Pointers

NEW YORK UNIVERSITY
ABU DHABI

- Special case of bounded-size natural numbers
  - Maximum memory limited by processor word-size
  - 2^{32} bytes = 4GB, 2^{64} bytes = 16 exabytes

- A pointer is just another kind of value
  - A basic type in C++ `int *ptr;`

The variable “ptr” is a pointer to an “int”.

Pointer Operations in C++

NEW YORK UNIVERSITY ABU DHABI

- Creation
  - `&` *variable* — Returns variable's memory address
- Dereference
  - `*` *pointer* — Returns contents stored at address
- Indirect assignment
  - `*` *pointer* = *val* — Stores value at address

- Of course, still have...

- Assignment
  - *pointer* = *ptr* — Stores pointer in another variable

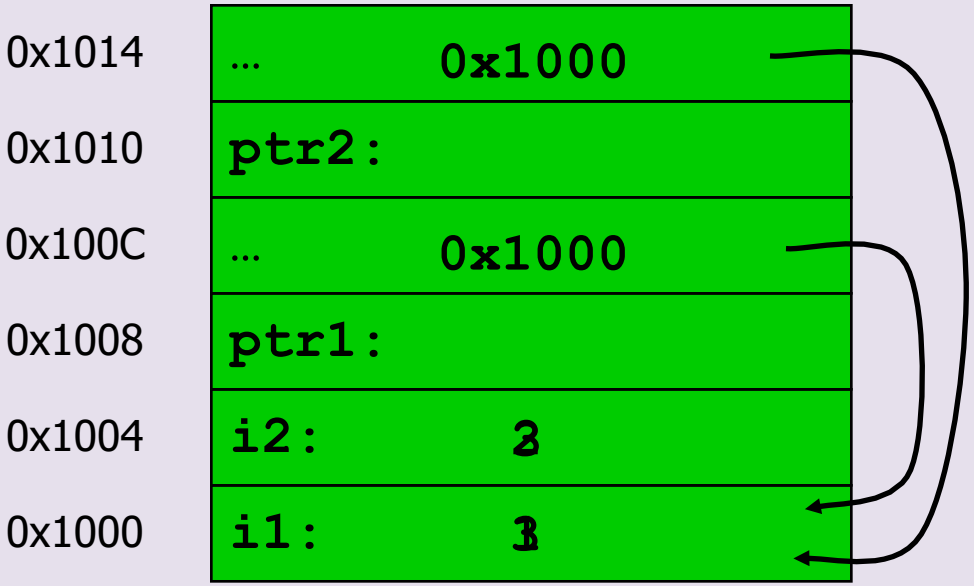
Using Pointers

```
int  i1;
int  i2;
int *ptr1;
int *ptr2;

i1 = 1;
i2 = 2;

ptr1 = &i1;
ptr2 = ptr1;

*ptr1 = 3;
i2 = *ptr2;
```



Using Pointers (cont.)

```
int  int1     = 1036;   /* some data to point to  */
int  int2     = 8;

int *int_ptr1 = &int1;  /* get addresses of data  */
int *int_ptr2 = &int2;

*int_ptr1 = int_ptr2;

*int_ptr1 = int2;
```

What happens?

Type check warning: `int_ptr2` **is not an** `int`

`int1` **becomes 8**

Using Pointers (cont.)

```
int  int1     = 1036;   /* some data to point to  */
int  int2     = 8;

int *int_ptr1 = &int1;  /* get addresses of data  */
int *int_ptr2 = &int2;

int_ptr1 = *int_ptr2;

int_ptr1 = int_ptr2;
```

What happens?

Type check warning: `*int_ptr2` **is not an** `int *`

Changes `int_ptr1` **– doesn't change** `int1`

Example

```
int x = 1, y = 2, z[10];
int *ip;                // ip is a pointer to an int, so it can point
                        // to x, y, or an element of z
ip = &x;                // ip now points at the location where x is stored
y = *ip;                // set y equal to the value pointed to by ip, or y = x
*ip = 0;                // now change the value that ip points to to 0,
                        // so now x = 0 but notice that y is unchanged
ip = &z[0];             // now ip points at the first location in the array z

*ip = *ip + 1;          // the value that ip points to (z[0]) is incremented
```

```
int x, *y, z, *q;
x = 3;
y = &x;                 // y points to x
printf("%d\n", x);      // outputs 3
printf("%d\n", y);      // outputs x's address, will seem like a random number
printf("%d\n", *y);     // outputs what y points to, or x (3)
printf("%d\n", *y+1);   // outputs 4 (print out what y points to + 1)
printf("%d\n", *(y+1));//this outputs the item after x in memory – what is it?
z = *(&x);              // z equals 3 (what &x points to, which is x)
q = &*y;                // q points to x, just like y; note *& and &* cancel out
```

Pointer Arithmetic

NEW YORK UNIVERSITY ABU DHABI

pointer + number *pointer – number*

E.g., *pointer* + 1 adds something to a pointer

```
char   *p;
char    a;
char    b;

p = &a;
p += 1;
```

In each, p now points to b
(Assuming compiler doesn't reorder variables in memory)

```
int   *p;
int    a;
int    b;

p = &a;
p += 1;
```

Adds 1*sizeof(char) to the memory address

Adds 1*sizeof(int) to the memory address

Pointer arithmetic should be used cautiously

The Simplest Pointer in C++

NEW YORK UNIVERSITY
ABU DHABI

- Special constant pointer `NULL`
  - Points to no data
  - Dereferencing illegal – causes *segmentation fault*

Pointer as Parameters

NEW YORK UNIVERSITY
ABU DHABI

```
#include <iostream>
using namespace std;

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

int main ()
{
  int m=3,n=4;
  cout<< m <<" " << n<<endl;
  swap(&m,&n);
  cout<< m <<" " << n<<endl;
  return 0;
}
```

Summary

- Address-of Operator (&)
- Dereference operator (*)
- Declaring pointers
- Pointer initialization
- Pointer arithmetics

Pointer Vs Arrays

NEW YORK UNIVERSITY

ABU DHABI

Pointers and arrays are strongly related. In fact, pointers and arrays are interchangeable in many cases.

Pointer Vs Arrays

NEW YORK UNIVERSITY
ABU DHABI

```
#include <iostream>
using namespace std;

double calMean(int *arr, int size)
{
  int    i, sum = 0;
  double mean;

  for (i = 0; i < size; ++i)
  {
    sum += arr[i];
   }

  mean = double(sum) / size;

  return mean;
}

int main ()
{
   // an int array with 5 elements.
   int score[5] = {1000, 12, 111, 170, 150};
   double mean;
    // pass pointer to the array as an argument.
   mean = calMean( score, 5 ) ;
    // output the returned value
   cout << "Mean value is: " << mean << endl;

   return 0;
}
```

Pointer Vs Arrays

NEW YORK UNIVERSITY
ABU DHABI

However, pointers and arrays are not completely interchangeable.

```
#include <iostream>

using namespace std;
const int MAX = 3;

int main ()
{
   int  var[MAX] = {10, 100, 200};

   for (int i = 0; i < MAX; i++)
   {
      *var = i;     // This is a correct syntax
      var++;        // This is incorrect.
   }
   return 0;
}
```

Pointer Vs Arrays

- The sizeof operator
  a) sizeof(array) returns the amount of memory used by all elements in array
  b) sizeof(pointer) only returns the amount of memory used by the pointer variable itself

- The & operator
  a) &array is an alias for &array[0] and returns the address of the first element in array
  b) &pointer returns the address of pointer

- A string literal initialization of a character array
  a) char array[] = “abc” sets the first four elements in array to ‘a’, ‘b’, ‘c’, and ‘\0’
  b) char *pointer = “abc” sets pointer to the address of the “abc” string (which may be stored in read-only memory and thus unchangeable)

- Pointer variable can be assigned a value whereas array variable cannot be.

```
int  a[10];
int  *p;
p=a;  /*legal*/
a=p;  /*illegal*/
```

- Arithmetic on pointer variable is allowed.

```
p++;  /*Legal*/
a++;  /*illegal*/
```