IS12 - Introduction to Programming
Lecture 21: Pointers

Peter Brusilovsky
http://www2.sis.pitt.edu/~peterb/0012-051/

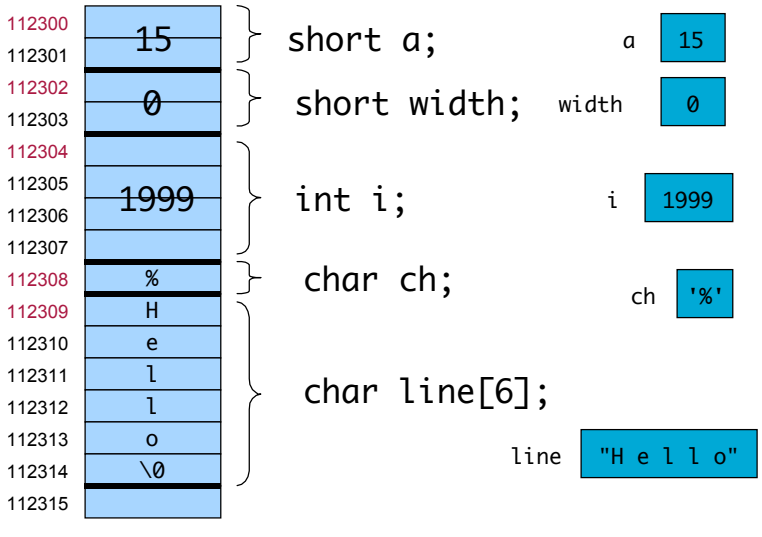
Outline

- Computer memory
- Pointers
- Pointers and function parameters
- Pointers and arrays
- Address Arithmetic

Computer memory

- Computer memory is a huge array of consecutively numbered *memory cells*
- On most modern computers a cell can store one *byte* of data
- An address of a memory cell is its number in an array
- Data of different type occupies one or more continuous cells (bytes)

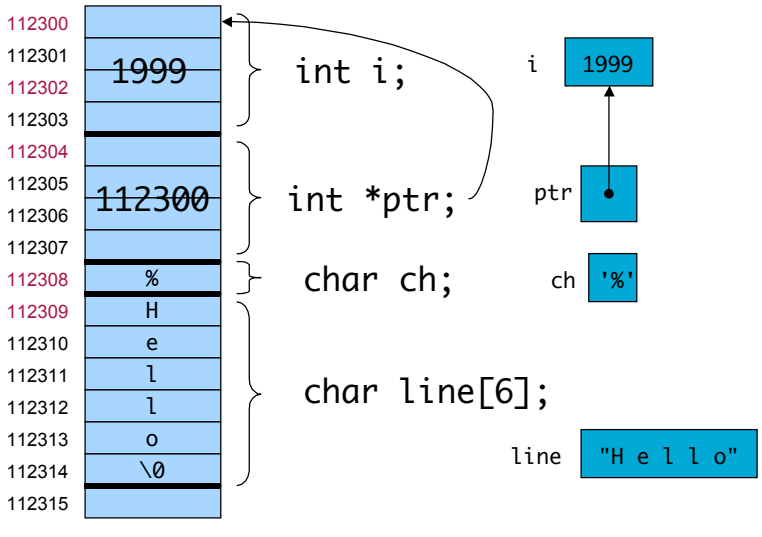
Computer memory



Pointers

- A pointer is a variable that can store an address of another variable or data object located in memory (i.e., 112304)
- We say that a pointer *points to* a variable that is stored at that address
- A pointer itself usually occupies 4 bytes of memory (then it can address cells from 0 to 2^{32}-1)

Pointers



Declaring pointers

- In C you can specify the type of variable a pointer can "points to"

```
int *ad; /* pointer to int */
char *s; /* pointer to char */
float *fp; /* pointer to float */
char **s; /* pointer to variable
  that is a pointer to char */
```

Operations with pointers

- We can assign an address to a pointer

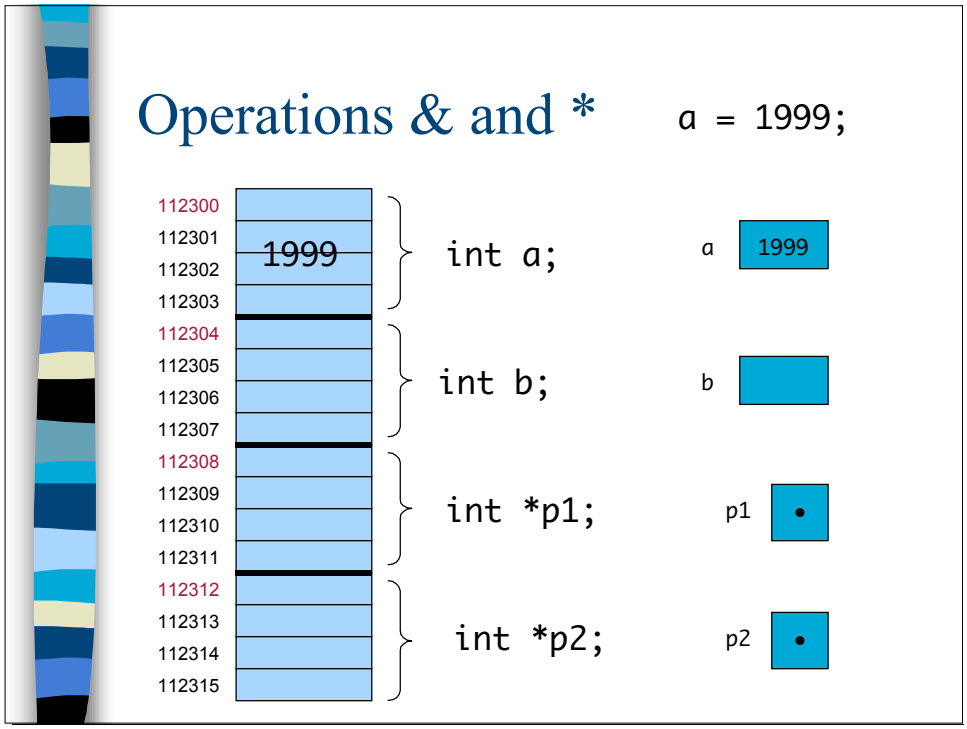
```
int *p1, *p2; int a, b;
p1 = &a; /* &a - an address of a */
```

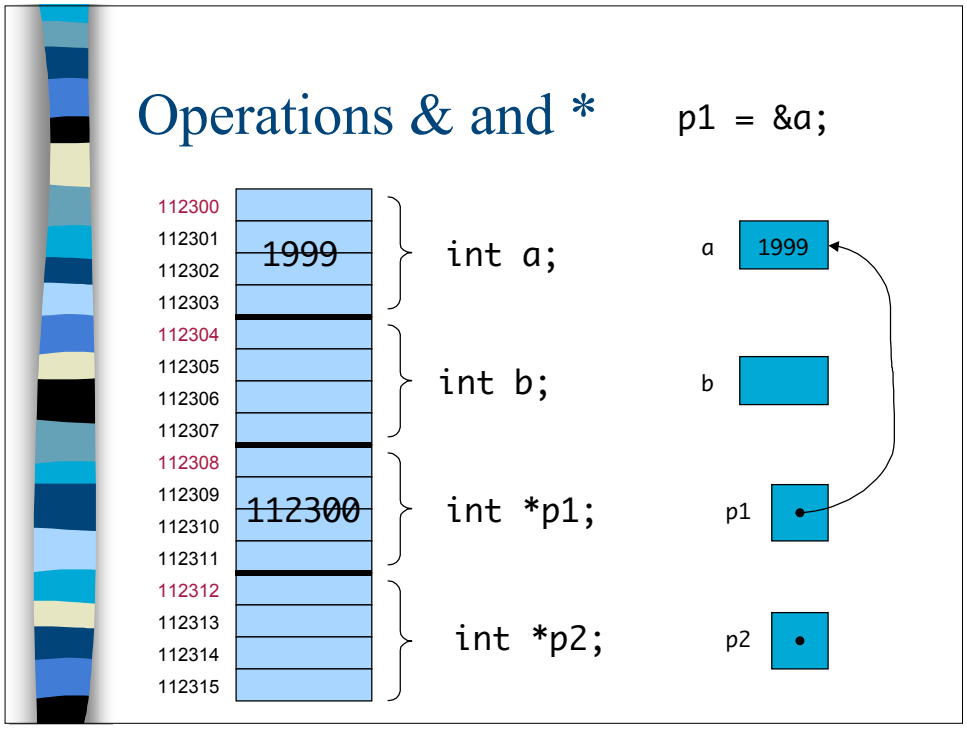
- We can assign pointers to each other

```
p2 = p1;
```

- We can *dereference* pointers

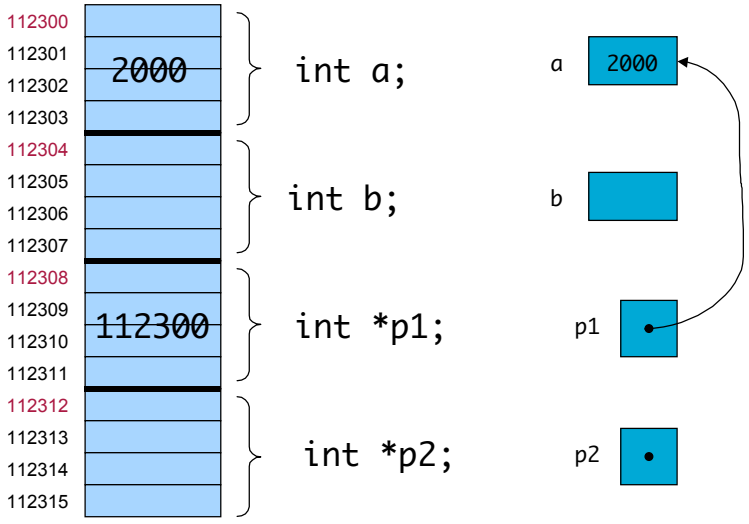
```
*p1 = 3; /* same as a = 3; */
b = *p1; /* same as b = a; */
```


Operations & and *
a = 1999;
112300
112301
112302
112303
112304
112305
112306
112307
112308
112309
112310
112311
112312
112313
112314
112315
1999
int a;
int b;
int *p1;
int *p2;
a
1999
b
p1
p2


Operations & and *
p1 = &a;
112300
112301
112302
112303
112304
112305
112306
112307
112308
112309
112310
112311
112312
112313
112314
112315
1999
112300
int a;
int b;
int *p1;
int *p2;
a
1999
b
p1
p2

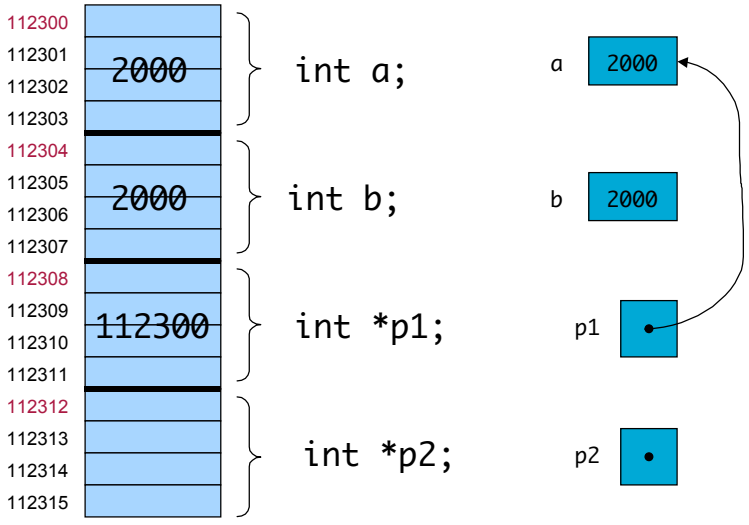
Operations & and *

*p1 = 2000;



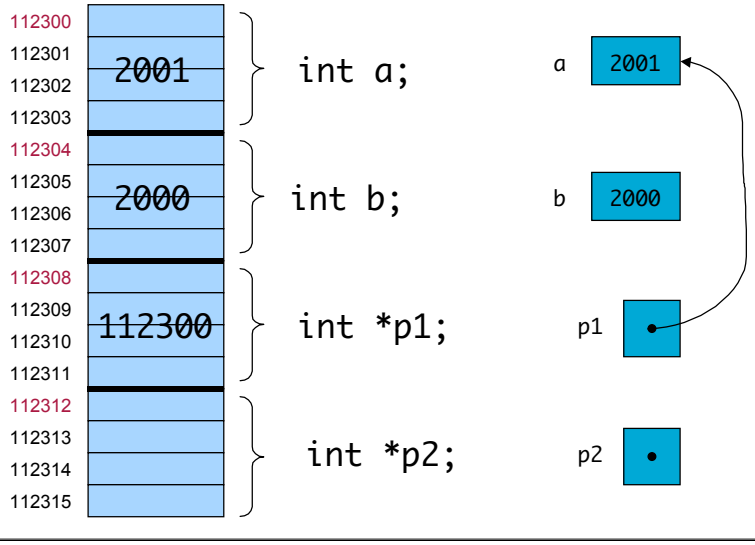
Operations & and *

b = *p1 ;



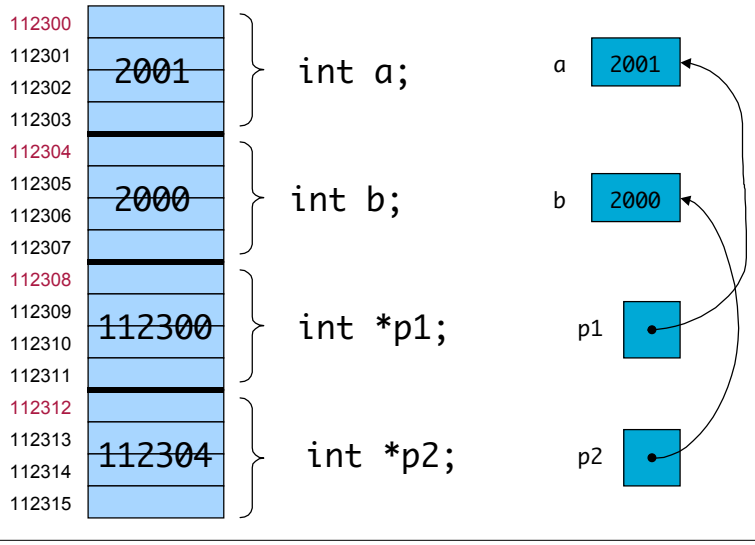
Operations & and *

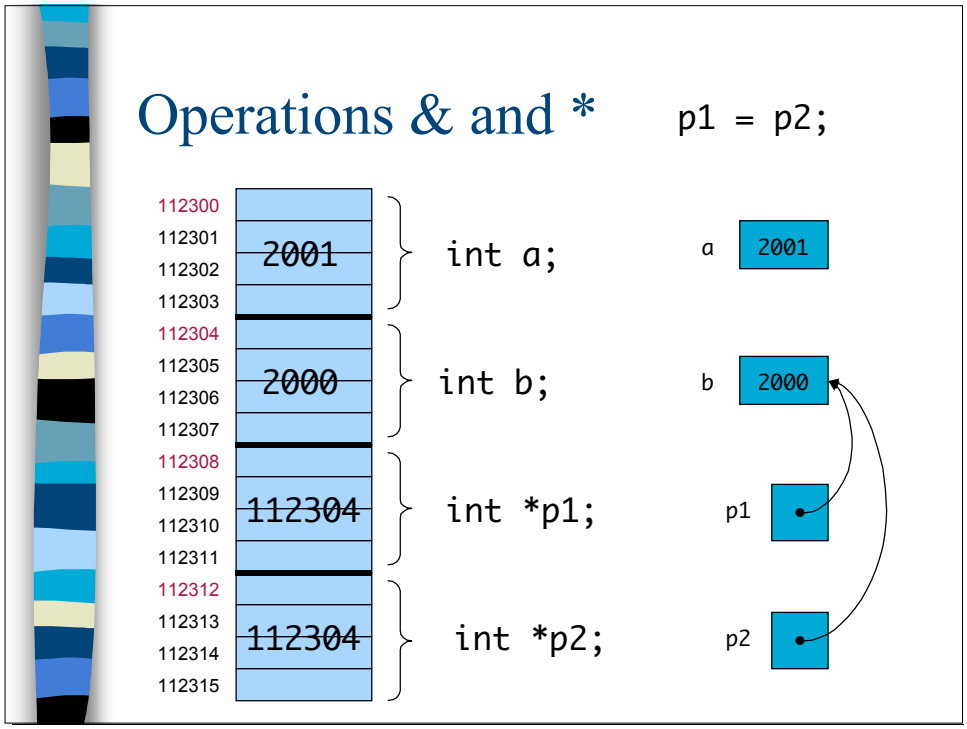
(*p1)++ ;

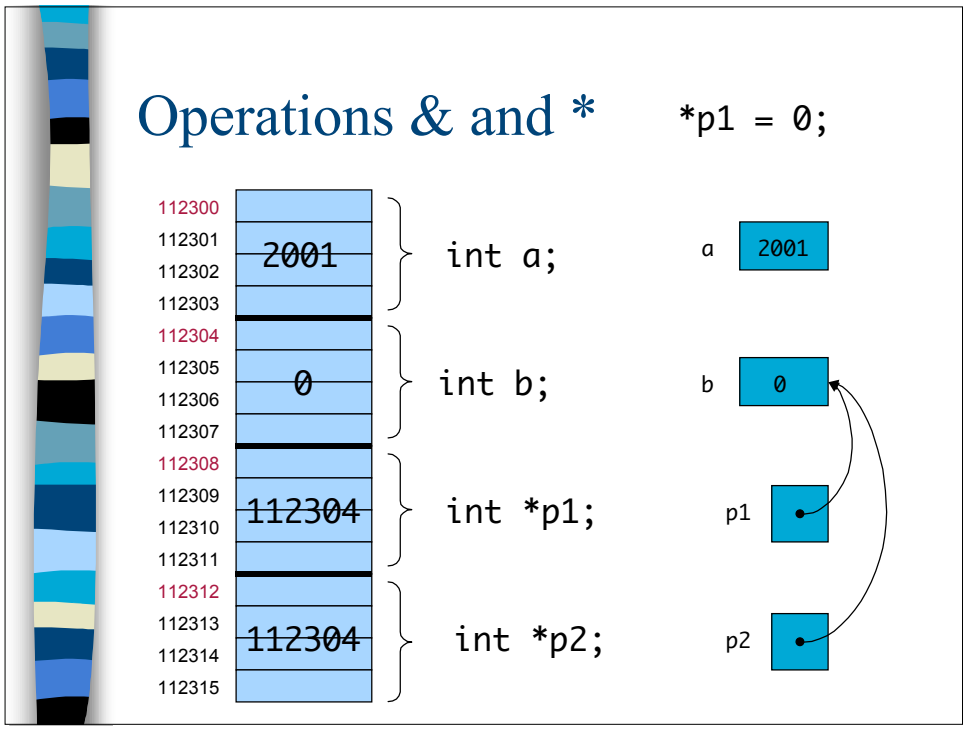


Operations & and *

p2 = &b ;

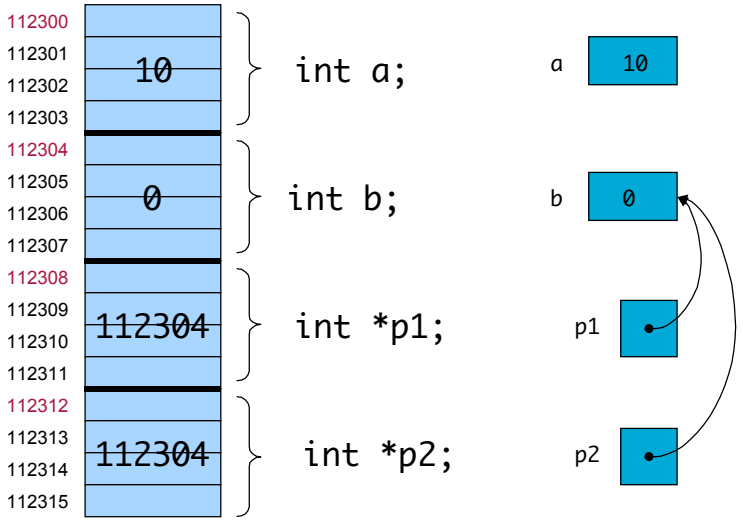



Operations & and *
p1 = p2;
112300
112301
112302
112303
2001
int a;
a
2001
112304
112305
112306
112307
2000
int b;
b
2000
112308
112309
112310
112311
112304
int *p1;
p1
112312
112313
112314
112315
112304
int *p2;
p2


Operations & and *
*p1 = 0;
112300
112301
112302
112303
2001
int a;
a
2001
112304
112305
112306
112307
0
int b;
b
0
112308
112309
112310
112311
112304
int *p1;
p1
112312
112313
112314
112315
112304
int *p2;
p2

Operations & and *

```
a = *p2 + 10;
```



Pointers and function arguments

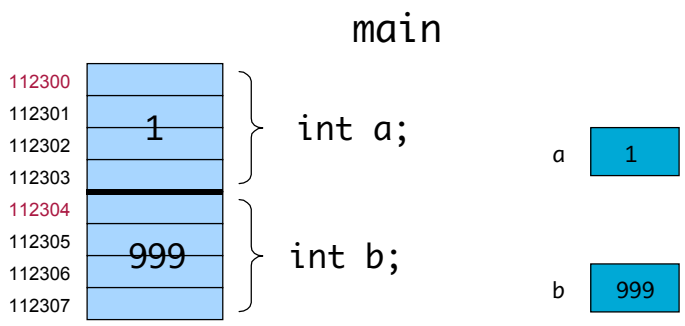
```
/* bad example of swapping
a function can't change
   parameters */
void bad_swap(int x,
              int y)
{
   int temp;

   temp = x;
   x = y;
   y = temp;
}
```

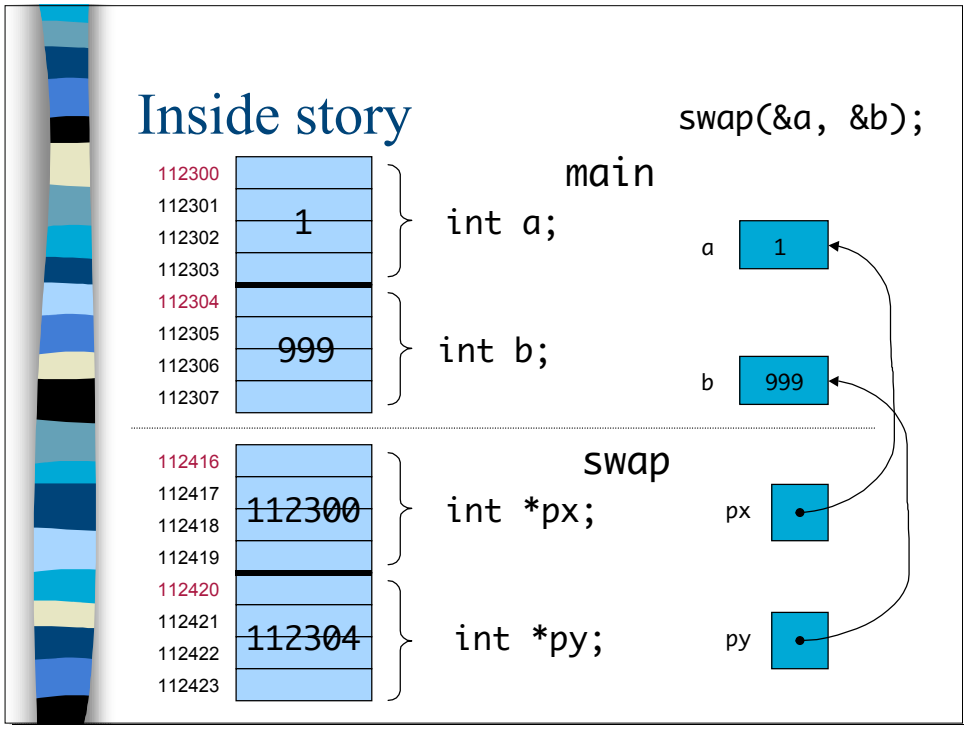
```
/* good example of swapping
   - a function can't change
   parameters but if a
   parameter is a pointer it
   can change the value it
   points to */
void good_swap(int *px,
               int *py)
{ int temp;

   temp = *px;
   *px = *py;
   *py = temp;
}
```

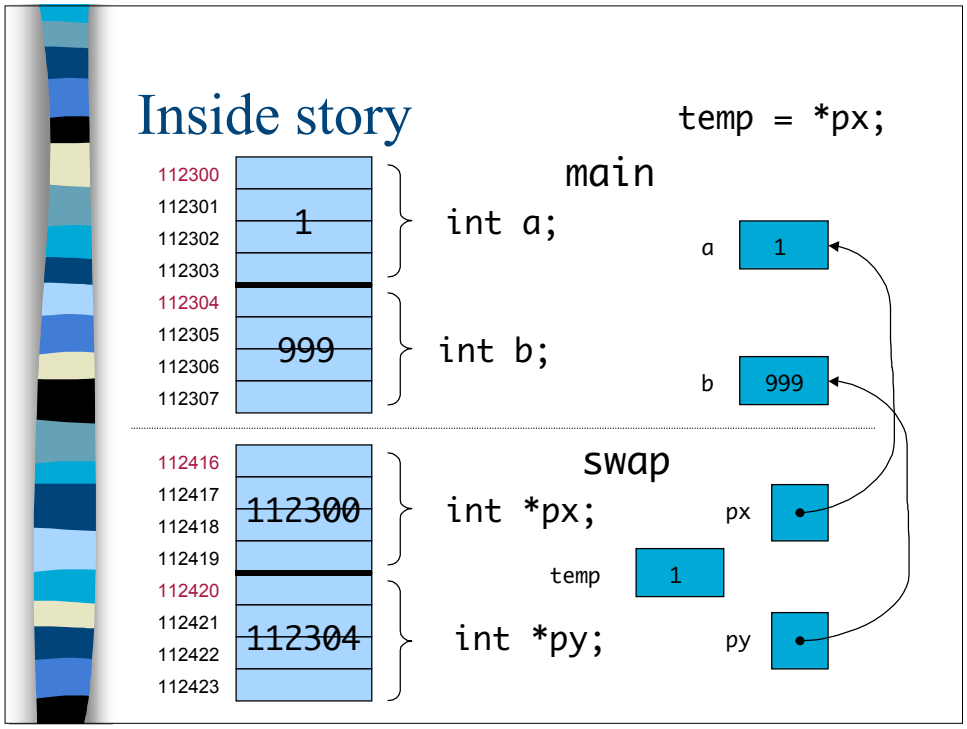
Pointers and function arguments

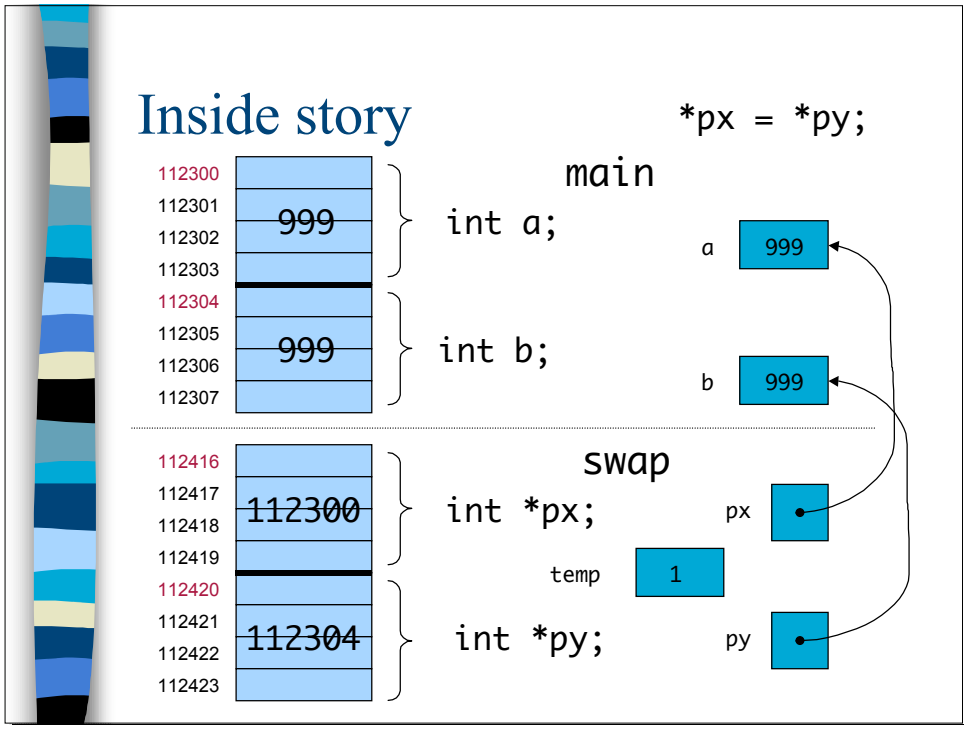
```
#include <stdio.h>
void bad_swap(int x, int y);
void good_swap(int *p1, int *p2);

void main() {
   int a = 1, b = 999;

   printf("a = %d, b = %d\n", a, b);
   bad_swap(a, b);
   printf("a = %d, b = %d\n", a, b);
   good_swap(&a, &b);
   printf("a = %d, b = %d\n", a, b);
}
```

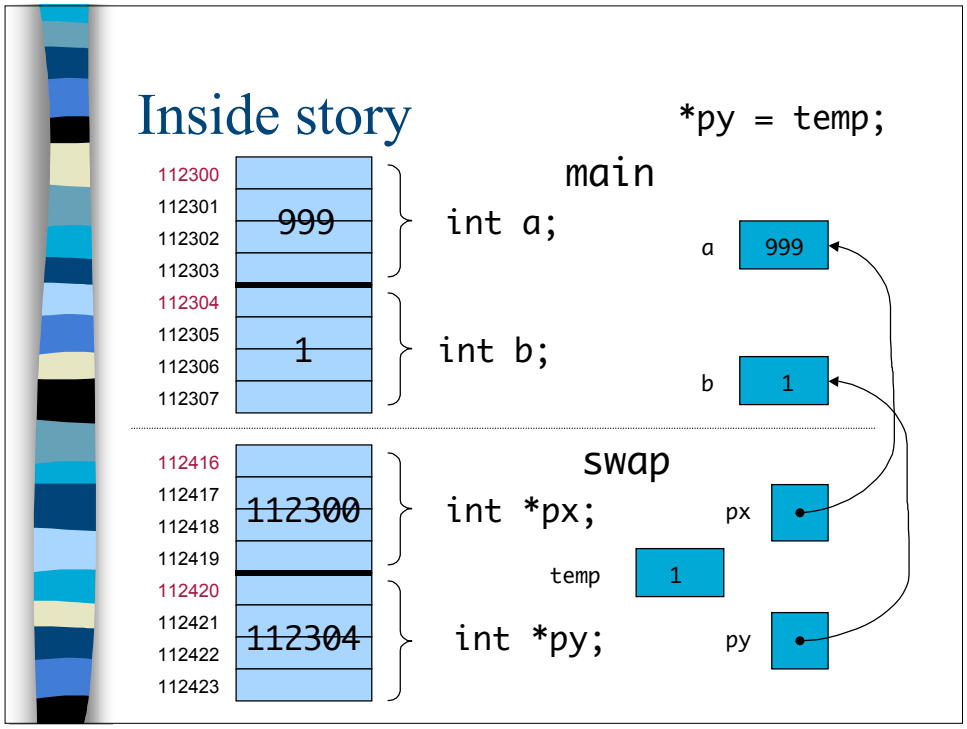
Inside story



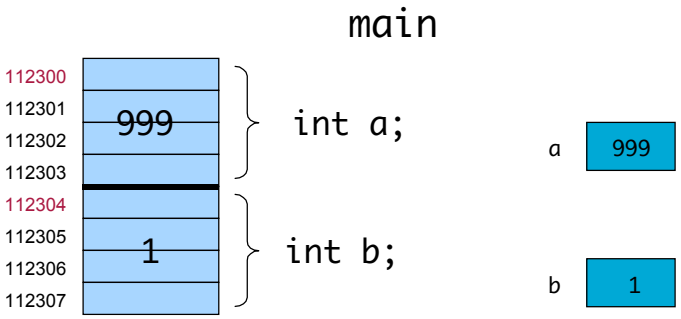

Inside story
swap(&a, &b);
main
112300
112301
112302
112303
1
int a;
a
1
112304
112305
112306
112307
999
int b;
b
999
swap
112416
112417
112418
112419
112300
int *px;
px
112420
112421
112422
112423
112304
int *py;
py


Inside story
temp = *px;
main
112300
112301
112302
112303
1
int a;
a
1
112304
112305
112306
112307
999
int b;
b
999
swap
112416
112417
112418
112419
112300
int *px;
px
temp
1
112420
112421
112422
112423
112304
int *py;
py


Inside story
*px = *py;
main
112300
112301
112302
112303
999
int a;
112304
112305
112306
112307
999
int b;
a
999
b
999
swap
112416
112417
112418
112419
112300
int *px;
px
temp
1
112420
112421
112422
112423
112304
int *py;
py


Inside story
*py = temp;
main
112300
112301
112302
112303
999
int a;
112304
112305
112306
112307
1
int b;
a
999
b
1
swap
112416
112417
112418
112419
112300
int *px;
px
temp
1
112420
112421
112422
112423
112304
int *py;
py

Inside story



The magical & in scanf()

- `scanf` is able to put the input into a variable because we pass its *address*

```
int capital = 0;
scanf("%d", &capital);
```

- What happens without &?

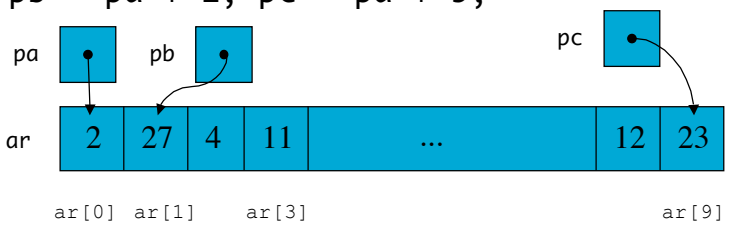
```
scanf("%d", capital);
```

- `scanf` thinks that the value of capital is an address where it has to place data!

Pointers and Arrays

- We can address array elements

```
int ar[10];
int *pa, *pb, *pc;
pa = &ar[0] /* same as pa = ar */
pb = pa + 1; pc = pa + 9;
```



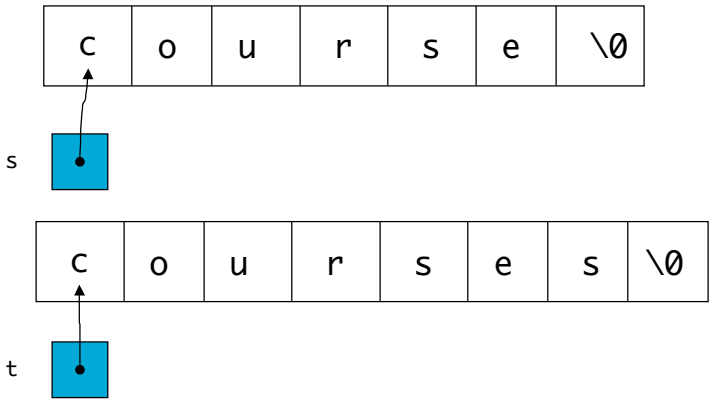
Address Arithmetic

- We can add and subtract integers to/from pointers - the result is a pointer to another element of this type
  - int *pa; char *s;
  - s-1 ⇨ points to char before s (1 subtracted)
  - pa+1 ⇨ points to next int (4 added!)
  - s+9 ⇨ points to 9th char after s (9 added)
  - ++pa ⇨ increments pa to point to next int

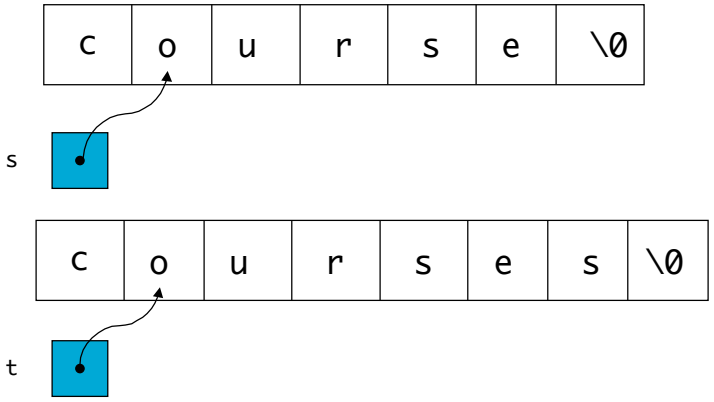
Example 2: String Comparison

```
#define MAXLEN 100
int strcmp(char *p1, char *p2);
int getline(char *line, int max);
void main() {
   char a[MAXLEN], b[MAXLEN];
   int comp;
   getline(a, MAXLEN);
   getline(b, MAXLEN);
   comp = strcmp(a, b);
   if(comp > 0)
        printf("First line is greater than second\n");
   else if (comp < 0)
        printf("First line is less than second\n");
   else printf("These lines are equal!\n");
}
```

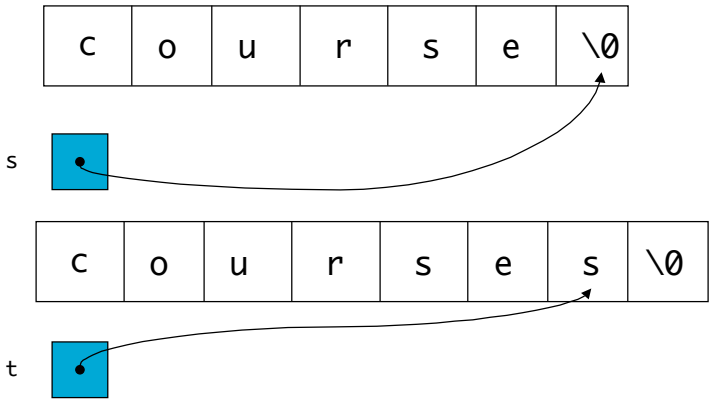
Example 2: String Comparison



Example 2: String Comparison



Example 2: String Comparison



Compare string comparisons

```
/* pointer version */
/* strcmp: return <0 if s<t,
   0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
   for( ; *s == *t; s++, t++)
        if(*s == '\0')
                return 0;
   return *s - *t;
}
```

```
/* array version */
/* strcmp: return <0 if s<t,
   0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
   int i;

   for(i = 0 ; s[i] == t[i];
                i++)
        if(s[i] == '\0')
                return 0;
   return s[i] - t[i];

}
```

Before Next Lecture:

- Do reading assignment
- Perry: Chapter 24; First reading of Chapter 25, also external materials
- Run Classroom Examples
- Knowledge Sea pair homework: find 4 pages that are good to read for a specific lecture. Assign praise comment
- Read other relevant material on pointers in Knowledge Sea