# Arrays and Pointers

CSE 2031

Fall 2010

## Arrays

- Grouping of data **of the same type.**
- Loops commonly used for manipulation.
- Programmers set array sizes explicitly.

## Arrays

- Syntax

```
type name[size];
```

- Examples

```
int bigArray[10];
double a[3];
char grade[10], oneGrade;
```

3

## Arrays

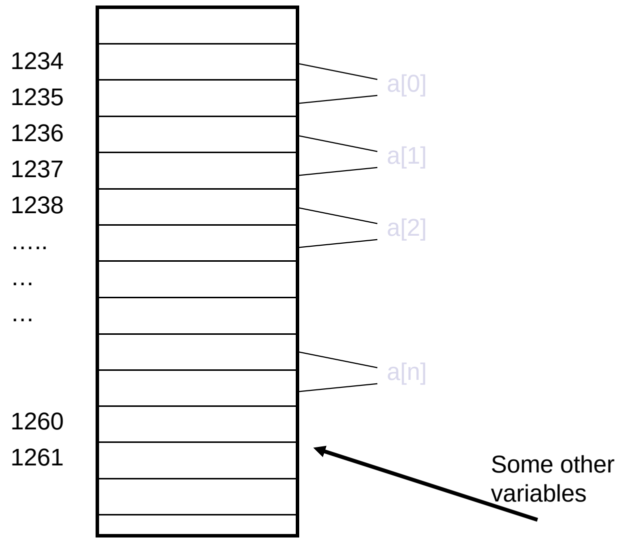
- Defining an array: allocates memory

```
int score[5];
```

  - Allocates an array of 5 integers named "score"
- Individual parts can be called:
  - Indexed or subscripted variables
  - "Elements" of the array
- Value in brackets called index or subscript
  - Numbered from 0 to (size – 1)

4

## Arrays Stored in Memory



5

## Initialization

- In declarations enclosed in curly braces

**int a[5] = {11,22};**

Declares array a and initializes first two elements and all remaining set to zero

**int b[ ] = {1,2,8,9,5};**

Declares array b and initializes all elements and sets the length of the array to 5

6

## Array Access

```
x = ar[2];
ar[3] = 2.7;
```

- What is the difference between
  `ar[i]++, ar[i++], ar[++i]` ?

## Strings

- No **`string`** type in C
- String = array of char
- char gretings[ ] = “Hello”

| | | | | | |
|---|---|---|---|---|---|
| H | e | l | l | o | \0 |

# Pointers

CSE 2031

Fall 2010

9

## Pointers and Addresses (5.1)

- Memory address of a variable
- Declared with data type, * and identifier
  type * pointer_var1, * pointer_var2, …
- Example.
  double * p;
  int *p1, *p2;
- There has to be a * before EACH of the pointer variables

10

## Pointers and Addresses (cont.)

- Use the **"address of"** operator (&)
- General form:

pointer_variable = &ordinary_variable

Name of the pointer — Name of ordinary variable

11

## Using a Pointer Variable

- Can be used to access a value
- Unary operator * used

  * pointer_variable

  - In executable statement, indicates value

- Example

```
int *p1, v1;
v1 = 0;
p1 = &v1;
*p1 = 42;
printf(“%d\n“,v1);
printf(“%d\n,*p1);
```

**Output:**
```
42
42
```

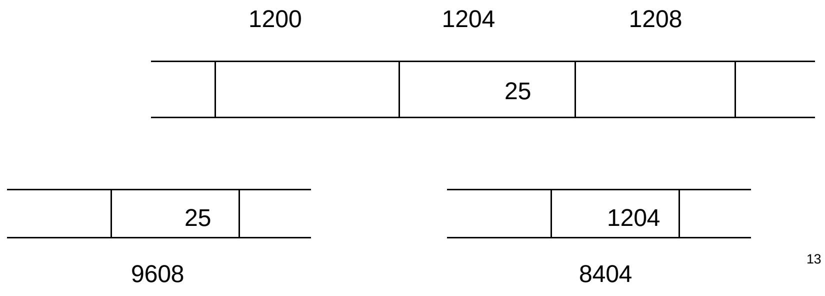
12

## Pointer Variables

**int x,y;**

**int * z;**

```
x = 25;
y = x;
z = &x;
```



13

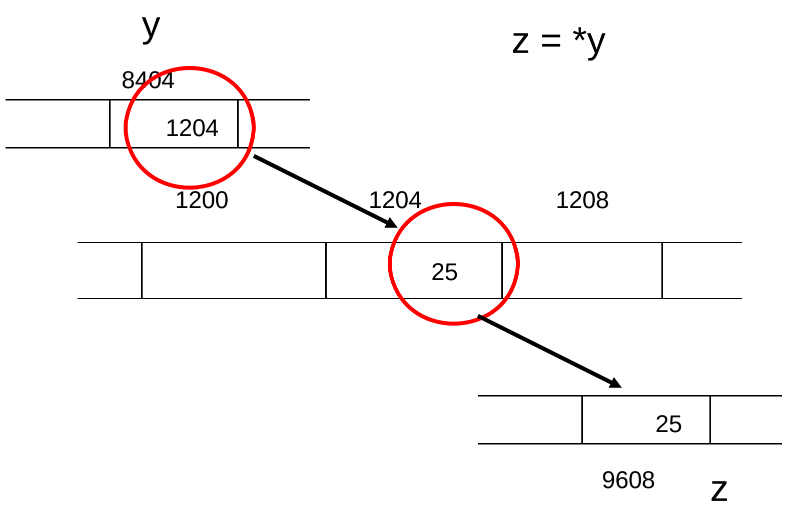
## Pointer Variables (cont.)

~~z= 0x12345A~~ BAD idea

Instead, use z = &x

14

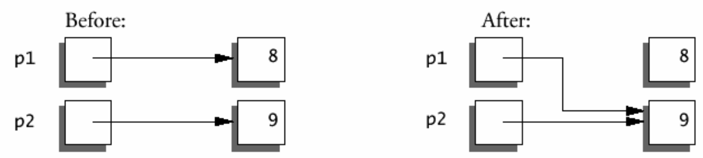
## Pointer Types



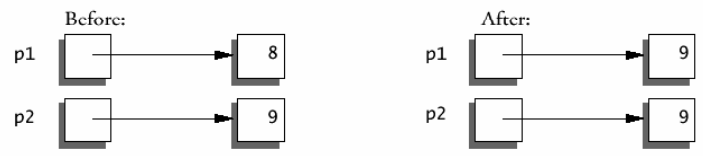
15

## Another Example of Pointers

```
p1 = p2;
```



```
*p1 = *p2;
```



16

## More Examples

```
int x = 1, y = 2, z[10], k;
int *ip;
ip = &x;    /* ip points to x*/
y = *ip;    /* y is now 1 */
*ip = 0;    /* x is now 0 */
z[0] = 0;
ip = &z[0]; /* ip points to z[0] */
for (k = 0; k < 10; k++)
  z[k] = *ip + k;
*ip = *ip + 100;
++*ip;
(*ip)++;    /* How about *ip++ ??? */
```

17

## Pointers and Function Arguments (5.2)

Write a function that swaps the contents of two integers a and b.

```
void main( ) {
   int a, b;
   /* Input a and b */
   swap(a, b);
   printf("%d %d", a, b);
{
```

C passes arguments to functions by values.

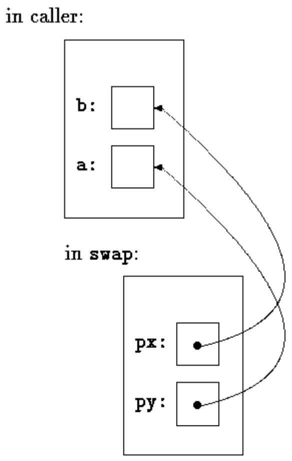
```
void swap(int x, int y)
{
  int temp;
  temp = x;
  x = y;
  y = temp;
}
```

18

## The Correct Version

```
void swap(int *px, int *py)
{
   int temp;
   temp = *px;
   *px = *py;
   *py = temp;
}

void main( ) {
    int a, b;
    /* Input a and b */
    swap(&a, &b);
    printf("%d %d", a, b);
{
```



## Next time ...

- Pointers, part 2 (Chapter 5)