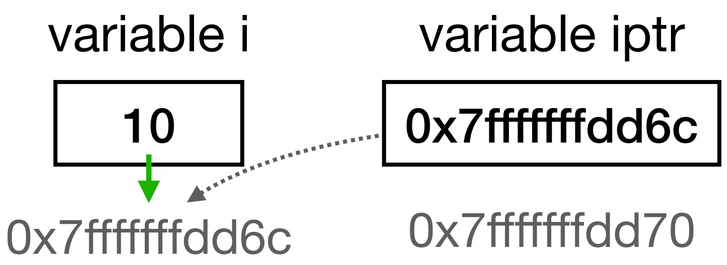
The output is **the address of the pointed variable**.

Red arrow(s)	Dereference (*)
Green arrow	Get address (&)
←- - - -	Pointing path

Step1: *iptr



Step2: &



[Illustration] Operation 2

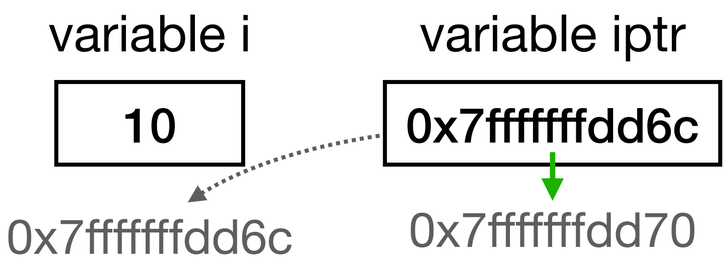
C-course-materials/06-Pointers/complicated_pointer_ops.c

```
printf("*(&iptr) = %p\n", *(&iptr));
```

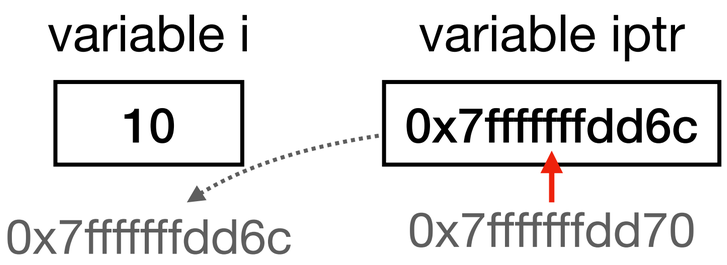
The output is **the stored address of the pointer**.

Red arrow(s)	Dereference (*)
Green arrow	Get address (&)
←- - - -	Pointing path

Step1: &iptr



Step2: *



[Illustration] Operation 3

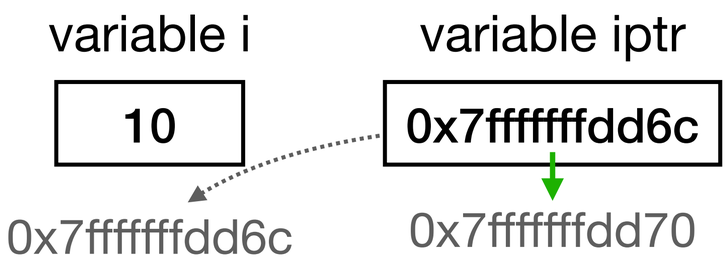
C-course-materials/06-Pointers/complicated_pointer_ops.c

```
printf("*(*(&iptr)) = %d\n", *(*(&iptr)));
```

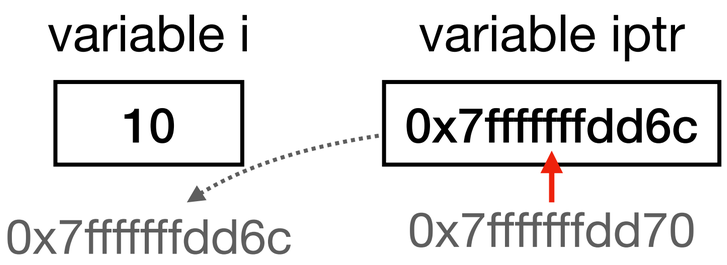
The output is **the value of the pointed variable**.

Red arrow(s)	Dereference (*)
Green arrow	Get address (&)
←- - - -	Pointing path

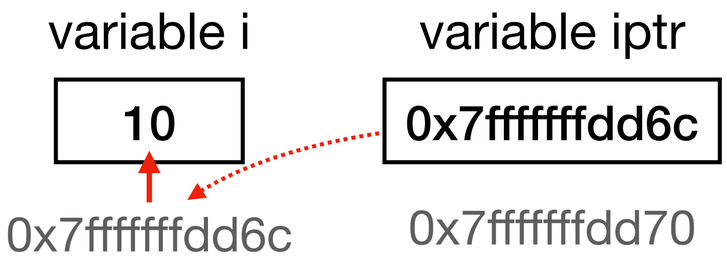
Step1: &iptr



Step2: *



Step3: *



[Illustration] Operation 4

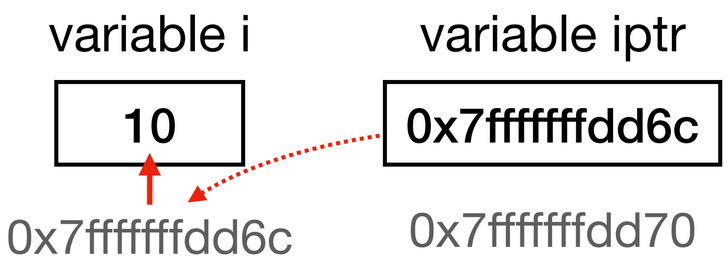
C-course-materials/06-Pointers/complicated_pointer_ops.c

```
printf("*(&(*iptr)) = %d\n", *(&(*iptr)));
```

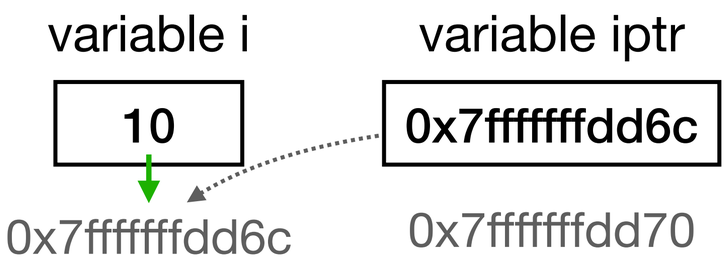
The output is **the value of the pointed variable**.

Red arrow(s)	Dereference (*)
Green arrow	Get address (&)
←- - - -	Pointing path

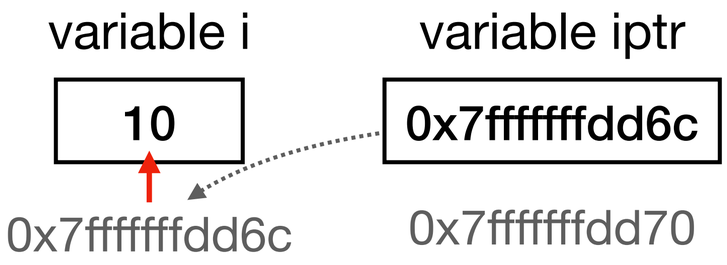
Step1: *iptr



Step2: &



Step3: *



[Illustration] Operation 5

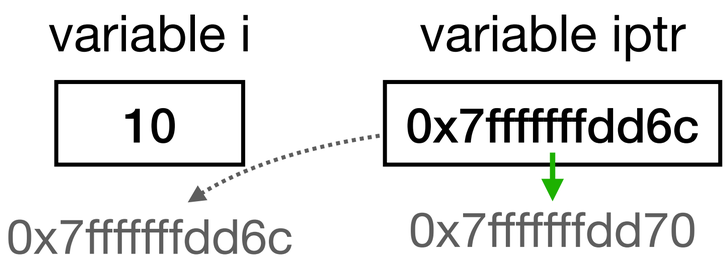
C-course-materials/06-Pointers/complicated_pointer_ops.c

```
printf("&(*(&iptr)) = %p\n", &(*(&iptr)));
```

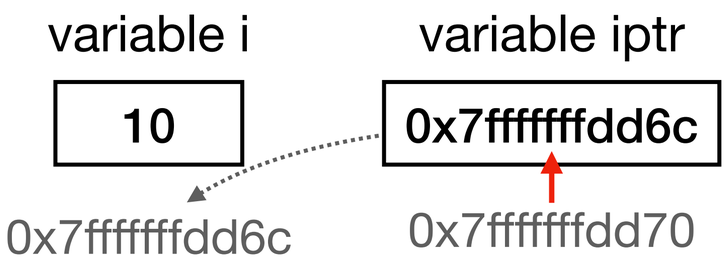
The output is **the value of the pointed variable**.

Red arrow(s)	Dereference (*)
Green arrow	Get address (&)
←- - - -	Pointing path

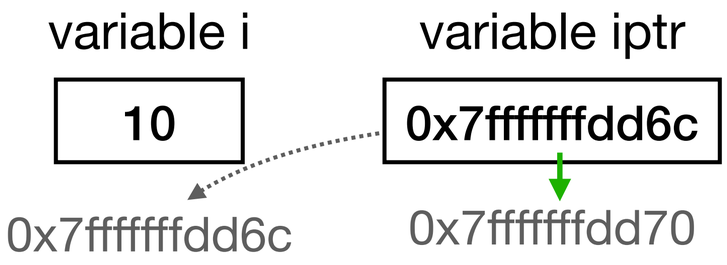
Step1: &iptr



Step2: *



Step3: &



運算子優先順序說明

C-course-materials/slides/data_types_op.pdf (p.37)

優先順序	Operator	Meaning	連在一起用？
1	( )	大於	由左至右
2	[ ]	小於	由左至右
3	! + - * &	非、取正負、解除參照、取位址	由右至左
4	++ --	遞增、遞減	由右至左
5	* / %	算數運算子	由左至右
6	+ -	算數運算子	由左至右
7	> >= < <=	關係運算子	由左至右
8	== !=	關係運算子	由左至右
9	&&	邏輯運算子	由左至右
10	||	邏輯運算子	由左至右
11	=	設定運算子	由右至左

[Declaration] Pass a Pointer to a Function

- A function prototype is:

```
return_type func_name(type1 *, type2 *, ...);
```

- Purposes:
  1. **Type Checking:** Help the compiler **check the correctness of data types** when you use a function in the main function.
  2. **Function Declaration**: Allows function calls before the function is defined.

- You can also write a function prototype as the following to increase readability:

```
return_type func_name(type1 *param1, type2 *param2, ...);
```

Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```

Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

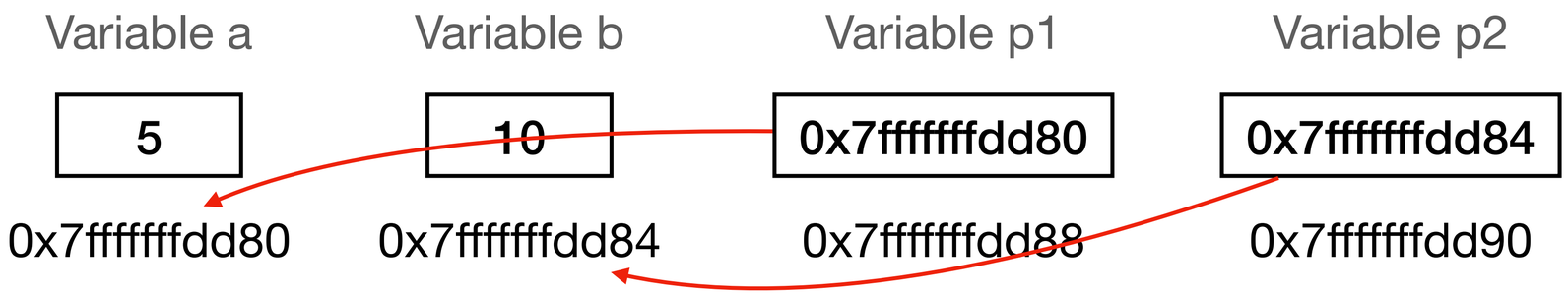
```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```

Variable a	Variable b
5	10
0x7fffffffdd80	0x7fffffffdd84

Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

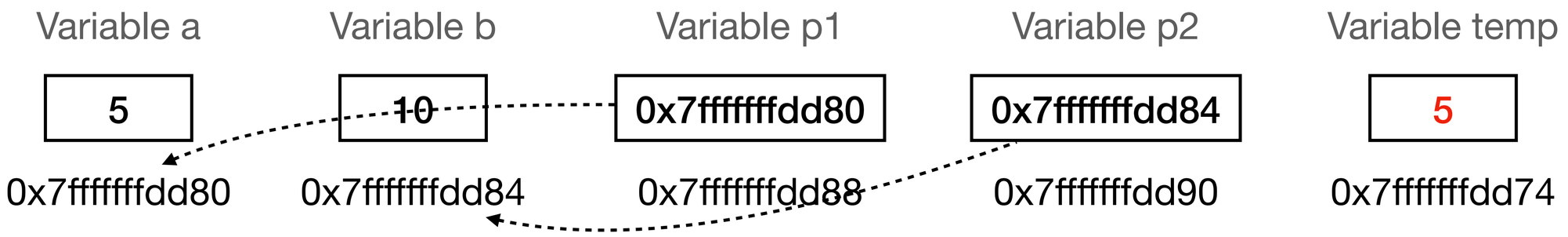
```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```



Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

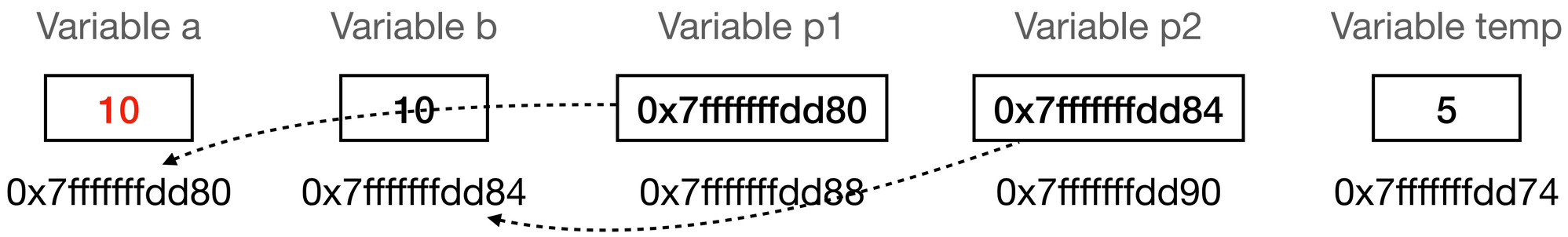
```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```



Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

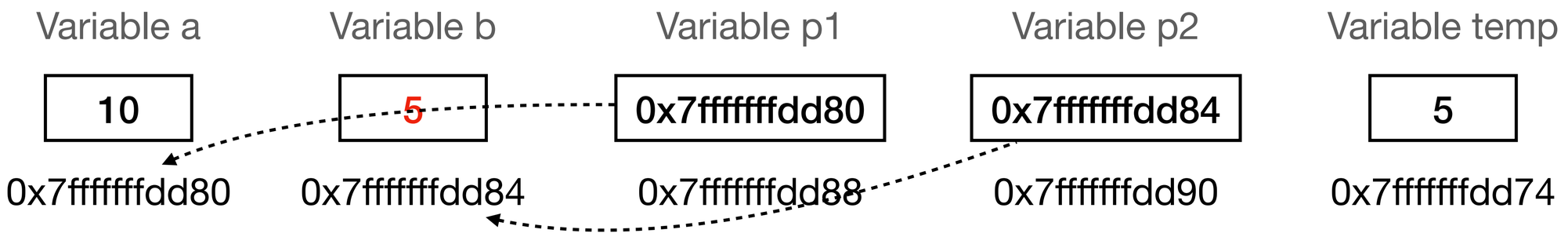
```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```



Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```



Swap two variables inside a function

C-course-materials/06-Pointers/swap_values.c

```c
#include <stdio.h>
void swap(int *p1, int *p2){
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
int main(void){
    int a = 5, b = 10;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
}
```

Variable a	Variable b
10	5
0x7fffffffdd80	0x7fffffffdd84