FUNCTIONS IN C

POINTERS

Objectives

To use pointers in programming

To master the use of pointers

C-114 ICS-CAS, UPLB

POINTER

C-114 ICS-CAS, UPLB

POINTER

C-114 ICS-CAS, UPLB

ADDRESS

C-114, ICS-CAS, UPLB

THE PLACE AT THE ADDRESS

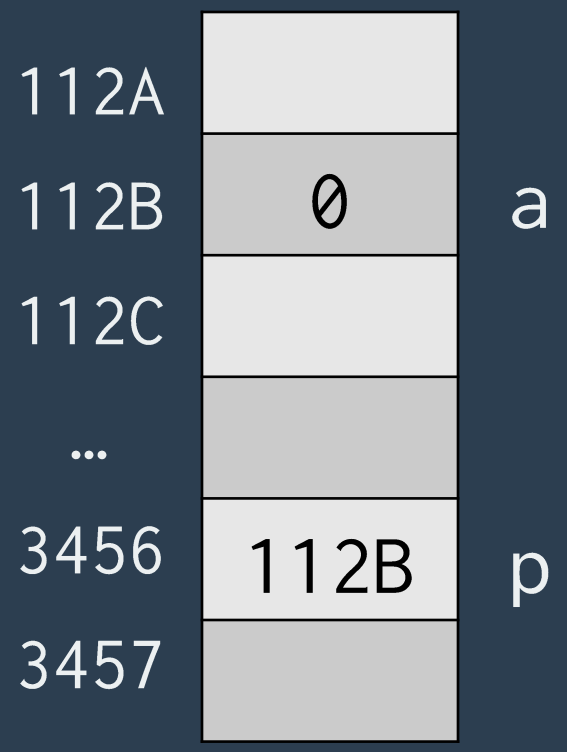
C-114, ICS-CAS, UPLB

AKA "THE DEREFERENCED VALUE"

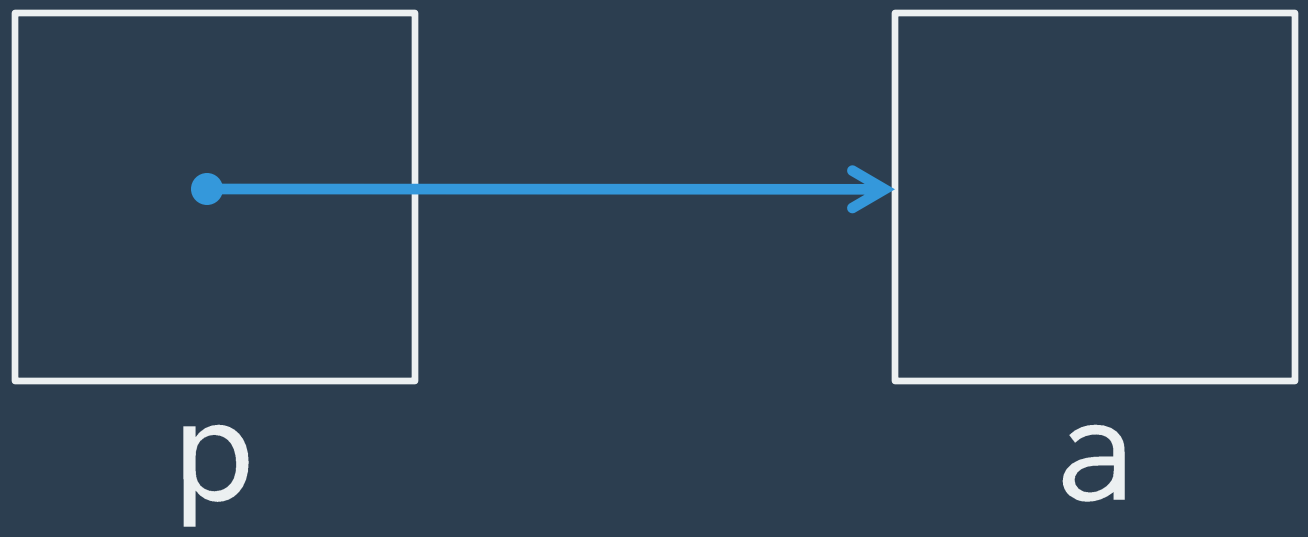
C-114, ICS-CAS, UPLB

C-112, ICS-CAS, UPLB

A pointer is a variable that holds the address of a variable stored in memory.



p holds the address where a is located in the memory


p
a

```
/*A pointer variable is
declared as:*/

<data_type> * <variable_name>;
```

<data_type>

Defines the type of variables that a pointer can point to.

```
//p is a pointer to an int
int *p;

//q is a pointer to a float
float *q;
```

<data_type>

Pointer arithmetic is done relative to <data_type>.

Pointer operators

* Indirection operator

*aka “the value-at-address operator”

* Returns the value of the variable located at the specified address

* Returns the value of the variable located at the specified address

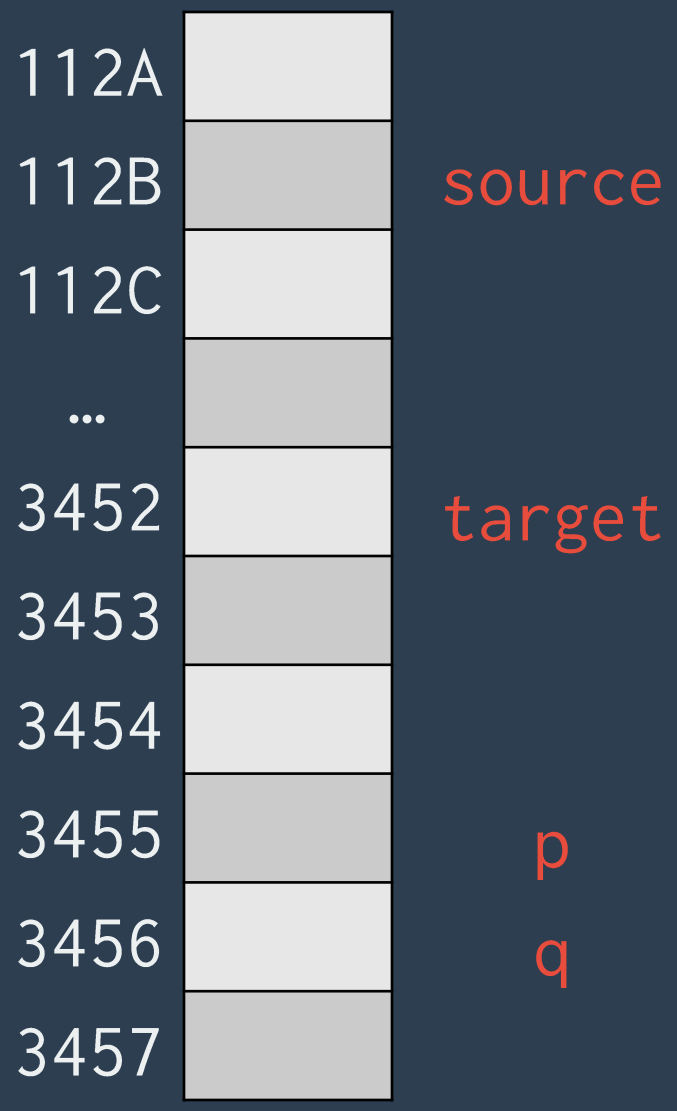
"dereferencing"

* Used both when declaring a pointer and when dereferencing a pointer.

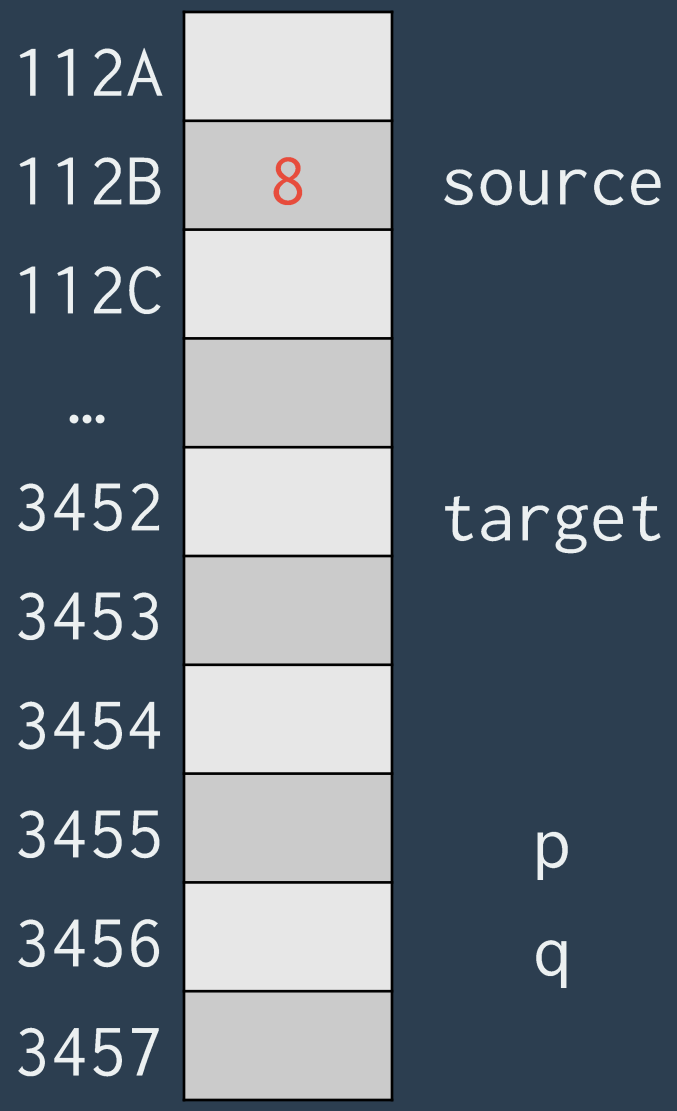
& Address operator

*aka “the memory address of operator”

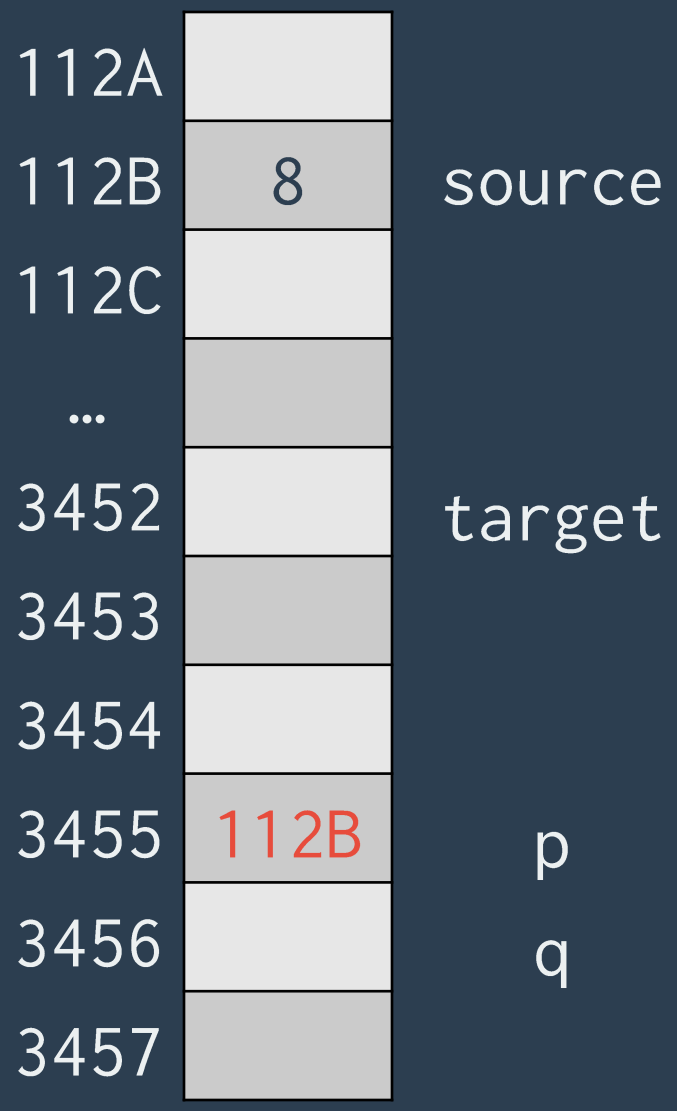
```
int main()
{
   int source;
   int target;
   int *p, *q;

   source = 8;
   p = &source;
   target = *p;
   q = p;
}
```



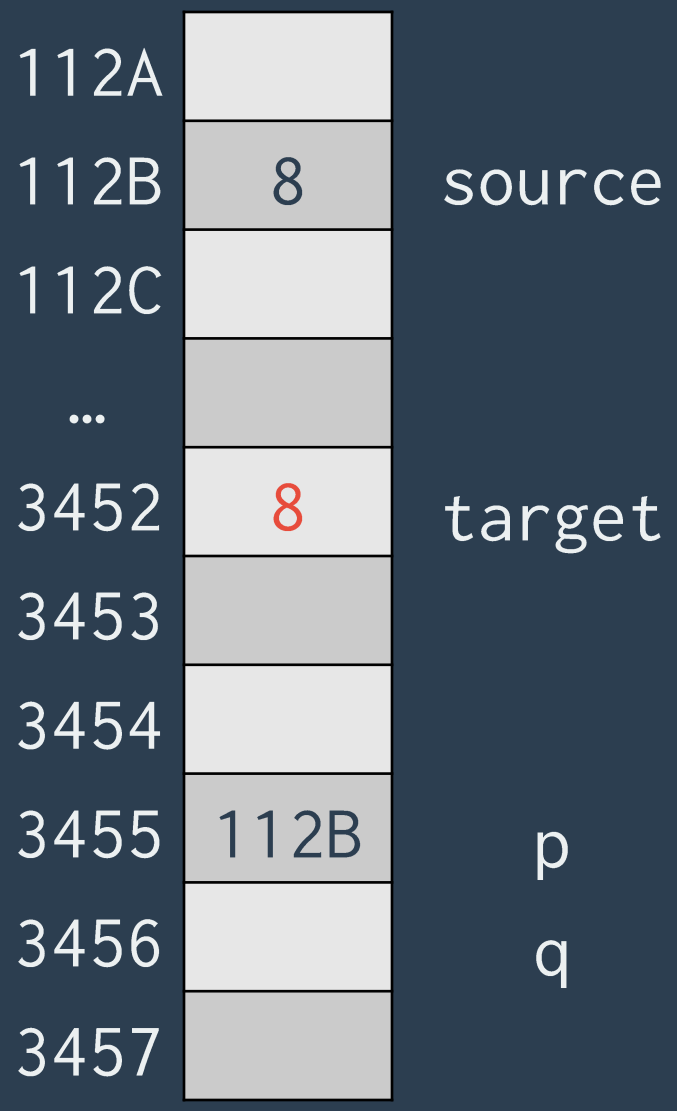
```
int main()
{
   int source;
   int target;
   int *p, *q;

   source = 8;
   p = &source;
   target = *p;
   q = p;
}
```



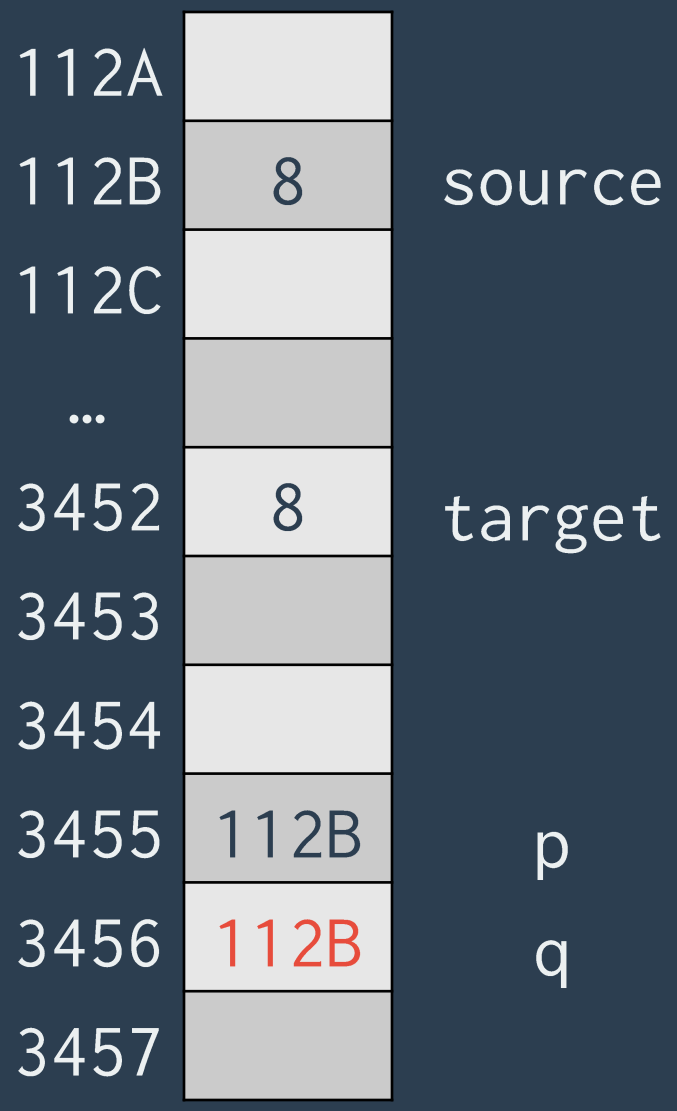
```
int main()
{
   int source;
   int target;
   int *p, *q;

   source = 8;
   p = &source;
   target = *p;
   q = p;
}
```

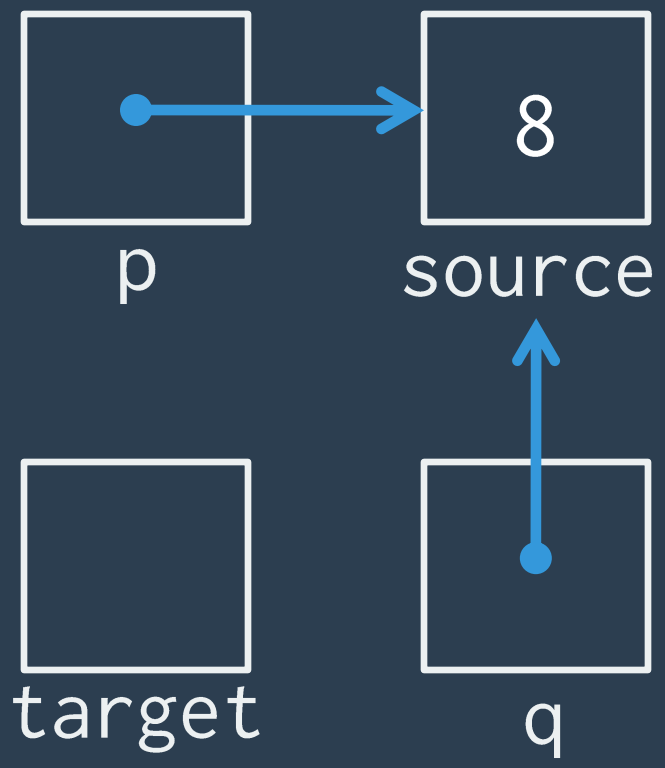


```
int main()
{
   int source;
   int target;
   int *p, *q;

   source = 8;
   p = &source;
   target = *p;
   q = p;
}
```



```
int main()
{
   int source;
   int target;
   int *p, *q;

   source = 8;
   p = &source;
   target = *p;
   q = p;
}
```





112A		
112B	8	source
112C		
…		
3452	8	target
3453		
3454		
3455	112B	p
3456	112B	q
3457		

Advantages in using pointers

They modify actual parameters.

Efficient accessing of array elements.

Improves the efficiency of certain routines.

Used to support
dynamic
data structures.

Some notes

Remember, * has two different uses.

```
//DECLARATION
int main()
{
  int num;
  int *p = &num;
  //is the same as:
}
```

```
//DECLARATION
int main()
{
  int num;
  int *p;
  p = &num;
}
```

```
//DEREFERENCING
int main()
{
   int num;
   int *p;
   p = &num;
   *p = 7;
}
```

Make sure that a pointer refers to a memory location *before* using it.

```
//INCORRECT
int main()
{
   int x, *p;

   x = 8;
   *p = x;
}
```

```
//INCORRECT
int main()
{
   int x, *p;

   x = 8;
   *p = x; //invalid
}
```

There should be a correspondence to the <data_type> when assigning values to pointers.

```
//INCORRECT
int main()
{
  int x = 8, *p;

  p = x;
  printf("%d", *p);
}
```

```
//INCORRECT
int main()
{
   int x = 8, *p;

   p = x; //invalid
   printf("%d", *p);
}
```

```
//INCORRECT
int main()
{
   int x = 8, *p;
   float *q;
   p = &x;
   q = p;
}
```

```
//INCORRECT
int main()
{
   int x = 8, *p;
   float *q;
   p = &x;
   q = p; //invalid
}
```

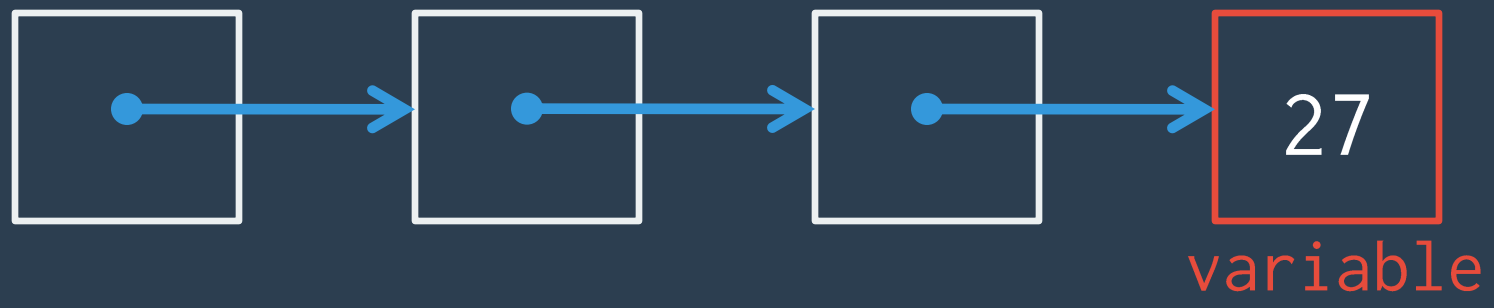
An address can be used whenever a pointer is used.

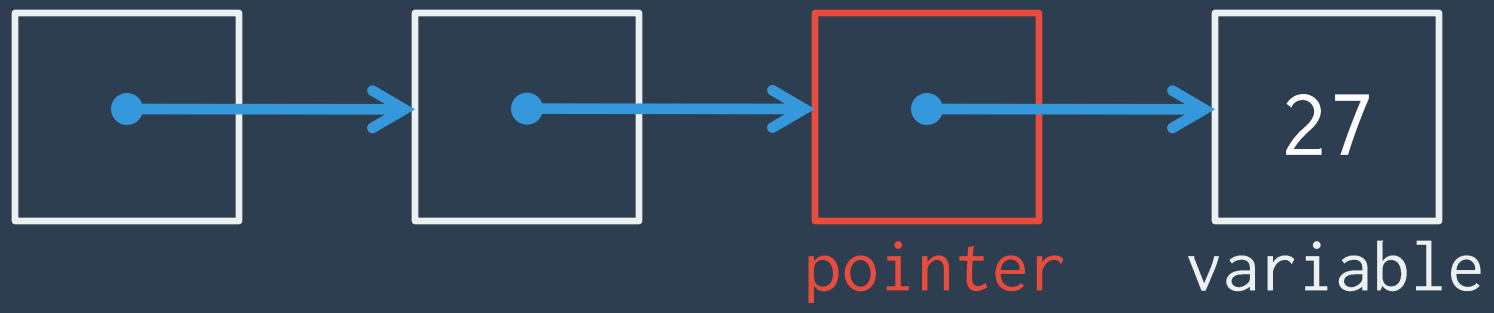
```
int main()
{
  int *ptr;
  int val = 1;
  p = &val;

  printf("DEREFERENCES");
  printf("*ptr = %d\n", *ptr);
  printf("*(&val) = %d\n", *(&val));
}
```

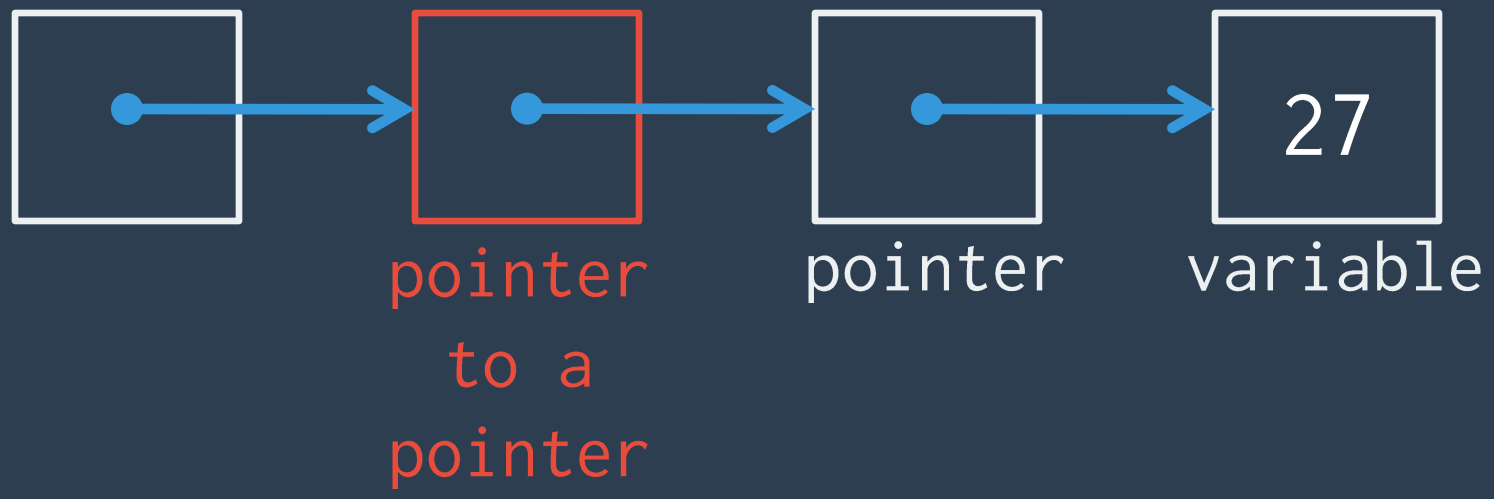
Pointers to pointers

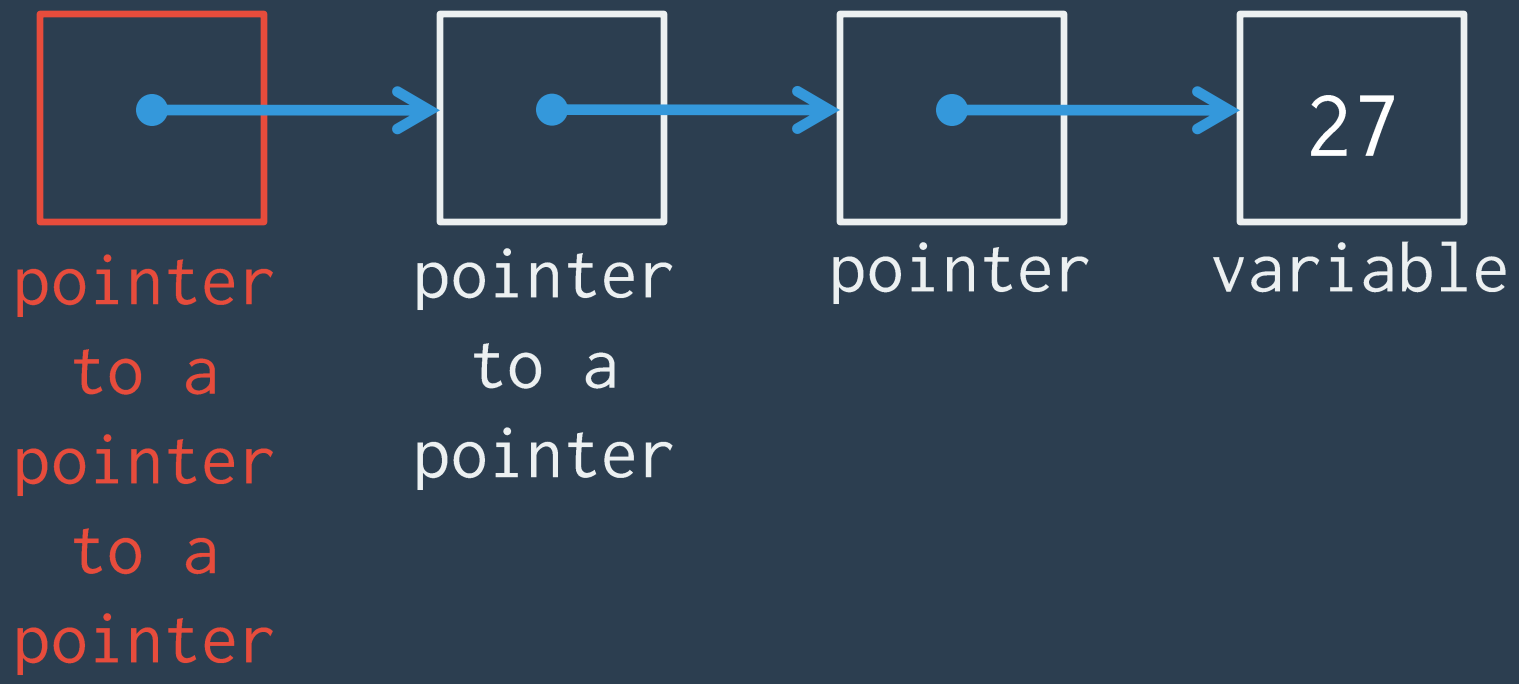
“A chain of pointers.”

Multiple indirection


27
variable


27
pointer
variable


27
pointer
to a
pointer
pointer
variable


27
pointer to a pointer to a pointer
pointer to a pointer
pointer
variable

```
/*To declare pointer to
pointers:*/

<data_type> ** <var_name>;
```

Multiple *s

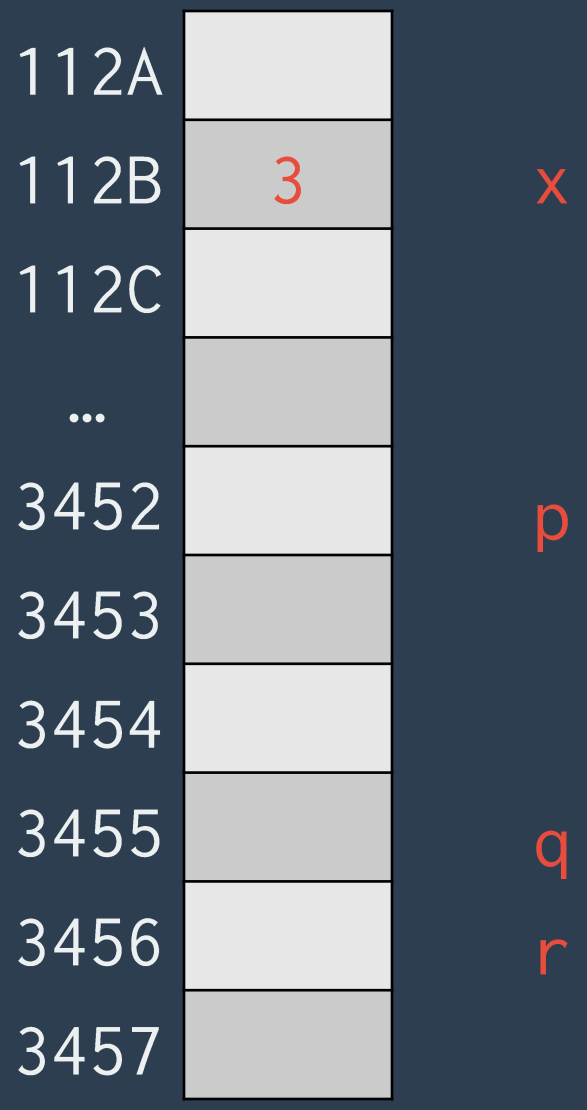
```
//p is a pointer to an int
int *p;
```

```
// p is a pointer to an int
int *p;

// q is a pointer to a pointer
// to an int
int **q;
```

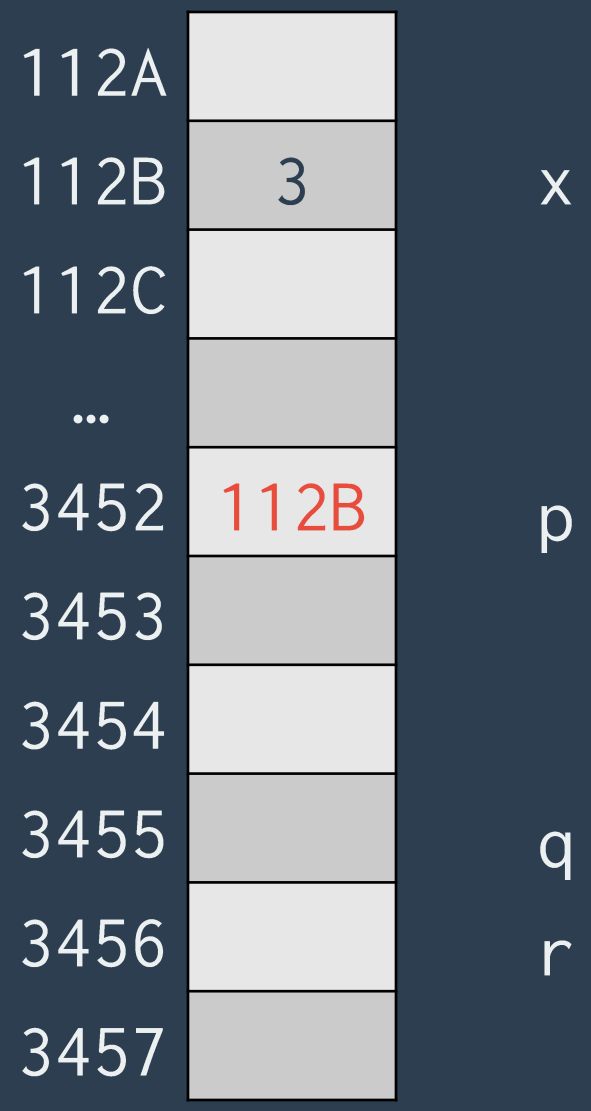
```
// r is a pointer to a pointer to
a pointer to an int
int ***r;
```

```
main ()
{
  int x = 3;
  int *p, **q, ***r;
  p = &x;
  q = &p;
  r = &q;
}
```

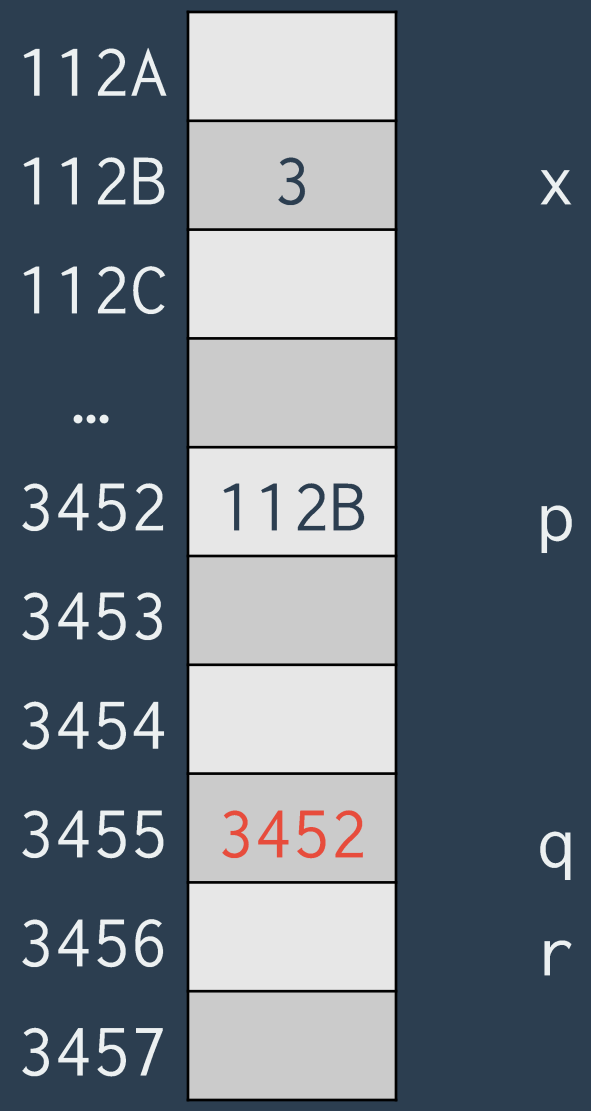
```
main ()
{
   int x = 3;
   int *p, **q, ***r;
   p = &x;
   q = &p;
   r = &q;
}
```



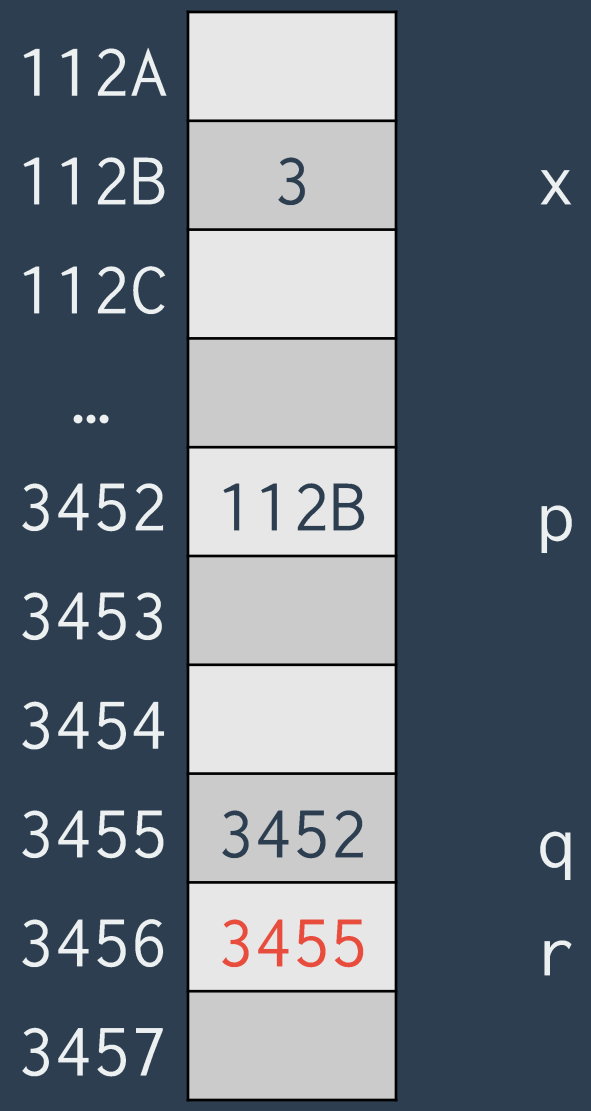
```
main ()
{
   int x = 3;
   int *p, **q, ***r;
   p = &x;
   q = &p;
   r = &q;
}
```

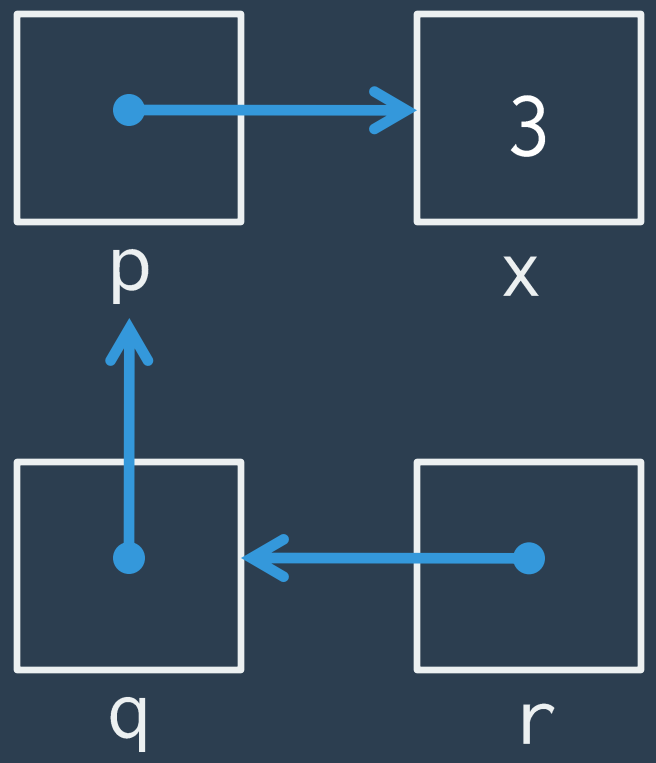


```
main ()
{
   int x = 3;
   int *p, **q, ***r;
   p = &x;
   q = &p;
   r = &q;
}
```

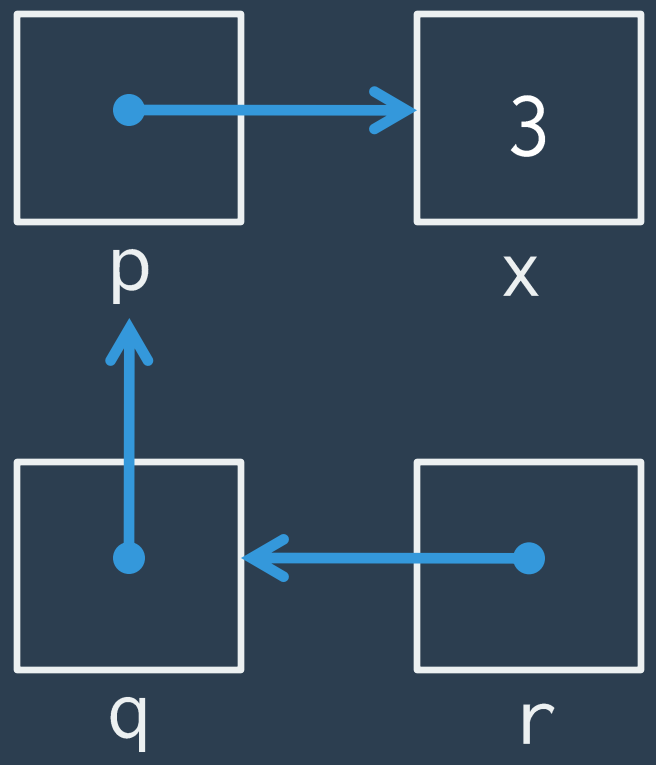


```
main ()
{
   int x = 3;
   int *p, **q, ***r;
   p = &x;
   q = &p;
   r = &q;
}
```





112A		
112B	3	x
112C		
...		
3452	112B	p
3453		
3454		
3455	3452	q
3456	3455	r
3457		



x can be accessed indirectly using p, q, and r

Pointers as parameters

Pointers are used in pass-by-reference parameter passing.

If the address of a variable is passed, the formal parameter should be a pointer.

```
int main ()
{
  int x = 10;
  foo(&x);
}

void foo(int *p)
{
  printf("%d", *p);
}
```

```
int main ()
{
  int x = 10;
  foo(&x);
}

void foo(int *p)
{
  printf("%d", *p);
}
```

"int *p = &x;"

```
int main ()
{
  int x = 10;
  foo(&x, 27);
}

void foo(int *p, int y)
{
  printf("%d", *p);
}
```

"int *p = &x;
int y = 27;"

If the address of a pointer is passed...

```
void foo(int *p)
{
  printf("%d", *p);
  bar(&p);
}

void bar(int **q)
{
  printf("%d", **q);
}
```

PROBLEM 2.

```
void one(int **p) {
  scanf ("%d",__1__); //store in x
  two(__2__); //print value of x
}
void two(int *q) {
  printf("%d",__3__);
}
int main() {
  int x, *a;
  a = &x;
  one(__4__);
}
```

Page intentionally left blank

PROBLEM 2.

```
void one(int **p) {
  scanf ("%d",  *p  ); //store in x
  two(  *p  ); //print value of x
}
void two(int *q) {
  printf("%d",  *q  );
}
int main() {
  int x, *a;
  a = &x;
  one(  &a  );
}
```