Practical Programming

The C Language : Pointers

David Bouchet

david.bouchet.epita@gmail.com

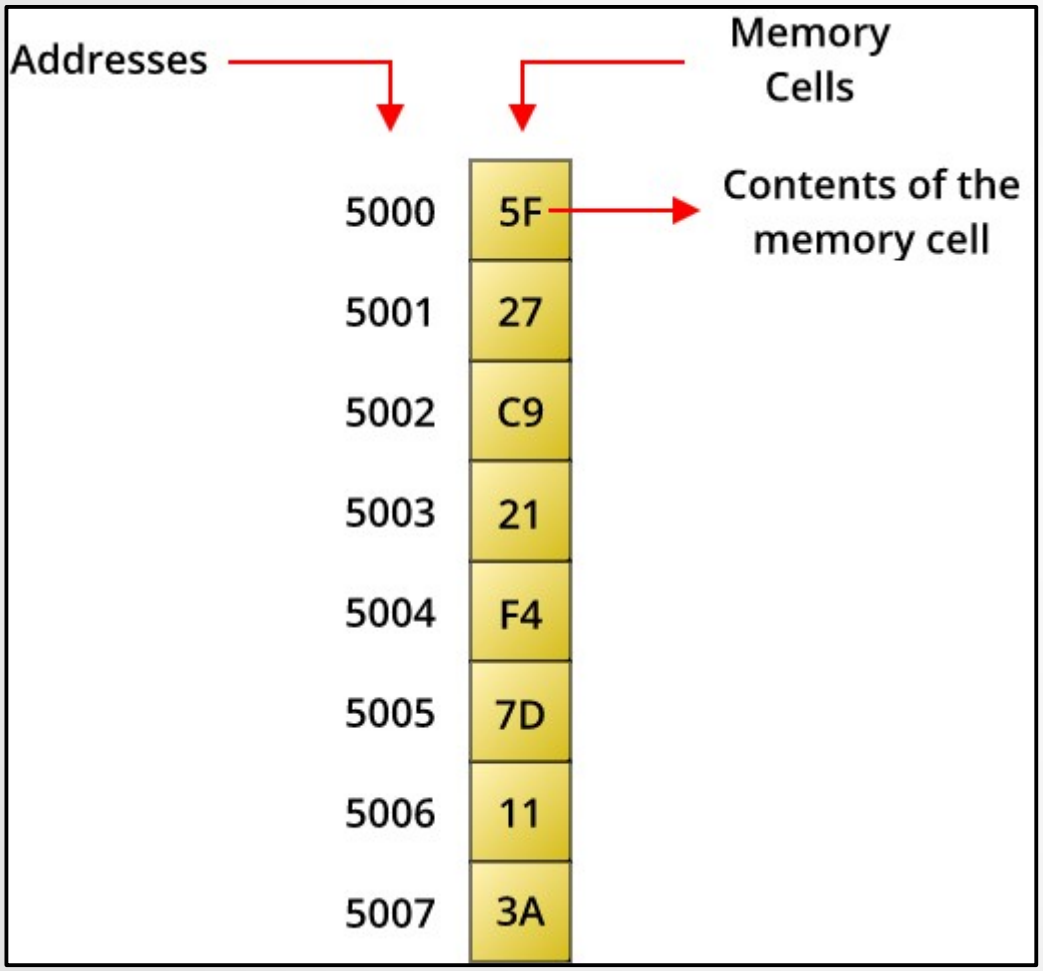
Pointers

- Pointers are probably one of the most important concepts in programming.
- Pointers are also an unsafe tool. Most software failures stem from pointer issues.
- Pointers also seem to be the hardest concept to learn.

Pointers

- Pointers hold memory addresses.
- Memory addresses are similar to array indexes.
- They are fixed-length unsigned integers.
- They point to specific memory cells.

Addresses and Memory Cells

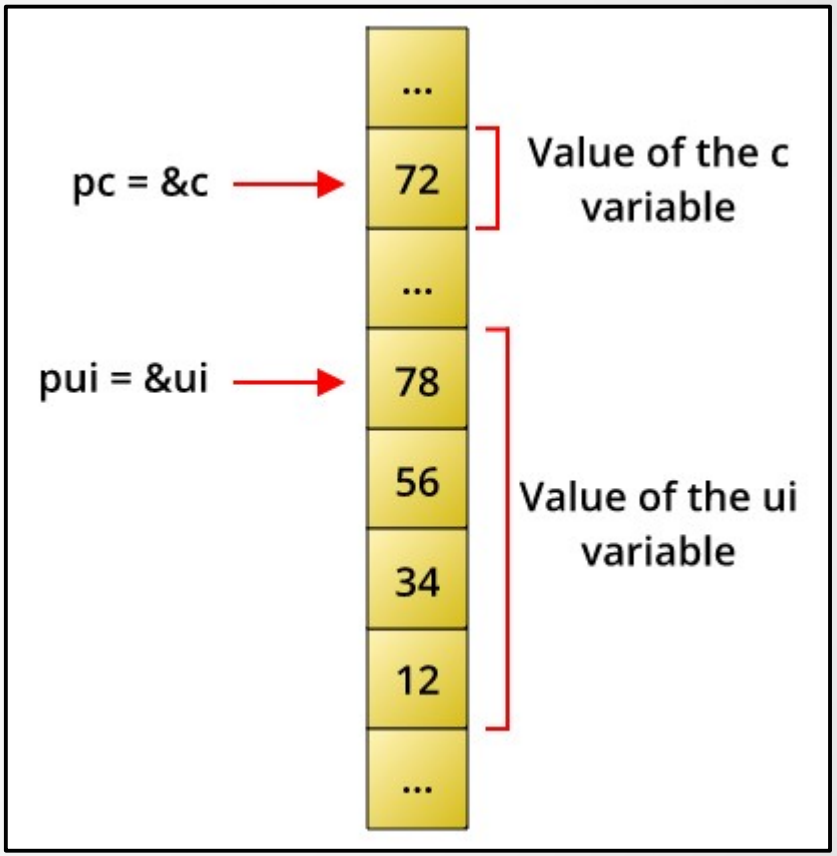


The address 5000_{16} contains $5F_{16}$
The address 5001_{16} contains 27_{16}
…

The address 5007_{16} contains $3A_{16}$

Declaration and Initialization

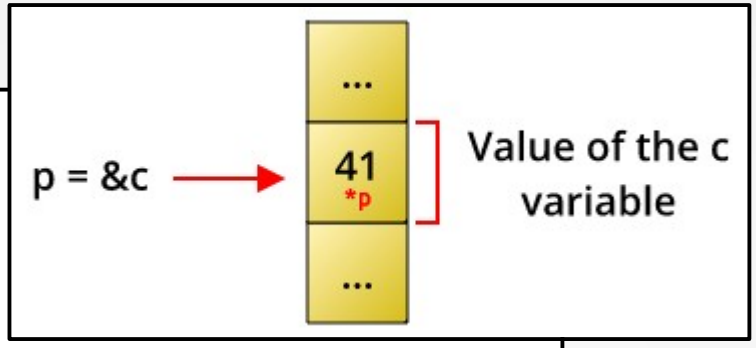
```
char c = 0x72;
unsigned int ui = 0x12345678;

// Declare a pointer to a char variable.
char *pc;

// Initialize the pointer.
// pc holds the address of c.
// pc points to the c variable in memory.
pc = &c;

// Declare and initialize a pointer to
// an unsigned int variable.
// pui holds the address of ui.
// pui points to the ui variable in memory.
unsigned int *pui = &ui;
```



The address of ***c*** is denoted by ***&c***.
The address of ***ui*** is denoted by ***&ui***.

Dereferencing Pointers (1)

```c
char c = 'A';
char *p = &c;

// Print c as a character.
printf("c  = %c\n", c);

// Print c as an 8-bit unsigned integer in hexadecimal.
printf("c  = 0x%hhx\n", c);

// Print the address of c in hexadecimal.
printf("p  = %p\n", p);

// Print the contents of p (i.e. c) as a character.
// The pointer is dereferenced: *p
printf("*p = %c\n", *p);

// Print the contents of p (i.e. c)
// as an 8-bit unsigned integer in hexadecimal.
// The pointer is dereferenced: *p
printf("*p = 0x%hhx\n", *p);
```




```
c   = A
c   = 0x41
p   = 0x7fff1815417f
*p  = A
*p  = 0x41
```

The value pointed to by *p* is denoted by ***p***.
p is then equivalent to *c*.

Dereferencing Pointers (2)

```c
unsigned int i = 0x12345678;
unsigned int *p = &i;

printf(" i = 0x%x\n", i);
printf(" p = %p\n", p);
printf("*p = 0x%x\n", *p);

char *q = (char *)p;

printf("--------------------\n");
printf("      q -> %hhx <- p\n", *(q));
printf(" q + 1 -> %hhx\n", *(q + 1));
printf(" q + 2 -> %hhx\n", *(q + 2));
printf(" q + 3 -> %hhx\n", *(q + 3));
```

```
 i = 0x12345678
 p = 0x7ffe6bc5e764
*p = 0x12345678
--------------------
      q -> 78 <- p
 q + 1 -> 56
 q + 2 -> 34
 q + 3 -> 12
```

Dereferencing Pointers (3)

```c
unsigned int i = 0x12345678;

printf("i = 0x%x\n", i);

char *q = (char *)&i;

*q = 0xaa;
printf("i = 0x%x\n", i);

*(q + 1) = 0xbb;
printf("i = 0x%x\n", i);

*(q + 2) = 0xcc;
printf("i = 0x%x\n", i);

*(q + 3) = 0xdd;
printf("i = 0x%x\n", i);
```

```
i = 0x12345678
i = 0x123456aa
i = 0x1234bbaa
i = 0x12ccbbaa
i = 0xddccbbaa
```

Common Mistakes

- Dereferencing uninitialized pointers
- Out-of-bound access
- Buffer overflow
- Use after deallocations

Common Mistakes – Example

```
short *p;

printf("p = %p\n", p);

*p = 0x1234;
```

```
p = (nil)
Segmentation fault (core dumped)
```

Passing Pointers as Parameters (1)

```c
int main()
{
    int x = 1;
    int y = 9;

    printf("x = %i, y = %i\n", x, y);

    swap(x, y);
    printf("x = %i, y = %i\n", x, y);

    pswap(&x, &y);
    printf("x = %i, y = %i\n", x, y);

    return 0;
}
```

```
x = 1, y = 9
x = 1, y = 9
x = 9, y = 1
```

```c
void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

```c
void pswap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

Passing Pointers as Parameters (2)

```c
int euclidean_div(int a, int b, int *q, int *r)
{
    if (b == 0)
        return 1;

    *q = a / b;
    *r = a % b;

    return 0;
}
```

```c
int main()
{
    int x, y;
    int q, r, error;

    x = 100;

    for (y = 0; y < 10; y += 2)
    {
        error = euclidean_div(x, y, &q, &r);

        if (error)
            printf("%i / %i = Error (division by zero)\n", x, y);
        else
            printf("%i / %i = %i it remains %i\n", x, y, q, r);
    }

    return 0;
}
```

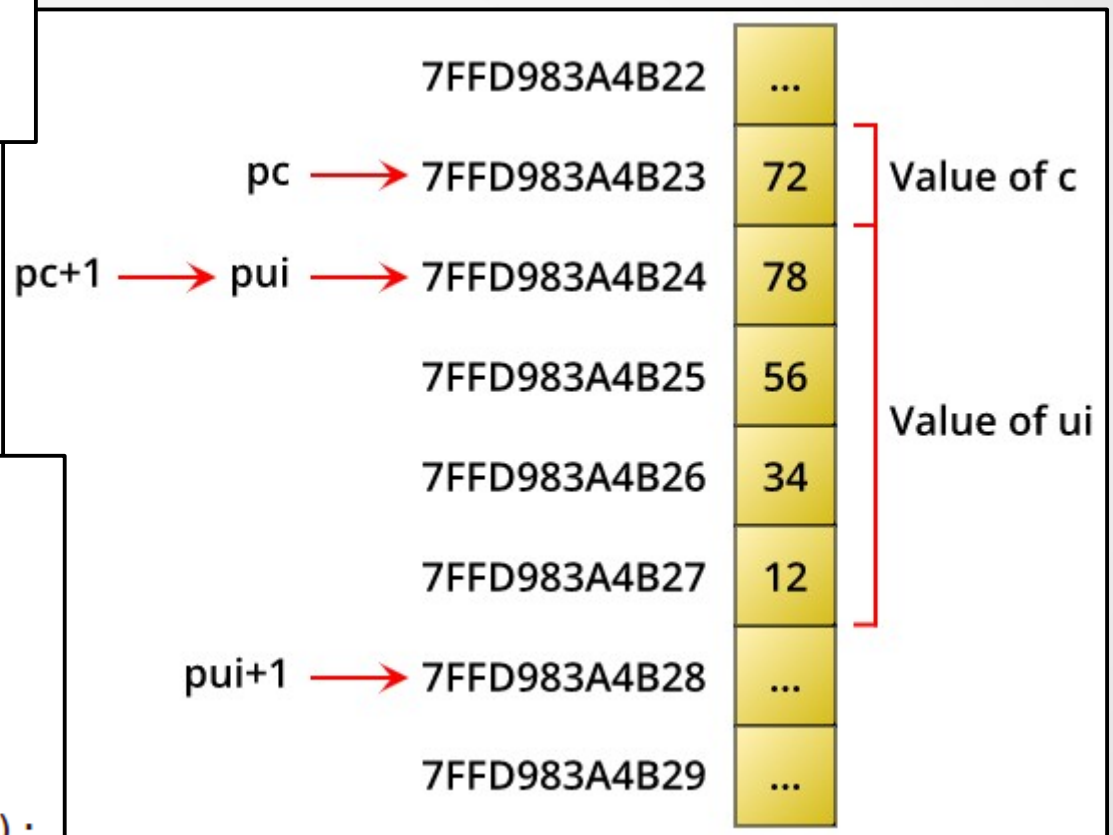
```
100 / 0 = Error (division by zero)
100 / 2 = 50 it remains 0
100 / 4 = 25 it remains 0
100 / 6 = 16 it remains 4
100 / 8 = 12 it remains 4
```

Pointer Arithmetic (1)

- Pointers are integers.
- Additions and subtractions are allowed on pointers.
- ***p + 1*** does not point to the next byte but to the next value.
- The number of bytes for a value depends on its type.

Pointer Arithmetic (2)

```
     pc = 0x7ffd983a4b23
 pc + 1 = 0x7ffd983a4b24
    pui = 0x7ffd983a4b24
pui + 1 = 0x7ffd983a4b28
```




```c
char c = 0x72;
unsigned int ui = 0x12345678;

char *pc = &c;
unsigned int *pui = &ui;

printf("     pc = %p\n", pc);
printf(" pc + 1 = %p\n", pc + 1);
printf("    pui = %p\n", pui);
printf("pui + 1 = %p\n", pui + 1);
```

Pointer Arithmetic (3)

- Operation between pointers of different types are not allowed.
- The ***void**** type can’t be used in pointer arithmetic (because the ***void*** type has no size).

Pointers to Arrays (1)

- An array variable is a constant pointer.
- It points to a memory location that contains values of the same size.

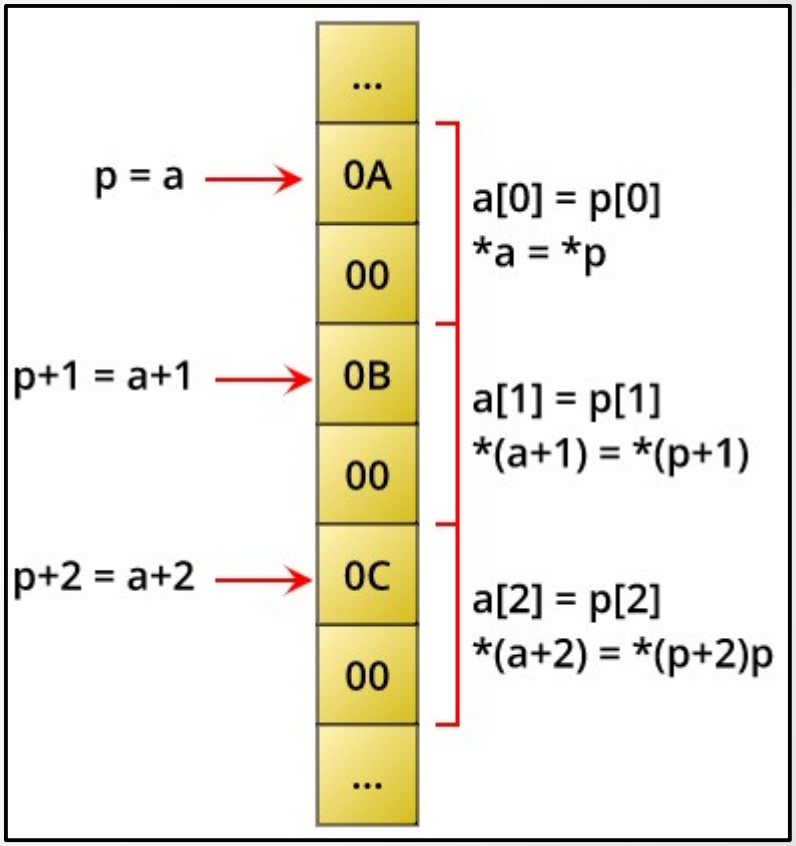
```
short a[3] = { 10, 11, 12 };
short *p = a;
```

***a* and *p* hold the same address value but:**

- ***a*** points to an array of size 3.
- ***p*** points to the first value of ***a*** *(p = &a[0])*.
- The size of ***a*** is the size of the array in bytes (i.e. 6).
- The size of ***p*** is the size of a pointer (it depends on the architecture).
- ***a*** is constant.
- ***p*** is not constant.

Pointers to Arrays (2)

```
short a[3] = { 10, 11, 12 };
short *p = a;

for (size_t i = 0; i < 3; i++)
{
    printf("a[%zu] = %hi  |  ", i, a[i]);
    printf("*(p + %zu) = %hi  |  ", i, *(p + i));
    printf("p[%zu] = %hi  |  ", i, p[i]);
    printf("*(a + %zu) = %hi\n", i, *(a + i));
}
```




```
a[0] = 10  |  *(p + 0) = 10  |  p[0] = 10  |  *(a + 0) = 10
a[1] = 11  |  *(p + 1) = 11  |  p[1] = 11  |  *(a + 1) = 11
a[2] = 12  |  *(p + 2) = 12  |  p[2] = 12  |  *(a + 2) = 12
```

Pointers to Arrays (3)

```
short a[3] = { 10, 11, 12 };
short *p = a;

for (size_t i = 0; i < 3; i++)
{
    printf("%p -> 0x%04hx\n", p, *p);
    p++;
}

// for (size_t i = 0; i < 3; i++)
// {
//     printf("%p -> 0x%04hx\n", a, *a);
//     a++; // NOT ALLOWED! a IS CONSTANT!
// }
```

```
0x7ffd40a6e4c0 -> 0x000a
0x7ffd40a6e4c2 -> 0x000b
0x7ffd40a6e4c4 -> 0x000c
```

- We cannot replace ***p*** by ***a*** in the for loop.
- ***a++*** is not allowed because ***a*** is constant.

Pointers to Arrays (4)

```c
short a[3] = { 10, 11, 12 };
short *p = a;

printf("Size of the array in bytes:\n");
printf("(a points to the array.)\n");
printf("sizeof(a) = %zu\n", sizeof(a));

printf("----------------------\n");

printf("Size of the p pointer:\n");
printf("(p points to the first element.)\n");
printf("sizeof(p) = %zu\n", sizeof(p));

printf("----------------------\n");

printf("Size of one element:\n");
printf("sizeof(*a) = %zu\n", sizeof(*a));
printf("sizeof(*p) = %zu\n", sizeof(*p));

printf("----------------------\n");

printf("Number of elements:\n");
printf("sizeof(a)/sizeof(*a) = %zu\n", sizeof(a)/sizeof(*a));
```

```
Size of the array in bytes:
(a points to the array.)
sizeof(a) = 6
----------------------
Size of the p pointer:
(p points to the first element.)
sizeof(p) = 8
----------------------
Size of one element:
sizeof(*a) = 2
sizeof(*p) = 2
----------------------
Number of elements:
sizeof(a)/sizeof(*a) = 3
```

Pointers to Arrays (5)

```c
int sum(short a[], size_t length)
{
    int s = 0;

    for (size_t i = 0; i < length; i++)
        s += a[i];

    return s;
}
```

```c
int psum(short *a, size_t length)
{
    int s = 0;

    short *end = a + length;
    while (a != end)
        s += *(a++);

    return s;
}
```

```c
int main()
{
    short a[] = { 10, 11, 12, 13, 14 };

    printf(" sum(a, 5) = %i\n", sum(a, 5));
    printf("psum(a, 5) = %i\n", psum(a, 5));

    return 0;
}
```

```
 sum(a, 5) = 60
psum(a, 5) = 60
```

Pointers to Pointers

```
 &a = 0x7fff81c61030 -> 0x11223344     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
----------------------------------------
```

```c
int a = 0x11223344;
int b = 0xaabbccdd;

int *p1 = &a;
int **p2 = &p1;

PRINT();

*p1 = 0x0;            PRINT();
**p2 = 0x12345678;    PRINT();
*p2 = &b;             PRINT();
**p2 = 0;             PRINT();

return 0;
```

```c
#define PRINT() \
printf(" &a = %p -> 0x%08x     = a\n", &a, a);\
printf(" &b = %p -> 0x%08x     = b\n", &b, b);\
printf("&p1 = %p -> %p = p1\n", &p1, p1);\
printf("&p2 = %p -> %p = p2\n", &p2, p2);\
printf("----------------------------------------\n")
```

Pointers to Pointers

```
 &a = 0x7fff81c61030 -> 0x11223344     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
----------------------------------------
 &a = 0x7fff81c61030 -> 0x00000000     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
----------------------------------------
```

```c
int a = 0x11223344;
int b = 0xaabbccdd;

int *p1 = &a;
int **p2 = &p1;

PRINT();

*p1 = 0x0;            PRINT();
**p2 = 0x12345678;    PRINT();
*p2 = &b;             PRINT();
**p2 = 0;             PRINT();

return 0;
```

```c
#define PRINT() \
printf(" &a = %p -> 0x%08x     = a\n", &a, a);\
printf(" &b = %p -> 0x%08x     = b\n", &b, b);\
printf("&p1 = %p -> %p = p1\n", &p1, p1);\
printf("&p2 = %p -> %p = p2\n", &p2, p2);\
printf("----------------------------------------\n")
```

Pointers to Pointers

```
 &a = 0x7fff81c61030 -> 0x11223344     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
---------------------------------------
 &a = 0x7fff81c61030 -> 0x00000000     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
---------------------------------------
 &a = 0x7fff81c61030 -> 0x12345678     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
---------------------------------------
```

```c
int a = 0x11223344;
int b = 0xaabbccdd;

int *p1 = &a;
int **p2 = &p1;

PRINT();

*p1 = 0x0;            PRINT();
**p2 = 0x12345678;    PRINT();
*p2 = &b;             PRINT();
**p2 = 0;             PRINT();

return 0;
```

```c
#define PRINT() \
printf(" &a = %p -> 0x%08x     = a\n", &a, a);\
printf(" &b = %p -> 0x%08x     = b\n", &b, b);\
printf("&p1 = %p -> %p = p1\n", &p1, p1);\
printf("&p2 = %p -> %p = p2\n", &p2, p2);\
printf("---------------------------------------\n")
```

Pointers to Pointers

```
 &a = 0x7fff81c61030 -> 0x11223344     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
---------------------------------------
 &a = 0x7fff81c61030 -> 0x00000000     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
---------------------------------------
 &a = 0x7fff81c61030 -> 0x12345678     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
---------------------------------------
 &a = 0x7fff81c61030 -> 0x12345678     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61034 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
---------------------------------------
```

```c
int a = 0x11223344;
int b = 0xaabbccdd;

int *p1 = &a;
int **p2 = &p1;

PRINT();

*p1 = 0x0;            PRINT();
**p2 = 0x12345678;    PRINT();
*p2 = &b;             PRINT();
**p2 = 0;             PRINT();

return 0;
```

```c
#define PRINT() \
printf(" &a = %p -> 0x%08x     = a\n", &a, a);\
printf(" &b = %p -> 0x%08x     = b\n", &b, b);\
printf("&p1 = %p -> %p = p1\n", &p1, p1);\
printf("&p2 = %p -> %p = p2\n", &p2, p2);\
printf("---------------------------------------\n")
```

Pointers to Pointers

```
 &a = 0x7fff81c61030 -> 0x11223344     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
----------------------------------------
 &a = 0x7fff81c61030 -> 0x00000000     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
----------------------------------------
 &a = 0x7fff81c61030 -> 0x12345678     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61030 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
----------------------------------------
 &a = 0x7fff81c61030 -> 0x12345678     = a
 &b = 0x7fff81c61034 -> 0xaabbccdd     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61034 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
----------------------------------------
 &a = 0x7fff81c61030 -> 0x12345678     = a
 &b = 0x7fff81c61034 -> 0x00000000     = b
&p1 = 0x7fff81c61038 -> 0x7fff81c61034 = p1
&p2 = 0x7fff81c61040 -> 0x7fff81c61038 = p2
----------------------------------------
```

```c
int a = 0x11223344;
int b = 0xaabbccdd;

int *p1 = &a;
int **p2 = &p1;

PRINT();

*p1 = 0x0;            PRINT();
**p2 = 0x12345678;    PRINT();
*p2 = &b;             PRINT();
**p2 = 0;             PRINT();

return 0;
```

```c
#define PRINT() \
printf(" &a = %p -> 0x%08x     = a\n", &a, a);\
printf(" &b = %p -> 0x%08x     = b\n", &b, b);\
printf("&p1 = %p -> %p = p1\n", &p1, p1);\
printf("&p2 = %p -> %p = p2\n", &p2, p2);\
printf("----------------------------------------\n")
```