Topic 2-3

Arrays

Welcome!

- In this lecture we will discuss:
  - Collections of memory
  - Arrays & pointers!

Review

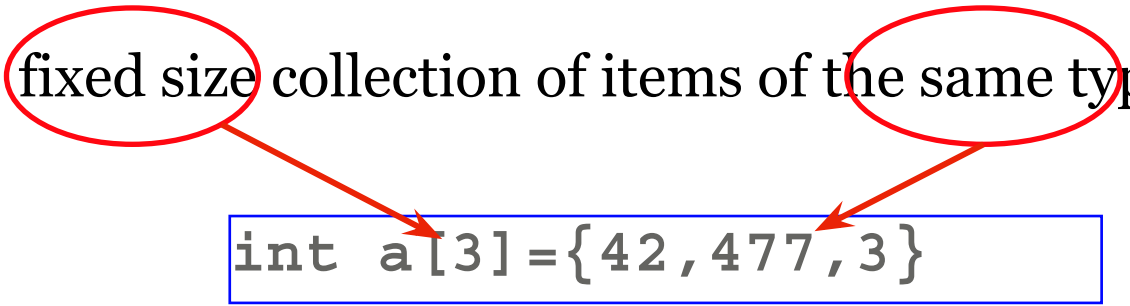
- We saw in the previous topic that pointers are powerful.
- We saw that we used pointers instead of “normal” variables because:
  - it saved us memory and
  - made our programs faster

Arrays

- Fixed size collection of variables of the same type
- also (most cool for pointers): **contiguous**
- The name of the array is a pointer to its first element.

Array

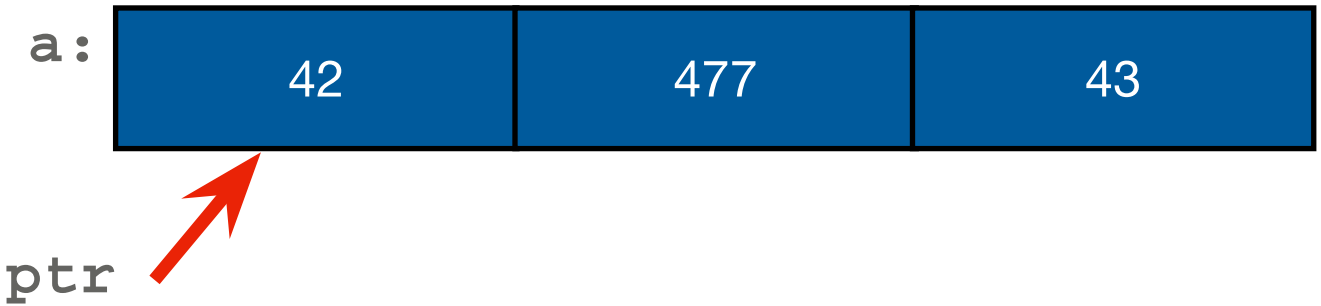
- fixed size collection of items of the same type



- all elements of array “a” are of type int; indexing starts at 0
- what is a[0]? a[1]? a[2]?
- what about a[-1] ? or a[3]?

5

Pointers and Arrays

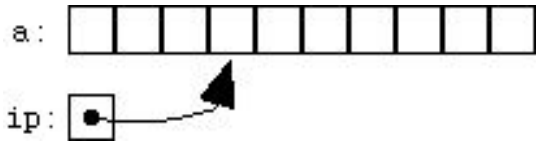



```
int *ptr;
int a[3]={42,477,3}
ptr = &a[0];
```

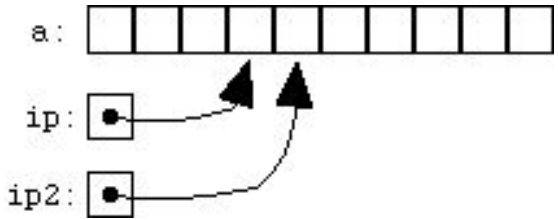
- value of *ptr ?
- value of *(ptr + 1) ?
- value of *(ptr + 2) ?

More arithmetic

```
what does this do?

int a[10];

int *ip;

ip = &a[3];
```



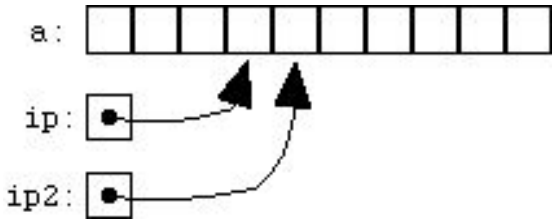

```
what does this do?
int a[10];
int *ip, *ip2;
ip = &a[3];
ip2 = ip + 1;
```



http://www.eskimo.com/~scs/cclass/notes/sx10b.html

And more ...

```
what does this do?
*(ip + 1) = 5;
*ip2 = 345;
```



Demo 6

```cpp
#include <iostream>

using namespace std;

int main()
{
    int *ptr;
    int a[3] = {43,477,34};

    /* address of a[0] is assigned to ptr */
    ptr = &a[0];

    cout << "Value of a[0] : "<< *(ptr+0) << endl;
    cout << "Value of a[1] : "<< *(ptr+1) << endl;
    cout << "Value of a[2] : "<< *(ptr+2) << endl;

    return 0;
}
```

And more …

- Arrays are not variables
  - **they are constants,**
  - they evaluate to the address of their first element.
- Pointer and array notation can be equivalent:

```
Address of A[0]:
A
&A[0]
A+0
Value of A[0]:
A[0]
*(&A[0])
*(A+0)
```

Demo 7

```cpp
int main()
{
    int a[10] = {1,2,3,4,5,6,7,8,9,10};
    int *ip;
    ip = &a[2];

    cout << "array address: " << &a[0] << endl;
    //or
    cout << "array address: " << a << endl;

    for (int i = 0 ; i < 10 ; i++ ){
        cout << "a[" << i << "] = " << a[i] << endl;
    }
    cout << "---------------\n";
    *ip = 500;

    for (int i = 0 ; i < 10 ; i++ ){
        cout << "a[" << i << "] = " << a[i] << endl;
    }

    return 0;
}
```

Pointers and Arrays as Parameters

- Pointer or array notation is a choice of style:

```
void swap(int a[],int b[])
{
    int t = a[0] ; a[0] = b[0] ; b[0] = t ;
}

void swap(int *a,int *b)
{
    int t = *a ; *a = *b ; *b = t ;
}
```

Demo 9

```
void swapAA(int a[],int b[])
{
    int t = a[0] ; a[0] = b[0] ; b[0] = t ;
}

void swapAP(int a[],int b[])
{
    int t = *a ; *a = *b ; *b = t ;
}

void swapPA(int *a,int *b)
{
    int t = a[0] ; a[0] = b[0] ; b[0] = t ;
}

void swapPP(int *a,int *b)
{
    int t = *a ; *a = *b ; *b = t ;
}

int main()
{
    int x = 5,y = 2 ;

    cout << "x " << x << ", y " << y << endl;
    swapAA(&x,&y) ;
    cout << "x " << x << ", y " << y << endl;
    swapAP(&x,&y) ;
    cout << "x " << x << ", y " << y << endl;
    swapPA(&x,&y) ;
    cout << "x " << x << ", y " << y << endl;
    swapPP(&x,&y) ;
    cout << "x " << x << ", y " << y << endl;
}
```

Pointers and Arrays as Parameters

- Single dimensional arrays need to describe their size
  - How many elements are there ?
- Multi dimensional arrays need to describe their size
  - How many elements are there ?
  - How large are the elements ?

```
a. int sum1d(int n, int a[])
b. int sum1d(int n, int *a)   // a
   // *** choose carefully
c. int sum2d(int n, int a[][2])
d. int sum2d(int n, int *a[2])  // not c
e. int sum2d(int n, int (*a)[2])     // c
f. int sum2d(int n, int **a, int n2) // d?
```

Demo 10

```
int main()
{
    int a1[] = {1,2,3} ;
    int a2[] = {1,2,3} ;
    int a3[] = {1,2,3} ;
    int aa1[][2] = {{1,2},{3,4}} ;
    int aa20[] = {1,2,6} ;
    int aa21[] = {3,4,8} ;
    int *aa2[] = {aa20,aa21} ;
    int aa3[][2] = {{1,2},{3,4}} ;

    cout << "sum of a1 is  " <<  sum1dAA(3,a1) << endl;
    cout << "sum of a2 is  " <<  sum1dPA(3,a2) << endl;
    cout << "sum of a3 is  " <<  sum1dPP(3,a3) << endl;
    cout << "sum of aa1 is " <<  sum2dAA(2,aa1) << endl;
    cout << "sum of aa2 is " <<  sum2dPA(3,aa2) << endl;
    cout << "sum of aa3 is " <<  sum2dQA(2,aa3) << endl;
    cout << "sum of aa4 is " <<  sum2dPP(2,aa2,3) << endl;
}
```

How do you handle pointers?

- … carefully

Smart Pointers

- We know we have to keep track of our pointers.
  - Why?

Smart Pointers

- We know we have to keep track of our pointers.
  - Because we have to make sure we release memory again.
- Remember the stack and the heap?

Smart Pointers

- Smart pointers are a way of wrapping up pointers to do some of the work for you.
- We'll return to this when we talk about objects but what do you think you'd like help with for pointers?