2D Arrays

Passing Pointers as Function Arguments and Return Values

Based on slides by Dianna Xu

Outline

- Arrays in terms of pointers
  - Multi-dimensional
  - Pointer arrays
- Pointers as function arguments
- Pointers as function return value

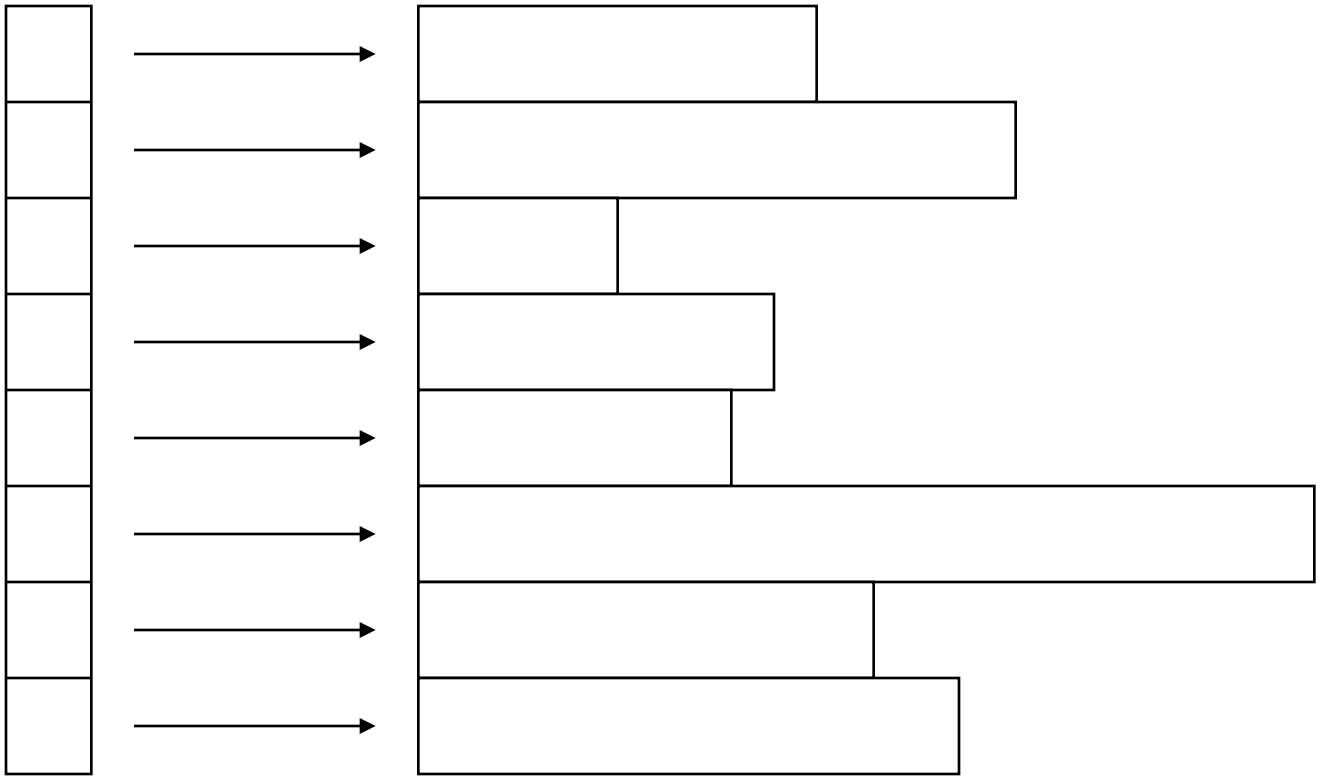
Pointer Arrays: Pointer to Pointers

- Pointers can be stored in arrays
- Two-dimensional arrays are just arrays of pointers to arrays.
  - **`int a[10][20]; int *b[10];`**
  - Declaration for b allows 10 **`int`** pointers, with no space allocated.
  - Each of them can point to an array of 20 integers
  - **`int c[20]; b[0] = c;`**
  - What is the type of **`b`**?

2D Arrays

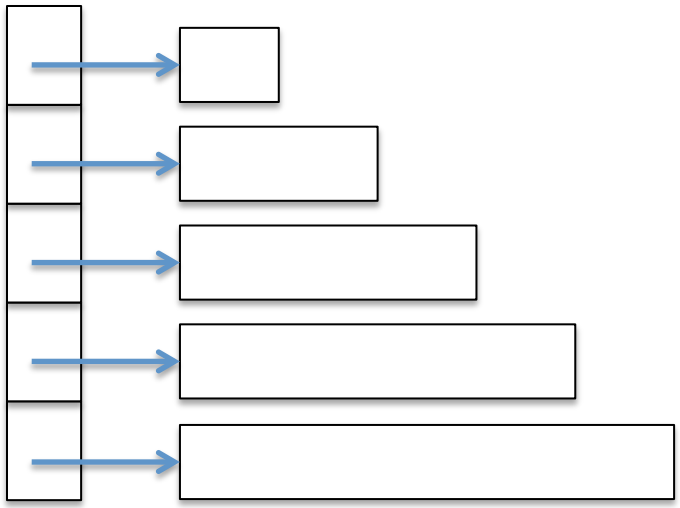
```
int rows = 10; int cols = 10;

int b[rows][cols];

int** a = new int*[rows];
for(int i = 0; i < rows; ++i)
    a[i] = new int[cols];
```

Ragged Arrays



Ragged 2D Arrays

```
int rows = 10;
int** a = new int*[rows];
for(int i = 0; i < rows; ++i)
    a[i] = new int[i];
```



Pointer Safety

- After you deallocate the memory associated with a pointer, set it to NULL for safety
- C++11 includes the nullptr keyword for a null pointer

```
int* p = new int[50];
...
delete[] p;
p = nullptr;
```

- While setting the pointer to 0 works, this isn't type-safe

Pointers are Passed by Value to Functions

- A copy of the pointer's value is made – the address stored in the pointer variable
- The copy is then a pointer pointing to the same object as the original parameter
- Thus modifications via de-referencing the copy STAYS.

Function Arguments

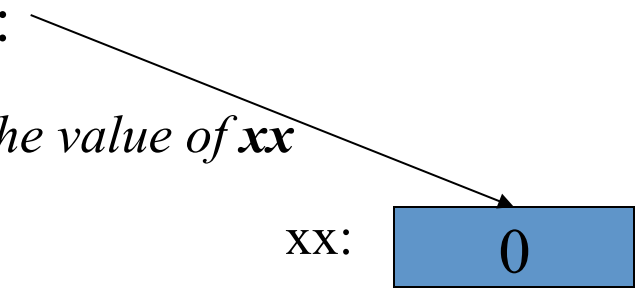
- **`x`** and **`y`** are copies of the original, and thus **`a`** and **`b`** can not be altered.

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

int main() {
   int a = 1, b = 2;

   swap(a, b);
   return 0;
}
```

Wrong! Could use pass-by-reference, but let's use pointers!

Recall: Pass by Reference

```
// pass-by-reference (pass a reference to the argument)
int f(int& a) { a = a+1; return a; }

int main() {
    int xx = 0;
    f(xx);                      // f() changed the value of xx
    cout << xx << endl;         // writes 1
}
```



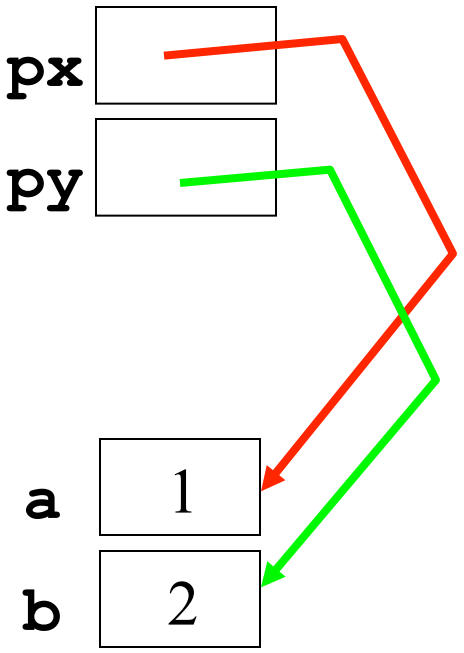
Really, you should only use pass-by-const-reference:

```
void g(int a, int& r, const int& cr) { ++a; ++r; int x = cr; ++x; }

int main() {
    int x = 0, y = 0, z = 0;
    g(x,y,z);   // x==0; y==1; z==0
}
// const references are very useful for passing large objects
```

Pointers and Function Arguments

- Passing pointers – **a** and **b** are passed by pointers (the pointers themselves **px** and **py** are still passed by value)

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

int main() {
   int a = 1, b = 2;

   swap(&a, &b);
   return 0;
}
```



Use Pointers to Modify Multiple Values in a Function

```
void decompose(double d, int *i, double *frac) {
  *i = (int) d;
  *frac = d - *i;
}

int main() {
  int int_part;
  double frac_part, input;

  scanf("%lf", &input);
  decompose(input, &int_part, &frac_part);
  printf("%f decomposes to %d and %f\n",
         *int_part, *frac_part);
  return 0;
}
```

Pass by Reference

- Do not equate pass-by-reference with pass-by-pointer
- The pointer variables themselves are still passed by value
- The objects being pointed to, however, are passed by reference
- In a function, if a pointer argument is de-referenced, then the modification indirectly through the pointer will stay

Modification of a Pointer

```
void g(int **ppx, int *py) {
   *ppx = py;
}

int main() {
   int x = 1, y = 2, *px;
   px = &x;
   g(&px, &y);
   printf("%d", *px); // will print 2
}
```

Pointer as Return Value

- We can also write functions that return a pointer
- Thus, the function is returning the memory address of where the value is stored instead of the value itself
- Be very careful not to return an address to a temporary variable in a function!!!

Example of Returning a Pointer

```
int* max(int *x, int *y) {
  if (*x > *y)
    return x;
  return y;
}

int main() {
  int a = 1, b = 2, *p;

  p = max(&a, &b);
  return 0;
}
```

Example of Returning a Pointer

- How do these two code samples compare?
  - Hint: **x** and **y** are copies of the original, so what are **&x** and **&y**?

```
int* max(int *x, int *y) {
  if (*x > *y)
    return x;
  return y;
}

int main() {
  int a = 1, b = 2, *p;

  p = max(&a, &b);
  return 0;
}
```

```
int* max(int x, int y) {
  if (x > y)
    return &x;
  return &y;
}



p = max(a, b);
```

Arrays as Arguments

- Arrays are passed by reference
- Modifications stay

```
#define SIZE 10

void init(int a[]) {
  int i;

  for(i = 0;i<SIZE;i++){
    a[i] = 0;
  }
}

int main() {
  int a[SIZE];

  init(a);
  return 0;
}
```

```
/* equivalent pointer alternative */
void init(int *a) {
  int i;

  for(i = 0;i<SIZE;i++){
    *(a+i) = 0;
  }
}
```

Pointer Arithmetic: Combining * and ++/--

- ++ and -- has precedence over *
  - a[i++] = j;
  - p=a; *p++ = j; <==> *(p++) = j;

  - *p++; value: *p, inc: p
  - (*p)++; value: *p, inc: *p
  - ++(*p); value: (*p)+1, inc: *p
  - *++p; value: *(p+1), inc: p