

## **Announcements**

- **Midterm is Friday, 2:50-4:10pm, Van Dyck 211**
  - read the **Examination Policy** before the exam (the link is off of the *Academic Integrity* page); please arrive early, bring id, etc...

1

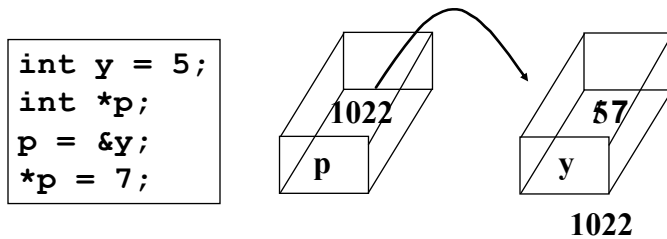
## **Imperative Programming: C**

- **Pointers (continued)**
- **Functions**
- **Structures**
- **Memory allocation**
  - malloc
- **Function pointers**

2

# Pointers

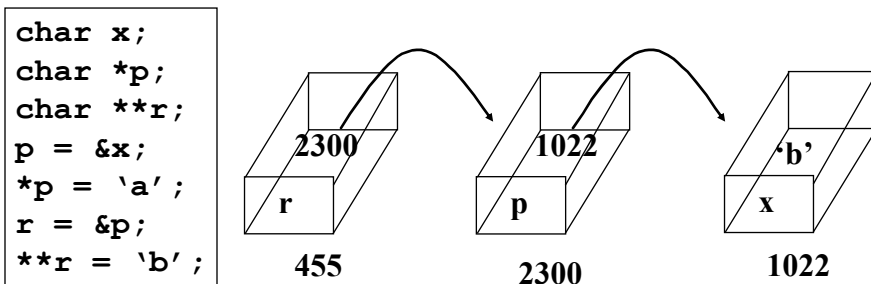
- **Pointer: A variable whose value is an L-value.**
  - declaration: `int *p;`
  - address-of operator `&` : returns L-value of variable
  - dereference operator for a pointer `*` : obtains R-value at that address value



3

# Pointers

- **Pointers can point to pointer variables (multi-level).**



- **Assignment Rule: `<lhs> = <rhs>`**
  - lhs must be an L-value; rhs is evaluated, with all L-values dereferenced once (unless blocked by `&`)

4

# Pointers

- **Pointer Expressions:**

```
int j = 5; int h = 10;
```

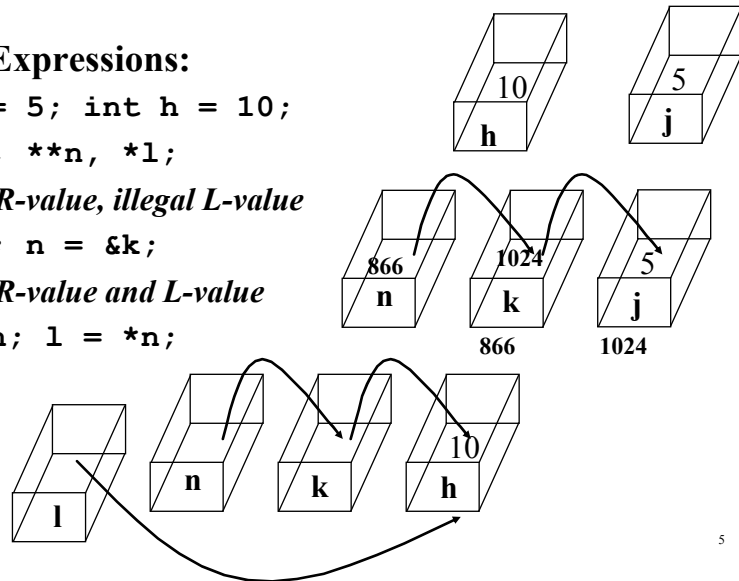
```
int *k, **n, *l;
```

*&j, legal R-value, illegal L-value*

```
k = &j; n = &k;
```

*\*n, legal R-value and L-value*

```
*n = &h; l = *n;
```



5

# Pointers and Arrays

- **An array name is considered pointer to first element:** `int a[5];`
  - `a` is pointer to `a[0]`
  - `pa = &a[0]` and `pa = a` mean the same thing
  - `a+1` means L-value of `a[0]` plus as many bytes as are needed to store value of elements of `a`'s type
    - Pointer arithmetic is an address calculation with respect to the underlying representation
- **An array name is a constant pointer**
  - `a++` and `a = pa` are illegal

6

## Problems with Pointers

- **Uninitialized pointers**

- `int *a; *a = 12;`
- Get *segmentation fault* when value in `a` is meaningless address
- Can actually store 12 into some memory location accessible to your program, whose address corresponds to the random bits in `a`

- **Aliases can be confusing**

```
int a;  
int *d;  
d = &a;  
a = 5;  
*d = 10 + *d;  
a → 15
```

7

## Problems with Pointers

- **NULL doesn't point to anything by definition so should not be de-referenced**

- `int *a; a = NULL;`
- `if (a == NULL) ...`  
    */\* tests for a NULL pointer value \*/*

- **Multiple level pointers**

- Can be used as L-values or R-values

8

# Pointer example

```
#include <stdio.h>
/* Show multiple levels of dereference*/
main()
{
    int j = 99;
    int *q;
    int **s;

    q = &j;          /* q = j is ILLEGAL */
    s = &q;          /* s = q is ILLEGAL */

    /* Here, **s, *q, j are aliases for the same
       storage location */
    printf(" %d %d %d\n", **s, *q, j);
}

% gcc multipleptr.c
% a.out
99 99 99
```

9

# Functions

- **Prototype:**

```
int mult(int, int); int square(int n);
```

- often in a \*.h or “header” file
- used by compiler for type checking

- **Definition:**

```
int square(int n) {return n*n;}
```

*(formal)  
parameter*

- **Invocation:**

```
int b = 3; c = square(b);
```

*argument*

10

# Functions

- **Parameter Passing uses “Call by Value”**
  - Value of actual argument is copied into formal parameter
  - Example:

```
void swap(int a,int b)
    {int temp; temp=a; a=b; b=temp;}
...
int x=3; int y=5;
swap(x,y);
printf(“%d %d”,x,y); → 3 5    Why?
```

11

# Functions

- **To get the effect of “Call by Reference” use pointers**
  - Example:

```
void swap(int *a, int *b)
    {int temp; temp=*a; *a=*b; *b=temp;}
...
int x=3;
int y=5;
swap( &x, &y );
printf(“%d %d”,x,y); --> 5 3
```

12

# Structures

- **Structure: Aggregate object with named fields**

- Declaration:

```
struct EMPLOYEE {  
    int age;  
    double payrate;  
} joe, mary;  
struct EMPLOYEE bill = {35, 65000.00};
```

- Accessed as:

```
joe.age = 45;  
bill.payrate *= 1.1;
```

13

# Structures

- **Defining new types:**

```
typedef float Celsius;  
typedef float Fahrenheit;
```

```
enum {BROWN, BLOND} hisHair, herHair;
```

```
typedef enum  
    {BLACK, BROWN, RED, BLOND, GRAY}  
    HairColors;
```

14

# Structures

- **Defining new structure types:**

```
typedef struct {  
    float x-coord;  
    float y-coord;  
} Point;  
  
struct POLYGON {  
    Point vertices[10];  
    int N;  
} p;
```