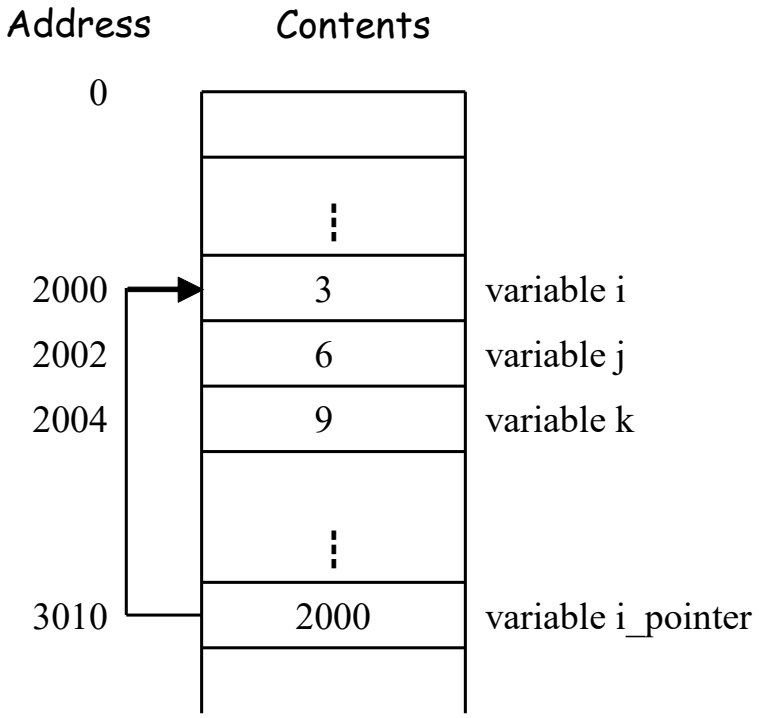
Pointer

Instructor: HOU, Fen

2025

Revisiting scanf()

- scanf("%d", &x);
  - Simply put an ampersand (&) in front of the variable and you can have the address of the variable.
  - You want to scan a value into a local variable, you need to pass the address of that variable. That is, put the ampersand (&) in front of the variable.
  - In scanf function, the address is used as an arguments to cause an appropriate value to be stored at a particular address in memory.

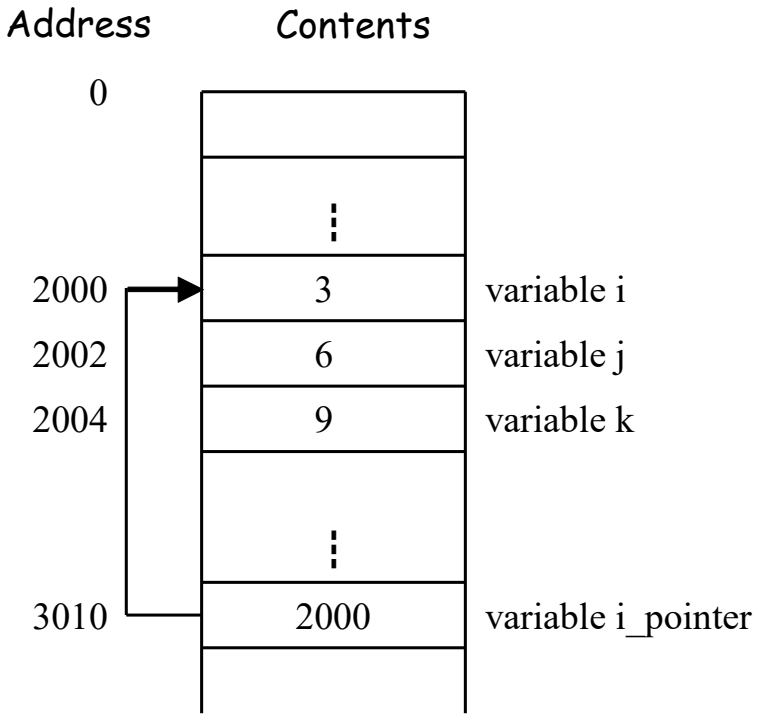


Data Types

- Variables hold values
  - Integer types
    - long int (or long)
    - int
    - short int (or short)
    - char
  - Floating point types
    - float
    - double
    - long double
- Pointer variables hold addresses

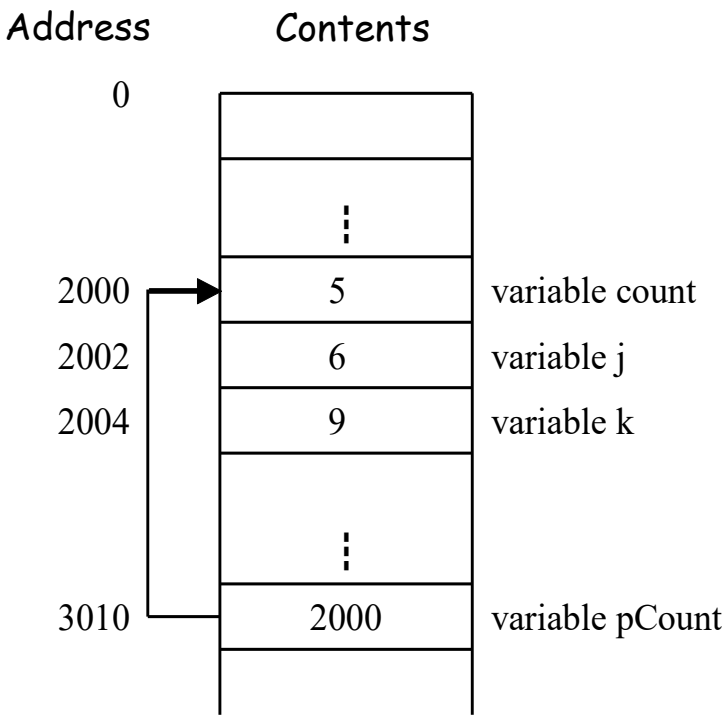
What is a pointer?

- Pointer variable, simply called as pointer
- Pointers are used to access memory and manipulate addresses.
- Pointers
  - Hold memory addresses as their values.
  - Its value much be the addresses, cannot be the data.
  - It is a kind of data type. However, it contains the memory address of a variable that in turn contains a specific value. That is, it stores the initial address of the variable which it wants to point to.
- Pointers deal with addresses—not value
  - The value in the pointer is an address
  - An operator performs an action at the value indicated by the pointer



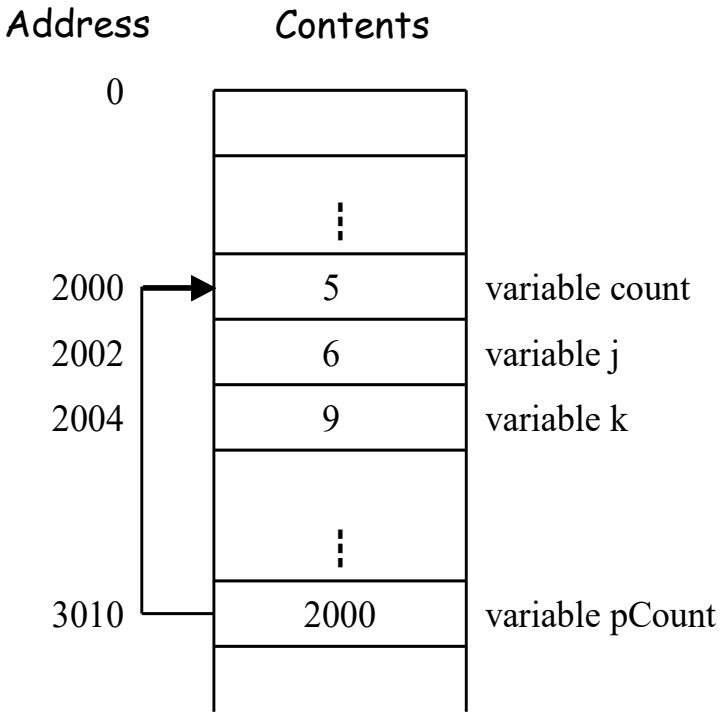
Pointer Declaration

- Declare a Pointer: Like any other variables, pointer must be declared before they can be used.
- It is declared like regular variable except that an asterisk (*) is placed in front of the variable.
- For example:
- int *x;
- Using this pointer now would be very careful since x points to some random piece of data.
- Declaring a variable does not allocate space on heap for it.
  - It simply creates a local variable (on the stack) that will be a pointer.
  - Use malloc() to actually request memory on the heap.

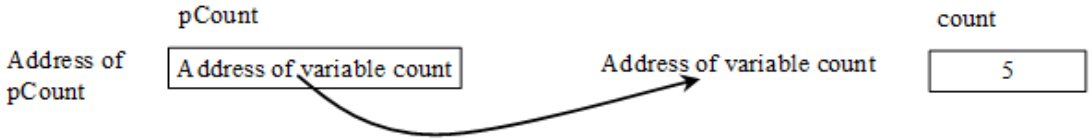


Pointer Declaration

- Basic syntax: To declare a pointer, use the following syntax:
- dataType *pVarName;
- For example, the following statement declares a pointer variable named pCount that can point to an int variable. The address operator & is unary, then &count is the address or location in memory of its stored value.

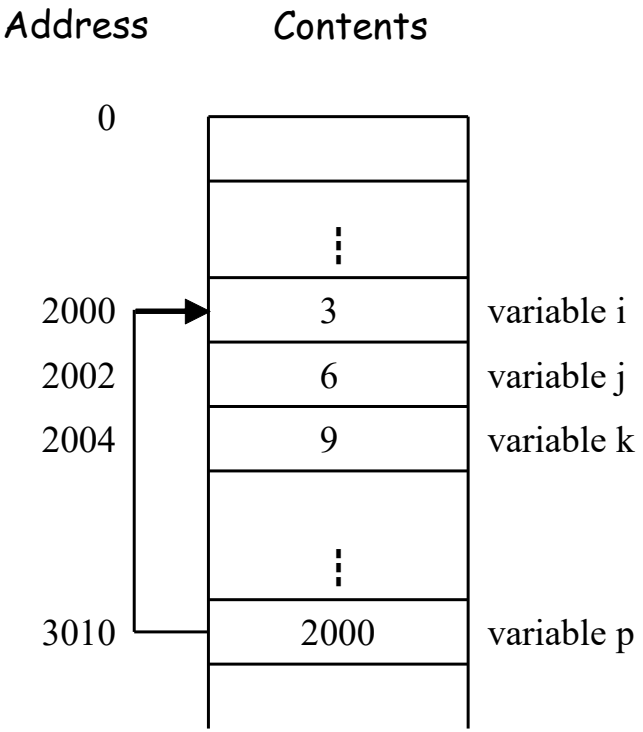



```
int count;
int *pCount;
pCount = &count;
```



Pointer Declaration

- For example,
  - int i, *p; /* declare a pointer p , p is a pointer to int variable*/
  - float *Q; /*Q is a float pointer. It points to a float variable */
  - char *R ; /* R is a char pointer. It points to a char variable.*/
- Can we declare two pointer variables on the same line as follows?
  - int *pI, pJ;
- No, it should be
  - int *pI, *pJ;
- We declare two variables on the same line as follows
  - int I, *pI ;
  - I is an int variable, and pI is int pointer. It points to an int variable.

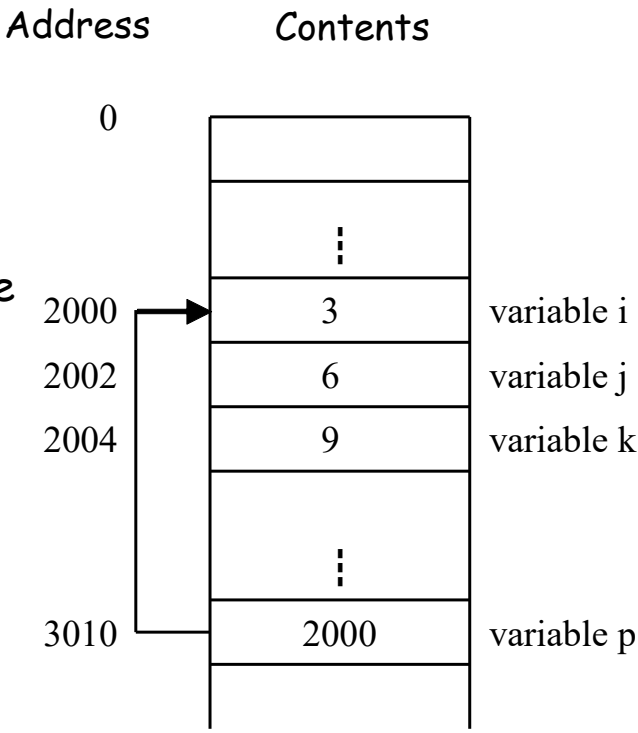


Using a Pointer: Dereferencing operator (*)

- The asterisk *
  - It is the dereferencing operator
  - Defines a pointer type in a declaration
  - Dereferences a pointer variable: dereferences a pointer variable is to get the contents at the address that a pointer points to.
  - To access a piece of data through a pointer, please put an asterisk (*) before the pointer.
- Example
  - int num, othernum; //Declare two int variables
  - int *ptrnum; //Declare a pointer variable
  - ptrum = # //Assign an address to a pointer variable
  - othernum = *ptrum; // Retrieve the value in the address that a pointer points to and save it to the int variable othernum.
- &num is the address (memory location) of integer variable num
- Use the pointer without the asterisk actually accesses the pointer value.
  - Not the data the pointer is referencing
  - This is a very common mistake to make when trying to access data.

Using a Pointer: Addressing Operator (&)

- The ampersand &
  - It is the addressing operator
  - Retrieves the address of a variable.
  - The addressing operator (ampersand &) can be used in front of any variable in C.
  - It is to retrieves the address of a variable. That is, the result of the operation is the address/memory location of the variable.
  - Example
    - int V;
    - int *P; //Declare a pointer variable P
    - scanf("%d", &V);
  - Example
    - int num, othernum; //Declare two int variables
    - int *ptrnum; //Declare a pointer variable
    - ptrum = # //Assign an address to a pointer variable
    - othernum = *ptrum; // Retrieve the value in the address that a pointer points to and save it to the int variable othernum.
  - &num is the address (memory location) of integer variable num



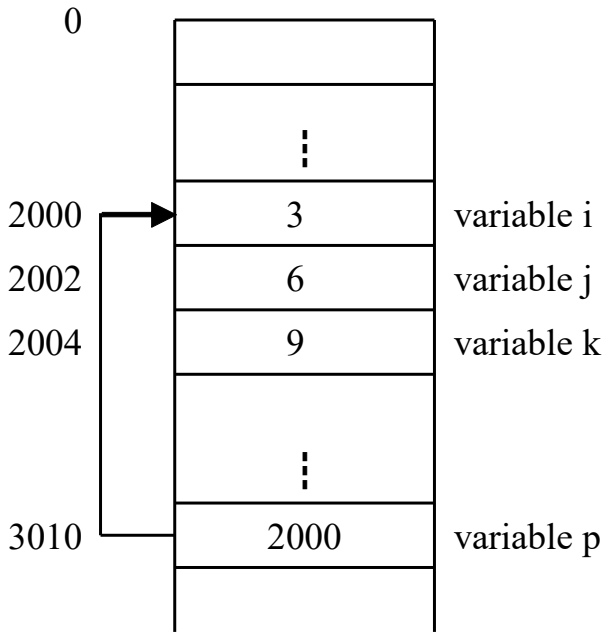
Pointer Type

- Must assign the address of the variable of the same type.
- It is a syntax error if the type of the variable does not match the type of the pointer. For example, the following code is wrong
- For example:

```
int area = 1;
double *pArea = &area;   /* Wrong */
```

Pointer Initialization

- A pointer is assigned an arbitrary value if it is not initialized.
- Several ways to do the pointer initialization
  - int i, *p; /* declare a pointer p , p is a pointer to int variable*/
  - P=&i; /* p contains the address of i, or p refers to i, or p points to i */
  - P=0;
  - P=NULL;
  - int m;
  - int *V=&m;
- zero is a special address used to initialize a pointer.
- NULL is the symbolic constant, which is defined as zero in stdio.h. It is used to indicate pointer points to nothing.



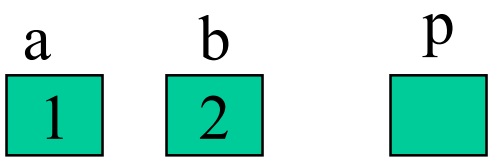
Pointer Initialization

- A pointer variable can be initialized in the declaration.
  - int a, *p = &a;
- P is a pointer pointing to an int variable, and the initial value of p is the address of a.

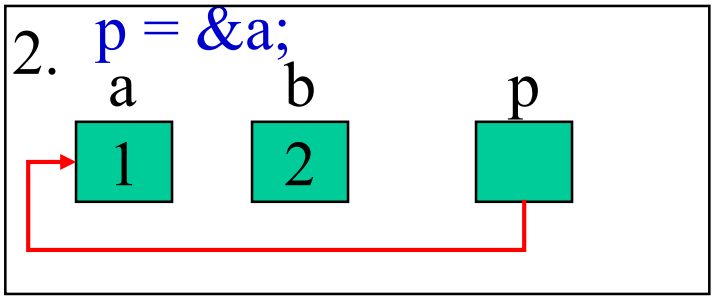
- int *p = &a, a; (Right or Wrong?)
  - Wrong, The complier must allocate space in memory a before p can be initialized with its address.
  - Change to int a, *p=&a;

Pointers

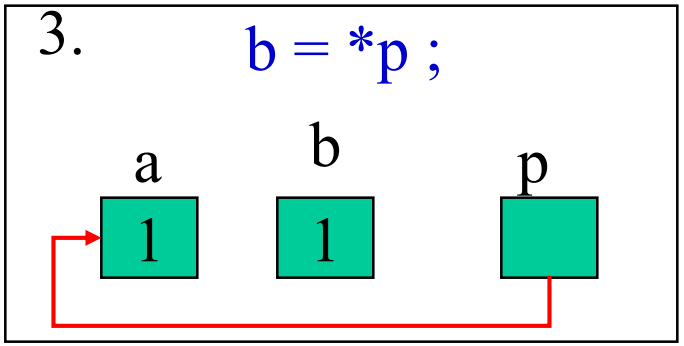
1. int a=1, b=2, *p;



2. p = &a;



3. b = *p ;



Dereferencing

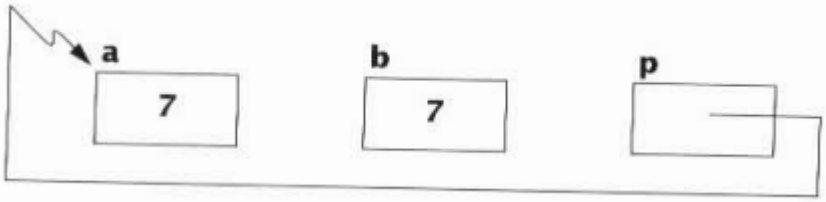
- Pointers deal with addresses – not value
  - an operator performs an action at the value indicated by the pointer
  - the value in the pointer is an address
- We can find the value of any variable by dereferencing it

Dereferencing

- Referencing a value through a pointer is called indirection. The syntax for referencing a value from a point is

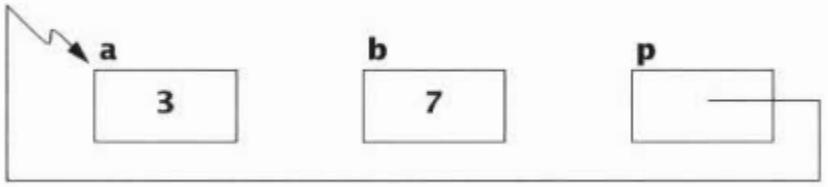
  *pointer /*use asterisk * to dereference the pointer */

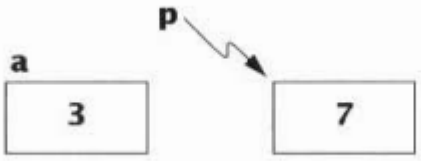
- Dereference or indirection operator * (间接访问运算符）：which is used to access the value stored in a variable that a pointer points to.
- This expression has the value of the variable to which the pointer points. And the pointer is said to store the address of the variable it points to.
- For example:
  - int a, b;
  - int *p;
  - a=7, b=7;
  - p=&a; /*p points to the variable a */
  - printf("*p=%d\n", *p); /*the expression *p has the value of the variable a */



Dereferencing

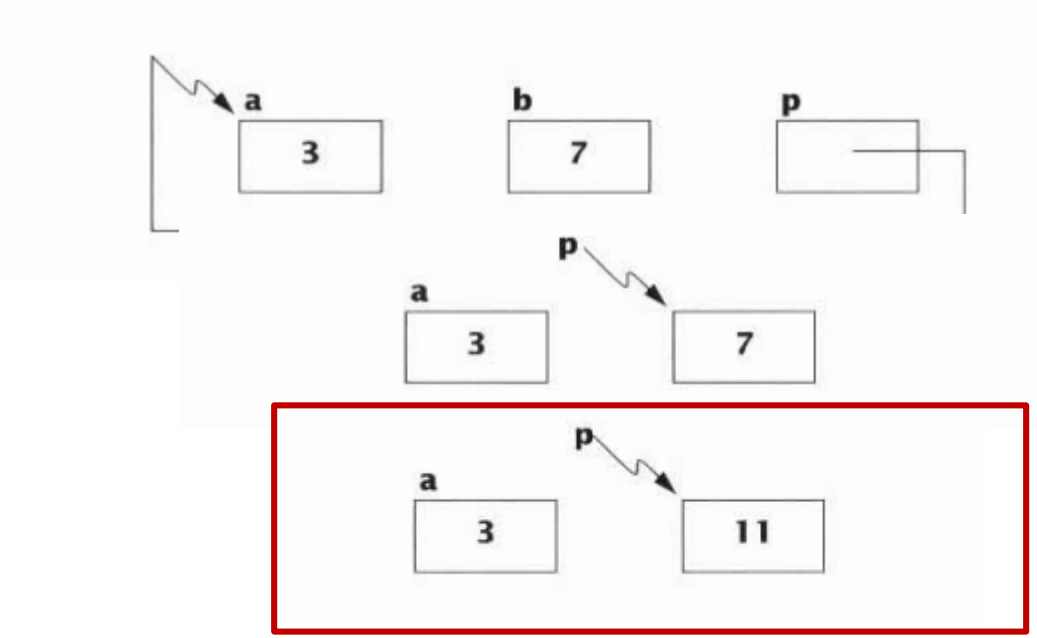
- For example:
  - int a, b;
  - int *p;
  - a=7, b=7;
  - p=&a; /* p points to the variable a*/
  - *p=3; /*the variable that the pointer p pointed to is assigned the value 3 */
  - printf("a=%d\n", a);
  - P=&b;
  - printf("*p=%d\n", *p);





Dereferencing

- For example:
  - int a, b;
  - int *p;
  - a=7, b=7;
  - p=&a; /* p points to the variable a*/
  - *p=3; /*the variable that the pointer p pointed to is assigned the value 3 */
  - printf("a=%d\n", a);
  - P=&b;
  - *p=2* *p - a;
  - printf("b=%d\n", b);



Dereferencing

- Use the scanf to input a value from the keyboard and store to the variable a:

```
int a;
scanf("%d", &a)
```

- Question: Use the pointer p to accomplish this

```
int a;
int *p;
p=&a;
printf("input an integer:   ");
scanf("%d", p);   /* put the input at the address of a */
```

Dereferencing

The dereference operator * is the inverse of the address operator &.

```
int x, y, *p;
x=5;
p=&x;   /* p is assigned the address of x  */
y=*p;   /*  y is assigned the value of the object pointed to by p */
printf("the value of y is: \n", y); //the value of y is 5
printf("the value of y is: \n", *&x); //the value of *&x is 5
```

- The dereference operator * is unary and it has the same precedence and right-to-left associativity as the other unary operators such as logic NOT, minus, etc.

Dereferencing

Operator precedence and associativity

Operator	Associativity	Precedence
() ++ (postfix) -- (postfix)	left to right	Highest
+ (unary) - (unary) ++ (prefix) -- (prefix) !	right to left	
* / %	left to right	
+ -	left to right	
< <= > >=	left to right	
== !=	left to right	
&&	left to right	
||	left to right	
?:	right to left	
= += -= *= /= etc.	right to left	
, (comma operator)	left to right	Lowest

Dereferencing

- Exercise: The following table illustrates how some pointer expression are evaluated. Fill in the following table.

Declarations and initializations		
int i=3,j=5, *p=&i, *q=&j, *r; double x;		
Expression	Equivalent expression	Value
p== &i		
p=i+7		
r=&x		

Dereferencing

- The following table illustrates how some pointer expression are evaluated. Fill in the following table.

Declarations and initializations		
int i=3,j=5, *p=&i, *q=&j, *r double x;		
Expression	Equivalent expression	Value
P== &i	p==(&i)	1
P=i+7	P=(i+7)	Illegal
r=&x	r=&x	Illegal

Dereferencing

- Exercise: The following table illustrates how some pointer expression are evaluated. Fill in the following table.

Declarations and initializations		
int i=3,j=5, *p=&i, *q=&j, *r; double x;		
Expression	Equivalent expression	Value
* * &p		
7 * * p / * q +7		

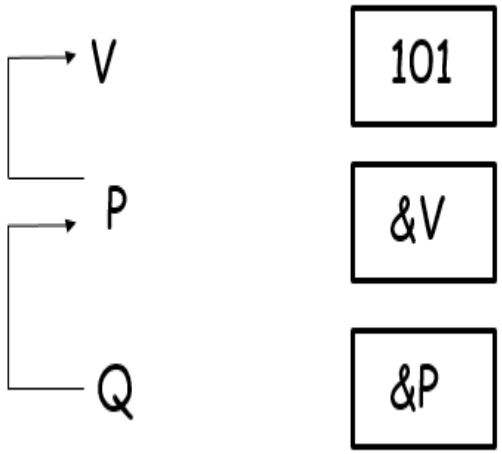
Dereferencing

- Exercise: The following table illustrates how some pointer expression are evaluated. Fill in the following table.

Declarations and initializations		
int i=3,j=5, *p=&i, *q=&j, *r; double x;		
Expression	Equivalent expression	Value
* * &p	* (* (&p))	3
7 * * p / * q +7	((7 * (* p)) / (* q)) +7	11

Pointers to pointers

- A pointer can also be made to point to a pointer variable.
- Example:
  - int V=101;
  - int *P=&V; //P points to int variable V
  - int **Q= &P; //Q points to integer pointer
  - printf("%d %d %d \n", V, *P, **Q);



- Will print 101 3 times. That is, 101 101 101

Dereferencing

- Read the following code

```
int i=777, *p=&i;
printf("the value of i is %d: \n", *p);
printf("the value of i is %u: \n", p);
```

- %u format is to print the address of i as an unsigned decimal integer;

```
Value of i: 777
Address of i: 234880252
```

Data Types

- Variables hold values
  - Integer types
    - long int (or long)
    - int
    - short int (or short)
    - char
  - Floating point types
    - float
    - double
    - long double
- Pointer variables hold addresses
  - Pointers have a base type of any legal C type including another pointer. For example,
    - float * (ptr-to-float)
    - int * (ptr-to-int)
    - char * (ptr-to-char)
    - int ** (ptr-to-ptr-to-int)
  - Example:

```
int V=60;
int *P=&V;  //P points to int V//
int **Q=&P; //Q points to int pointer P//
```

Pointer Types and Casting Pointers

- Pointers are generally of the same size (enough bytes to represent all possible memory addresses)
- It is inappropriate to assign an address of one type of variable to a different type of pointer
  - Example:

    ```
    int V = 101;
    float *P = &V;
    ```
- When assigning a memory address of a variable of one type to a pointer that points to another type it is best to use the cast operator to indicate the cast is intentional.
  - Example:

    ```
    int V = 101;
    float *P = (float *) &V;
    ```

The General (void) pointer

- A void * is considered to be a general pointer.
- No cast is needed to assign an address to a void *
  - Example:

```
int V = 101;
void *G = &V; /* No warning */
```

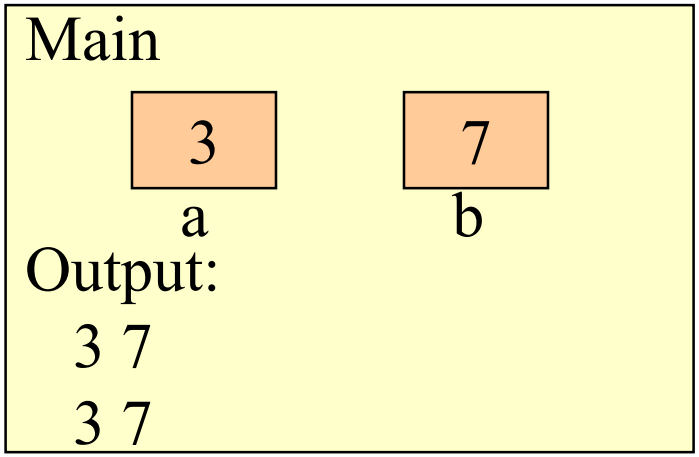
- No cast is needed to assign from a void * to another pointer type
  - Example:

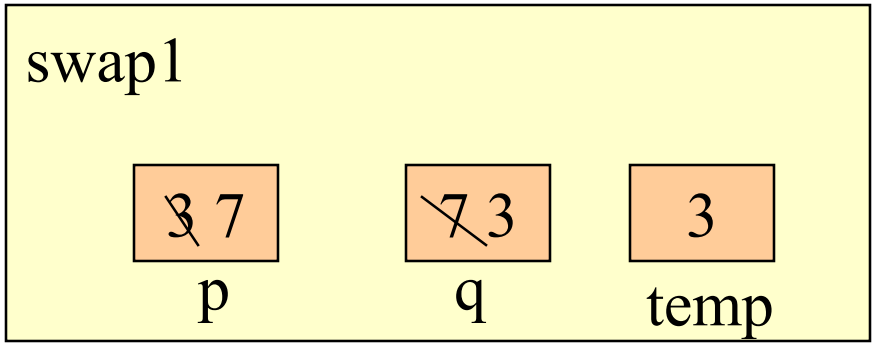
```
int V = 101;
void *G = &V; /* No warning */
float *P=G   /* No warning*/
```

Call-by-Value

- Call-by-Value: Whenever variables are passed as arguments to a function, their values are copied to the corresponding function parameters, and the variables themselves are not changed in the calling environment.
- For example

```
 include <stdio.h>
void swap (int p, int q);
int main (void )
{
     int a = 3, b = 7;
    printf ("%d,%d\n",a,b);
     swap (a, b) ;
     printf ("%d,%d\n",a,b);
     return 0;
}
void swap (int p, int q)
{
      int temp;
      temp = p;
      p= q;
      q = temp;
}
```



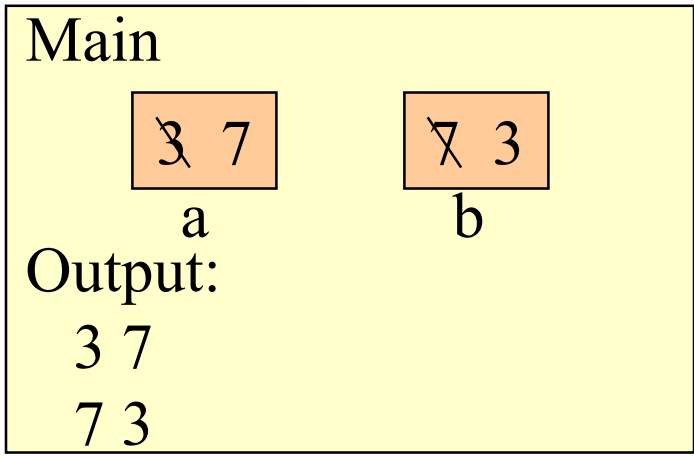


Call-by-Reference

- The addresses of variables can be used as arguments to functions.
- Read the following code: The function is to interchange the values of a and b

```
#include <stdio.h>

void   swap(int *, int *);

int main(void)
{
   int   a = 3, b = 7;

   printf("%d  %d\n", a, b);      /* 3  7 is printed */
   swap(&a, &b);
   printf("%d  %d\n", a, b);      /* 7  3 is printed */
   return 0;
}
```

```
void swap(int *p, int *q)
{
   int   tmp;

   tmp = *p;
   *p = *q;
   *q = tmp;
}
```

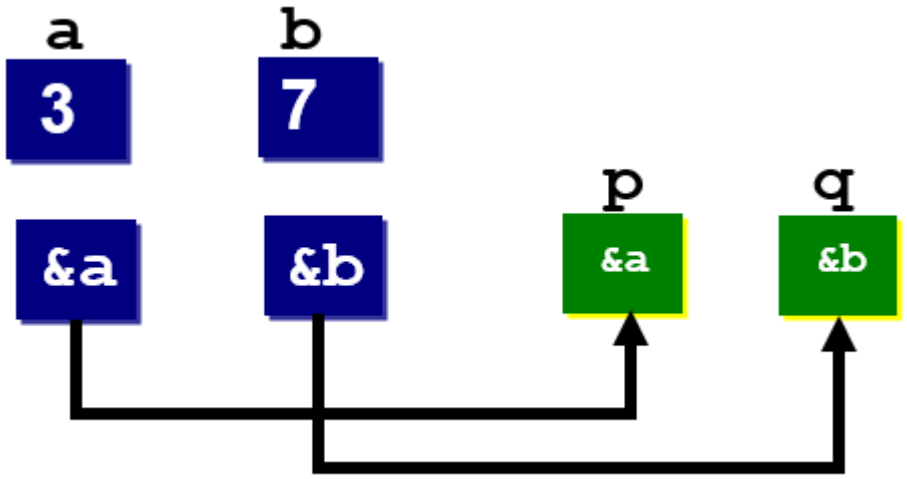


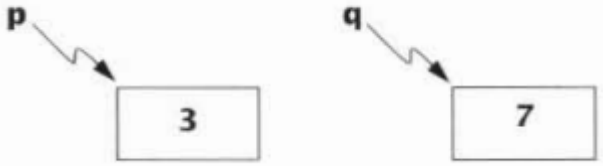
- The addresses of a and b are passed as arguments to the function swap().

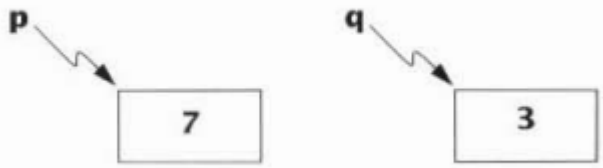
- The addresses of a and b are passed as arguments to the function swap().

  swap (&a, &b);

```
void swap(int *p, int *q)
{
   int   tmp;

   tmp = *p;
   *p = *q;
   *q = tmp;
}
```







Exercise

- What is printed by the following code?

```
int i=5, *p=&i;
printf("%d  %d  %d \n", *p +2, **&p, 3**p);
*p=1;
printf("%d\n",  i);
```

Output:

7, 5, 15

1

Exercise

- The following expression is correct or error?

  int c, *pc;

  A. pc=c;

  B. *pc - &c;

  C. pc = &c;

  C. *pc=c;

- Answer

  A. Error. pc is address but c is not;

  B. Error. &c is address, but *pc is not

  C. Correct. Both &c and pc are addresses

  D. Correct. Both c and *pc are values.