9. Pointers

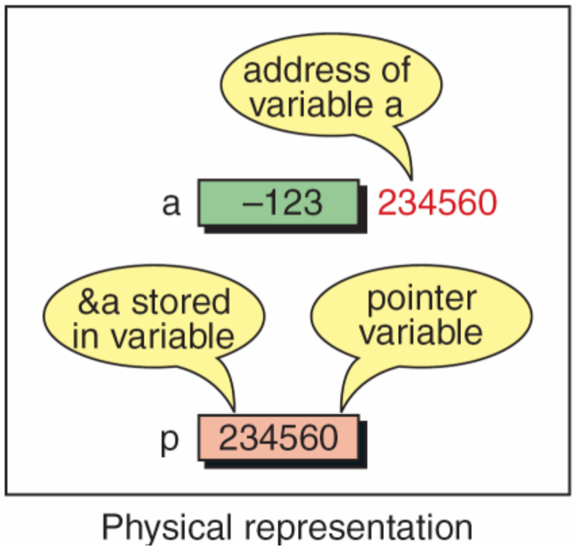
[ECE10002] C Programming

Agenda

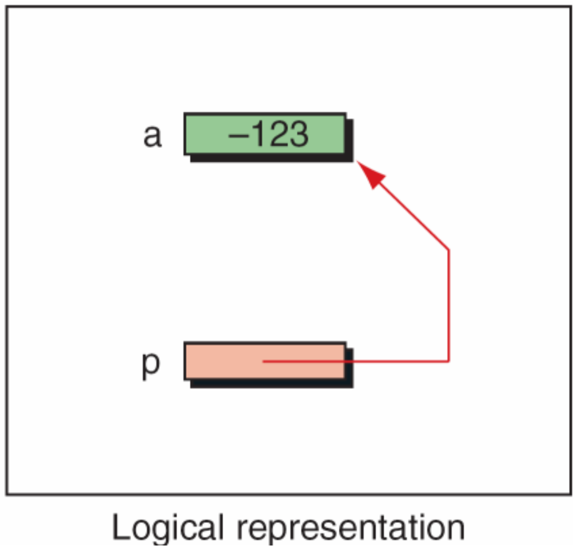
- Introduction
- Pointer for Inter-Function Communication
- Pointers to Pointers
- Compatibility
- (Lvalue and Rvalue)

Pointer Variables

- Pointer variable: a variable to store an address



Physical representation



Logical representation

  - We can access value of **a** through **p**, but the opposite is impossible.

Using Pointer Variables

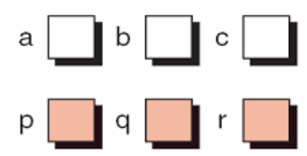
- Declaration
  - int *pa;
- Extracting address of a variable (address operator &)
  - pa = &a;
- Dereferencing (dereferencing operator *)
  - *pa = 89;
  - c = *pa * 2;
- Address operator vs. dereferencing operator
  - & is inverse of *

    Ex) *&a ≡ a; // * and & cancel each other

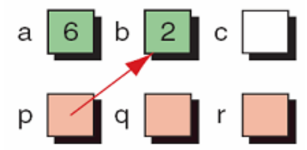
    cf. How about &*a ?

Example

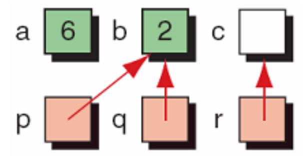
```
int a, b, c;
int *p, *q, *r;
```



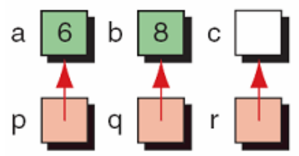
```
a = 6;
b = 2;
p = &b;
```



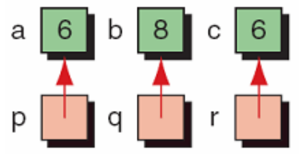
```
q = p;
r = &c;
```



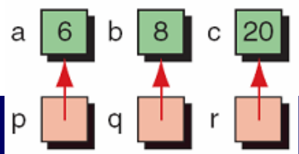
```
p = &a;
*q = 8;
```



```
*r = *p;
```



```
*r = a + *q + *&c;
```



Agenda

- Introduction
- <u>Pointer for Inter-Function Communication</u>
- Pointers to Pointers
- Compatibility
- (Lvalue and Rvalue)

Pointers for Inter-Function Communication

- Passing addresses

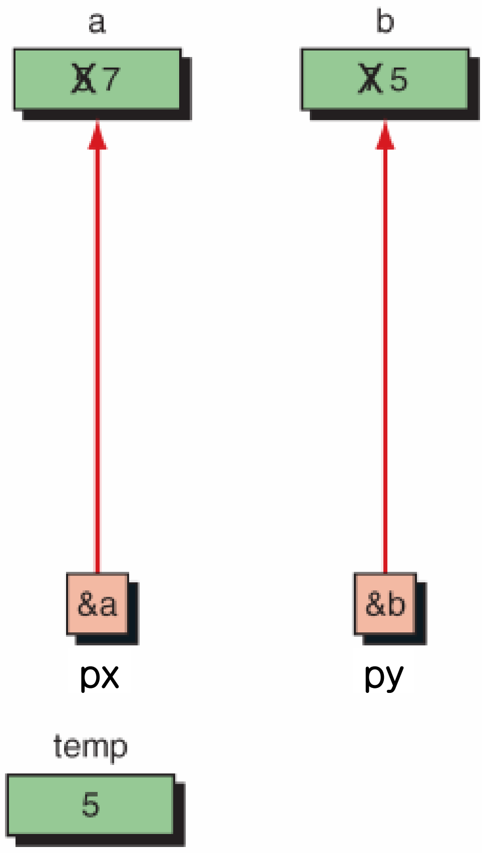
```
// Function Declaration
void exchange (int*, int*);

int main (void)
{
   int a = 5;
   int b = 7;

   exchange (&a, &b);
   printf("%d %d\n", a, b);
   return 0;
} // main
```

```
void exchange (int* px, int* py)
{
   int temp;

   temp = *px;
   *px  = *py;
   *py  = temp;
   return;
} // exchange
```



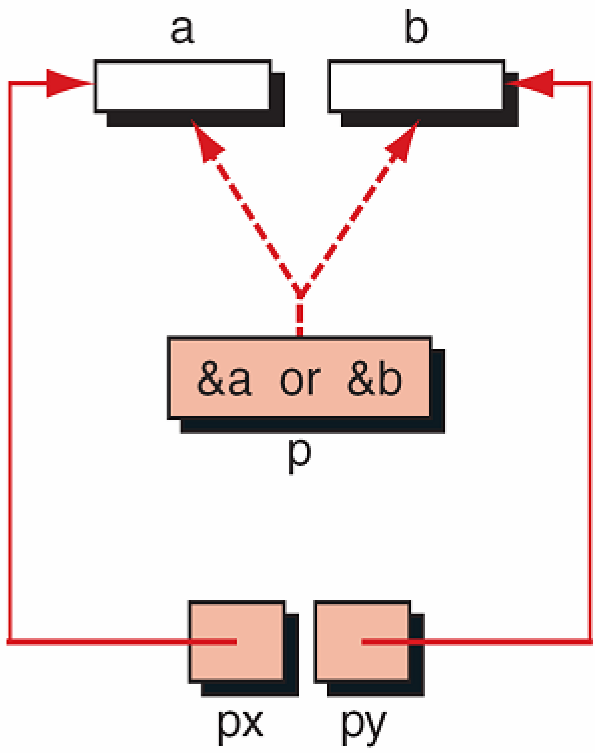
Pointers for Inter-Function Communication

- Functions returning pointers

```
// Prototype Declarations
int* smaller (int* p1, int* p2);

int main (void)
   …
   int  a;
   int  b;
   int* p;
   …
   scanf ( "%d %d", &a, &b );
   p = smaller (&a, &b);
   …
```

```
int* smaller (int* px, int* py)
{
   return (*px < *py ? px : py);
} // smaller
```

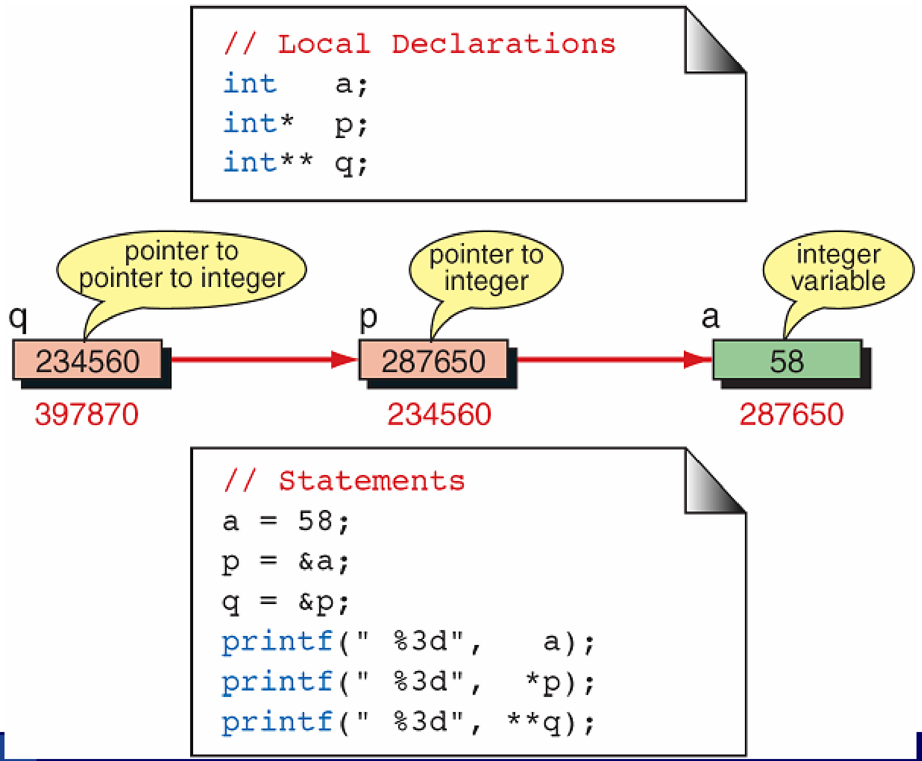


Agenda

- Introduction
- Pointer for Inter-Function Communication
- <u>Pointers to Pointers</u>
- Compatibility
- (Lvalue and Rvalue)

Pointers to Pointers

- Pointer to pointer (double pointer): a pointer that points a pointer variable
  - Note! Pointer variable itself occupies memory space



```
// Local Declarations
int    a;
int*   p;
int**  q;
```

```
// Statements
a = 58;
p = &a;
q = &p;
printf(" %3d",   a);
printf(" %3d",  *p);
printf(" %3d", **q);
```

Example: Double Pointers

- Exchange pointer variables

```
int main()
{
   int a = 10, b = 20;
   int *p1 = &a, *p2 = &b;

   ExchangePointers(&p1, &p2);
   printf("*p1 = %d, *p2 = %d₩n",
    *p1, *p2);
}
```

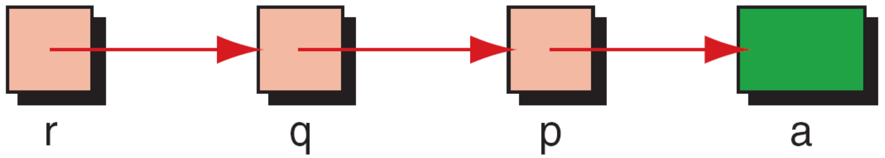
```
void ExchangePointers(
      int **pa, int **pb)
{
   int *temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```

Pointers to Pointers

- Triple pointer

```
int a = 0;
int *p = &a;           // same with int *p; p = &a;
int **q = &p;
int ***r = &q;

// Note a ≡ *p ≡ **q ≡ ***r
```



Agenda

- Introduction
- Pointer for Inter-Function Communication
- Pointers to Pointers
- <u>Compatibility</u>
- (Lvalue and Rvalue)

Compatibility

- Pointer type compatibility
  - A pointer variable can store a pointer of the same type.

```
Ex) char c, *pc;
    int a;
    pc = &c;       // no problem
    pc = &a;       // prohibited
```

- Pointer size compatibility
  - Although size of a variable vary with types, size of all pointers are the same.
    - int   i, *pi;
    - char   c, *pc;
    - float f, *pf;

    sizeof(i) ≠ sizeof(c) ≠ sizeof(f)

    sizeof(pi) = sizeof(pc) = sizeof(pf)

Pointer to Void

- void type pointer (void *) is just to store a generic address
  - A generic type that is not associated with a reference type

- void pointer can store any type of pointers

```
void *vp;
int a;
char c;
vp = &a;      // assigning integer pointer to vp
vp = &c;      // assigning character pointer to vp
```

- NULL pointer
  - NULL is defined by (void*)0, in stdio.h
  - Frequently used to initialize pointer variables

```
Ex) int a = 0;
    int *p = 0;         // type mismatched
    int *p = NULL;      // OK
```

Pointer to Void

- void pointer cannot be dereferenced as it is

```
int a = 10;
void *pVoid = &a;
*pVoid = 10;          // illegal
```

  To be dereferenced, void pointer should be casted.

- void pointer can be dereferenced by casting

```
int a = 10;
void *pVoid = &a;
printf("*(int)pVoid = %d₩n", *(int*)pVoid);
```