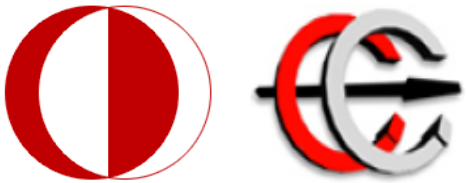


EE 441 Data Structures

Lecture 2: Arrays, Pointers, Input/Output

Memory

- A computer's memory is made up of bytes
- Each byte has an address, associated with it
- Different data types occupy different number of consecutive bytes in memory
- Examples

address	contents
1000	57
1004	31
1008	0
1012	1004
1016	

Type Name	Bytes	Range of Values
char	1	–128 to 127
unsigned char	1	0 to 255
short	2	–32,768 to 32,767
unsigned short	2	0 to 65,535
long	4	–2,147,483,648 to 2,147,483,647
unsigned long	4	0 to 4,294,967,295
enum	*	Same as int
float	4	3.4E +/- 38 (7 digits)
double	8	1.7E +/- 308 (15 digits)
long double	10	1.2E +/- 4932 (19 digits)

Memory: Variables Example

- Character type variable declaration

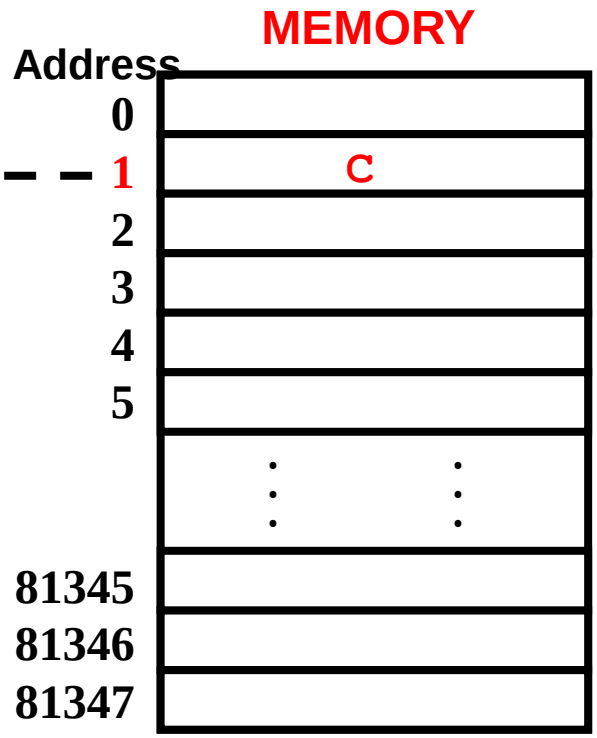
```
char mychar ='C';
```

  - Allocated memory for mychar is at Location 1 (this example)
  - Amount of memory allocated for mychar

```
cout << "Memory: " << sizeof(char) << endl;
```

```
Memory: 1
```

→ Location 1 contains the binary representation for character ‘C’

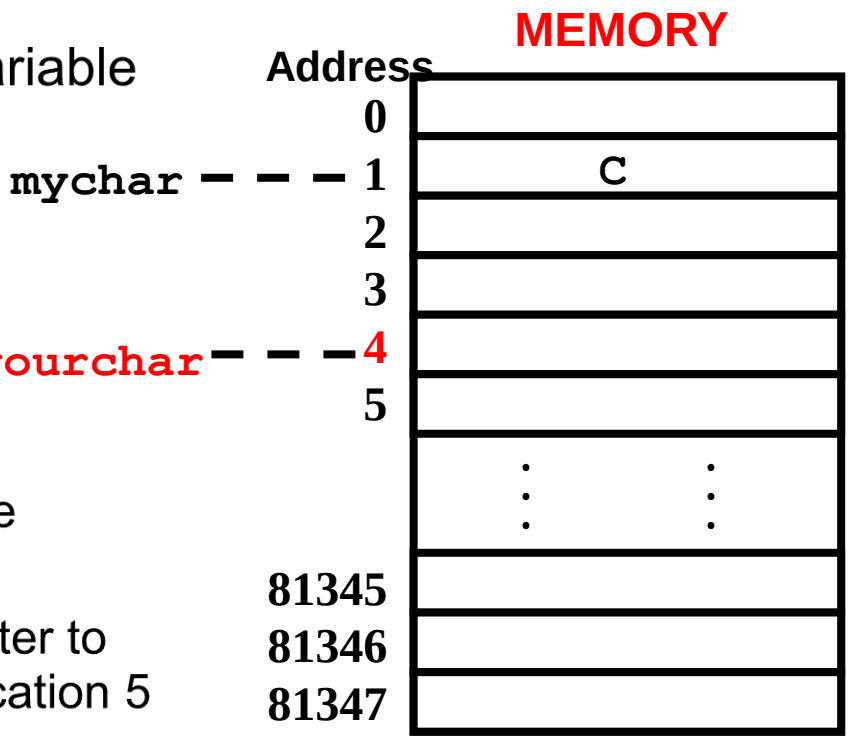


Memory: Variables Example

- Declare another character type variable

```
char yourchar;
```

- Allocated memory for `yourchar` is at Location 4
- Location 4 is uninitialized
- Important note
  - Programmer has no control on the selected location
  - It is not possible to tell the computer to store `mychar` or `yourchar` at location 5



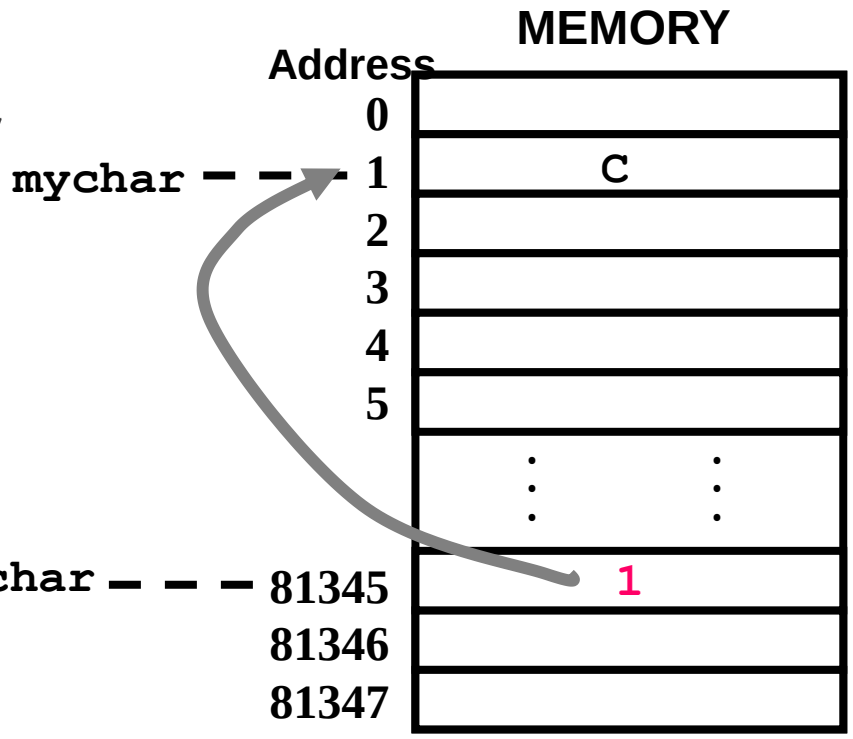
Pointers: Introduction

- A pointer is a variable that holds the address of another object
- Example: pointer to a character type variable declaration

```
char* pchar;
```

- Example: storage of the address of an existing variable of the same type in the pointer

```
pchar = &mychar;
```



Pointers: Type

- Type of the pointer variable changes according to the type of the pointed object

```
char* pchar;//pchar is a variable. Its type is pointer to character
int* intaddres;//intaddress's type is pointer to integer
pchar = intaddress;//will not work type mismatch!!
```

Pointers

- Typical uses of pointers are the creation of linked lists and the management of objects dynamically allocated during execution
- Example declarations

```
int* ip; // pointer declaration
int *ip; // the same as above
int *ip1, *ip2; // two pointers
int *ip1, ip2; // a pointer, an integer
```

Pointers: Assigning a value

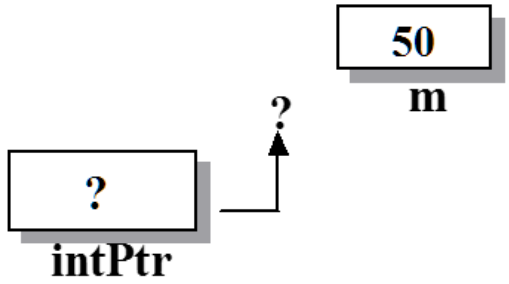
- Example

```
int m = 50;
int* intPtr;
(int m, *intPtr; //is valid too)
```

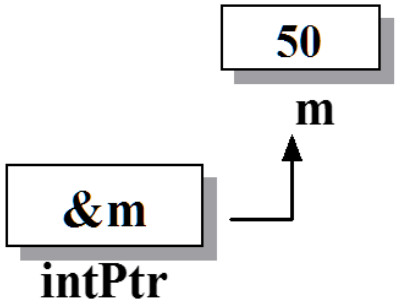
- "address-of" operator(&): Retrieves the memory address of a variable or object declared in the program
- `&m` is the address of the integer in memory

```
intPtr = &m;
```

→ sets `intPtr` to point at an actual data item



(a) After declaration



(b) After assignment

Pointers: Accessing data

- "de-reference" operator (*)

  → Allows access to the contents referenced by the pointer

```
intPtr = &m;
```

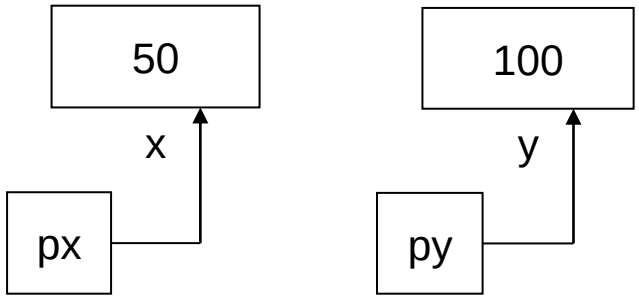
  → `*intPtr` is an alias form (alternative name) for `m`

- Equivalent operations

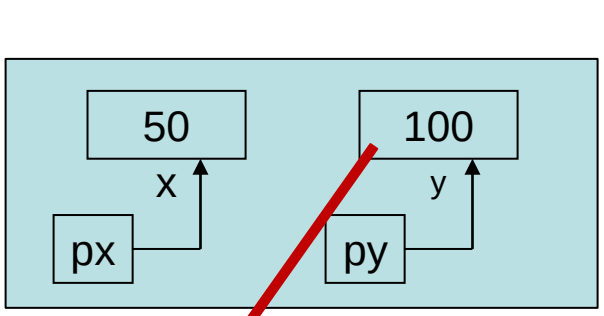
```
*intPtr = 60;
m = 60;
```

- Examples

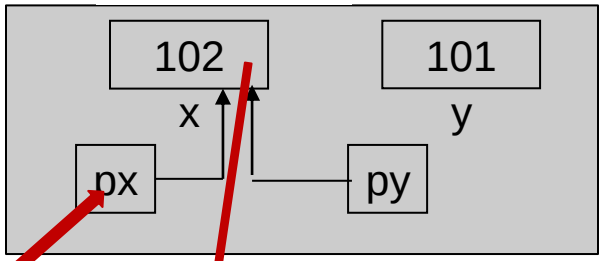
```
int x=50, y=100;
int *px = &x, int *py = &y;
```



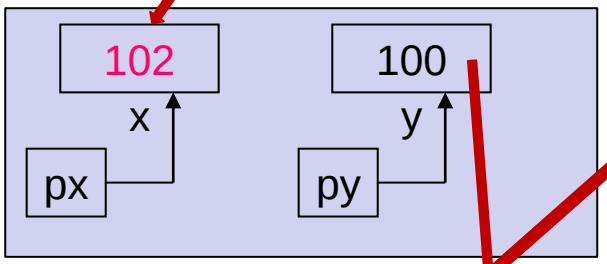
Pointers: Accessing data



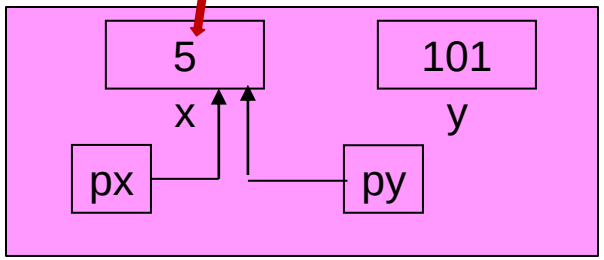

```
py = px;
```



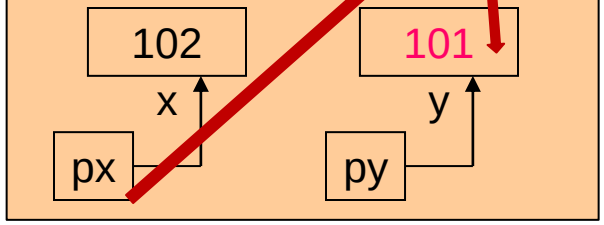

```
*px = *py + 2;
```




```
*py = 5;
```




```
*py = *py + 1;
```

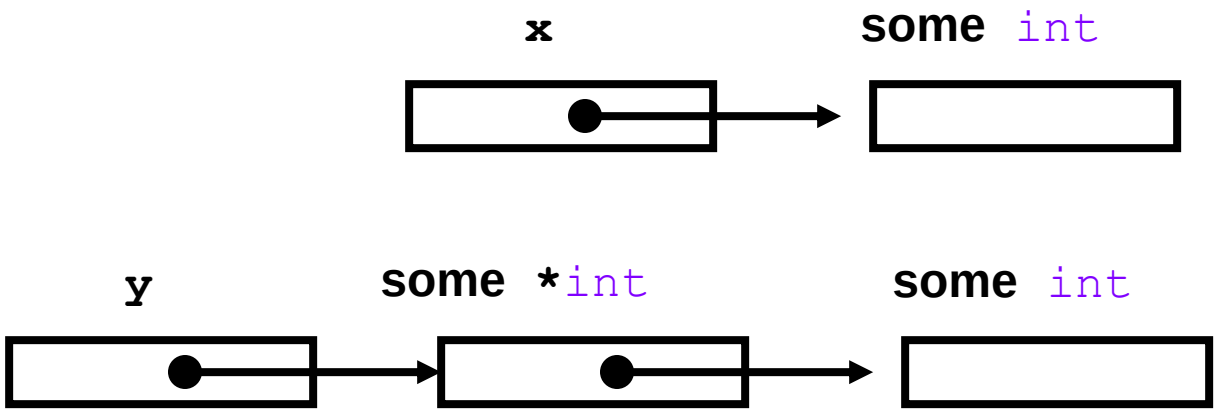



```
cout<<"x: "<<x<<" y: "<<y<<endl;
cout<<"px: "<<px<<" py: " <<py<<endl;
```

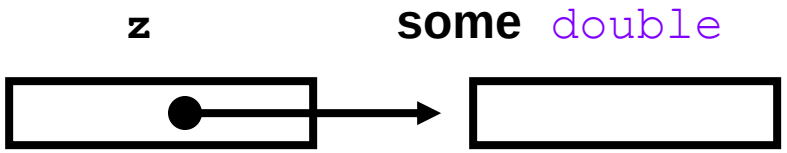
```
x: 5 y: 101
px: 0x61e3bc py: 0x61e3bc
```

Pointers: Accessing Data

```
int *x;
int **y;
```




```
double *z;
```



Pointers: Accessing Data: Example

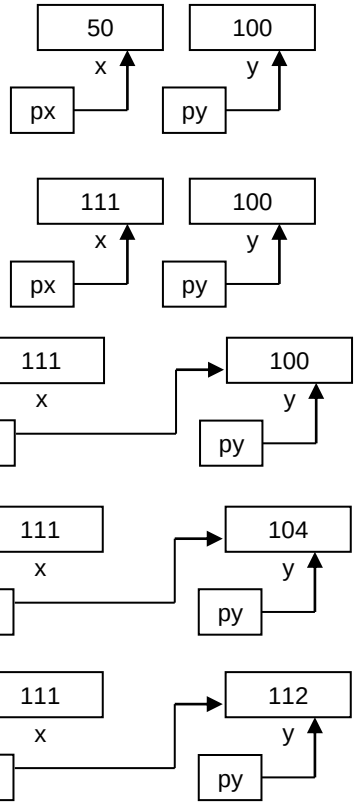
```
int i=1024;

//create a pointer that points to i
int *ip = i;
// error, type mismatch


int *ip = &i;
// ok. the operator & is referred as address-of operator
```

```
int *ip2 = ip;
// ok. now ip2 is another pointer that also addresses i

int *ip3 = &ip2;
// error, type mismatch



int **ip3 = &ip2;
// ok, it is a pointer to a pointer
```

Example RUN IT @home

```
#include <iostream>
using namespace std;
int main(){
    int *px, *py;
    int x, y;
    x=50;
    y=100;
    px=&x;
    py=&y;
    cout<<"x:"<<x<<" y:"<<y<<"\n";
    cout<<"px:"<<px<<" py:"<<py<<"\n";
    cout<<"*px:"<<*px<<" *py:"<<*py<<"\n";
    cout<<"\n";
    *px=y+11;
    cout<<"x:"<<x<<" y:"<<y<<"\n";
    cout<<"px:"<<px<<" py:"<<py<<"\n";
    cout<<"*px:"<<*px<<" *py:"<<*py<<"\n";
    cout<<"\n";
```

```
    px=py;
    cout<<"x:"<<x<<" y:"<<y<<"\n";
    cout<<"px:"<<px<<" py:"<<py<<"\n";
    cout<<"*px:"<<*px<<" *py:"<<*py<<"\n";
    cout<<"\n";


    *px=(*px)+4;
    *py=(*py)+8;
    cout<<"x:"<<x<<" y:"<<y<<"\n";
    cout<<"px:"<<px<<" py:"<<py<<"\n";
    cout<<"*px:"<<*px<<" *py:"<<*py<<"\n";
    cout<<"\n";
    return 0;
}
```



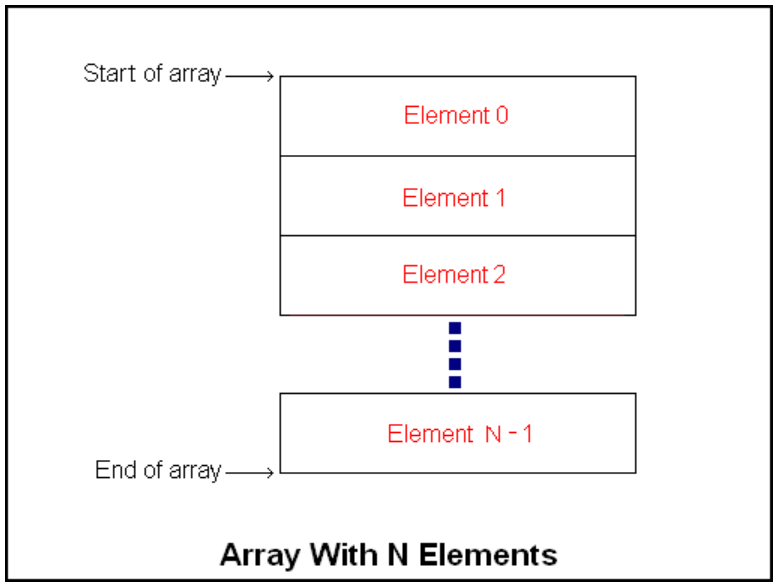

```
x:50 y:100
px:0x61fe0c py:0x61fe08
*px:50 *py:100

x:111 y:100
px:0x61fe0c py:0x61fe08
*px:111 *py:100
```

```
x:111 y:100
px:0x61fe08 py:0x61fe08
*px:100 *py:100

x:111 y:112
px:0x61fe08 py:0x61fe08
*px:112 *py:112
```

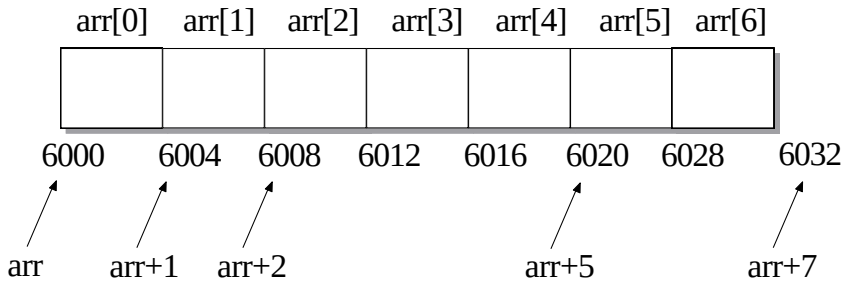
Code: Lecture2_pointers_run_at_home

Arrays

- An array is a consecutive group of memory locations (i.e., elements of the array)
- The contents of each element are of the same type
- Example: array int, double, char, long int, …
- We know the start position of the array corresponding to element 0
- We can refer to individual elements by giving the position number (index, subscript) of the element in the array



Pointers and Arrays



- Array name is the starting address of the memory block

```
int arr[7];
```

→ `arr` is the starting address

- At runtime, the computer
  - Allocates space for 7 integer objects
  - Assigns to `arr` the starting address of the memory block
  - Associates the pointer with the type and the size of the item it references
  - Knows that each data item is a 4-byte integer
- Pointer arithmetic
  - `arr+i` points at the i-th location of the array
  - Indexing starts with 0: `arr = arr+0`

Declaring an Array

- General Format

```
element_type array_name[number_of_elements];
```

  - `element_type`: can be any C++ data type
  - `array_name`: can be any valid variable name
  - `number_of_elements`: must be a constant expression known at compile time

- Declaration Examples

```
char  C[20]; // valid since fixed number
const int ArraySize = 50;
float  A[ArraySize]; // valid since constant
long  X[Arraysize+10];
int n;
std::cin >> n;
int arr[n];      // not valid according to standard C++
```

Initialization

- Arrays can be initialized with the declaration (like variables)

```
int grades[5] = {1,8,3,6,12};
double d[2] = {0.707, 0.707};
char s[] = { 'M', 'E', 'T', `U' };
int arr[5];
memset(arr, 0, sizeof(arr));   // set all bytes to 0 → {0,0,0,0,0}
```

You don't need to specify a size when initializing, the compiler will count for you

- Examples

```
long factorial(const int n)
{
    long f = 1;
    for (int i=1; i<=n; ++i)
        f *= i;
    return f;
}
```

Code: Lecture2_factorial

```
int main (){
        int facs[10];
        for (int i = 0; i < 10; i++)
                facs[i] = factorial(i);
        for (int i = 0; i < 10; i++)
        cout << "factorial(" << i << ") is "
                << facs[i] << endl;
        return 0;
}
```

Initialization

- Arrays can be initialized with the declaration (like variables)

```
int grades[5] = {1,8,3,6,12};
double d[2] = {0.707, 0.707};
char s[] = { 'M', 'E', 'T', `U' };
int arr[5];
memset(arr, 0, sizeof(arr));   // set all bytes to 0 → {0,0,0,0,0}
```

You don't need to specify a size when initializing, the compiler will count for you

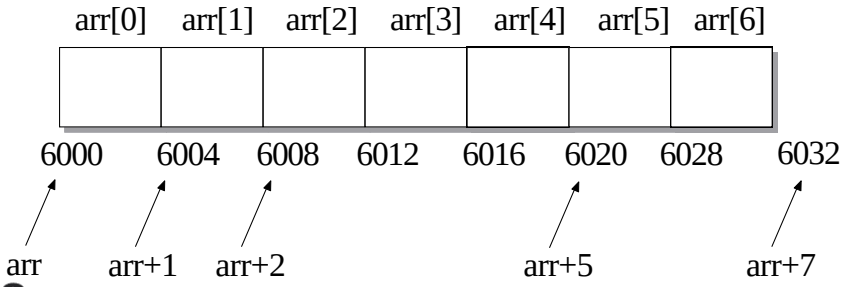
- Examples

```
long factorial(const int n)
{
    long f = 1;
    for (int i=1; i<=n; ++i)
        f *= i;
    return f;
}
```

Code: Lecture2_factorial

```
factorial(0) is 1
factorial(1) is 1
factorial(2) is 2
factorial(3) is 6
factorial(4) is 24
factorial(5) is 120
factorial(6) is 720
factorial(7) is 5040
factorial(8) is 40320
factorial(9) is 362880
```

Pointer Arithmetic



- Pointers behave like array indices
  - If `p` points to `arr[0]`, then `p+1` points to `arr[1]`
  - The arithmetic is scaled by the size of the type `sizeof(arr)`
- Valid operations on pointers
  - Increment/decrement (`p++, p--`)
    → move to next/previous element
  - Integer addition/subtraction (`p+n, p-n`)
    → move n elements
  - Difference between two pointers (`p-q`)
    → number of elements between them (if pointing into same array)
  - Comparisons (`<, >, ==`)
    → relative positions inside same array

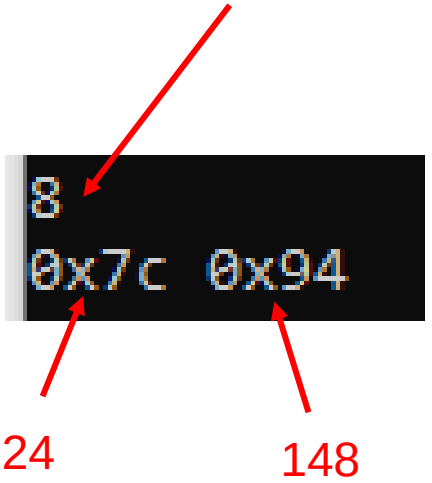
```
int arr[5] = {10, 20, 30, 40, 50};
int* p = arr;  // points to arr[0]
p++;           // move to arr[1]
p--;           // back to arr[0]
int d = *(p + 2); // d = 30 (arr[2])
int e = *(p + 4); // e = 50 (arr[4])
int* q = &arr[4];
int diff = q - p;    // diff = 4
bool cmp1 = (p < q);   // true
bool cmp2 = (q > p);   // true
bool cmp3 = (p == arr); // true
```

Pointer Arithmetic: Discussion

- Dereferencing with arithmetic
  - `*(p + n)` is the same as `arr[n]`
  - `p[i]` is shorthand for `*(p + i)`

```
double p;
  // Get size of double in Bytes
  cout << sizeof(p) << endl;
  double *pp;
  // Increment pointer content by 3
  double *q = pp + 3;
  // Print pointer contents
  cout << pp << " " << q;
  return 0;
```

8 Bytes is the size of double



→ difference is 3*8 = 24 Bytes

- Remarks
  - Only meaningful within the same array object
  - Moving outside array bounds = undefined behavior
  - Arithmetic is based on element size, not bytes

C++ Operations on Arrays

- Assignment
  - Direct index assignment using brackets

```
int arr[5];
arr[0] = 100;
arr[4] = 200;
```

  - Pointer-based assignment using "*" and pointer arithmetic

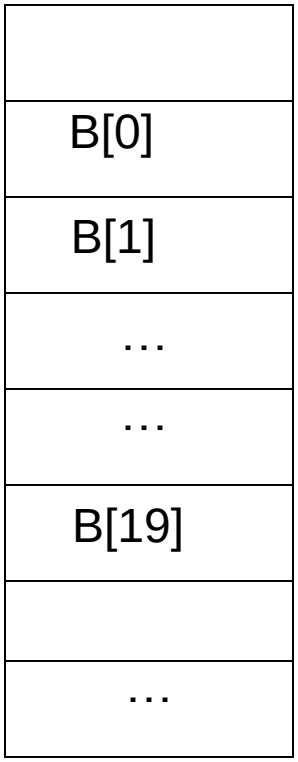
```
int arr[5];
int* p = arr; // arr start  pointer
*p = 100;   // assign via pointer arithmetic
*(p + 4) = 200;
```

- Assignment Examples

```
A[i] = z;
t = X[i];
X[i] = X[i+1] = t;
/*this is equivalent to x[i+1]=t; x[i]=x[i+1];i.e., right to left*/
```

C++ Operations on Arrays

- Attention: Most C++ compilers don't check array index range !!




```
int V = 20;
int B[20]; /*  index range is 0-19  */
B[V] = 0; /* index is out of range, but most
                    C++ compilers don't check this */
```

Try to assign B[20], which is not in the address range

Two Dimensional Arrays

- Example

```
int T[3][4] = {{20,5,-3,0},{-85,35,-1,2},
                {15,3,-2,-6}};
for(unsigned int i = 0; i < 3; i++){
    for(unsigned int j = 0; j < 4; j++)
        cout << T[i][j] << "\t";
    cout << endl;
 }
```

```
20      5       -3      0
-85     35      -1      2
15      3       -2      -6
```

- Assignment

```
T[2] [3] = 12;
T[2][0] = -5
```

T

20	5	-3	0	T[0]
-85	35	-1	2	T[1]
-5	3	-2	12	T[2]

Pointers and Arrays: Example

```
#include <iostream>
using namespace std;
int main(){
    int arr[7];
    int i;
    // Insert values in arrray
    for(i=0; i<7; i++)
        arr[i] = i;
    // Print values in array
    for(i=0;i<7;i++)
        cout << arr[i] << " ";
    cout << endl;
    // Print addresses
    for(i=0;i<7;i++)
        cout<< &arr[i] << endl;
     // Access values by dereferencing
     for(i=0;i<7;i++)
         cout << *(arr+i) << " ";
    cout << endl;
// Pointer for first array element
    int *p=arr;
    // Print values using pointer increment
    for(i=0;i<7;i++){
        cout << *p << " ";
        p++;
    }
    cout << endl;

    return 0;
}
```

```
0 1 2 3 4 5 6
0x61fdf0
0x61fdf4
0x61fdf8
0x61fdfc
0x61fe00
0x61fe04
0x61fe08
0 1 2 3 4 5 6
0 1 2 3 4 5 6
```

Code: Lecture2_pointers_and_arrays

Input/Output in C++

- Output (cout)
  - From <iostream>
  - Uses insertion operator <<
- Input (cin)
  - From <iostream>
  - Uses extraction operator >>
- Chaining
  - Multiple inputs/outputs in one statement

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

int main() {
   int x = 10;
   cout << "Value of x = " << x << endl;

   int y;
   cout << "Enter a number: ";
   cin >> y;

   int a, b;
   // read two numbers
   cin >> a >> b;
   // output both
   cout << “a: " << a << “ b: " << b << endl;
   return 0;
}
```

Namespaces in C++

- Namespace definition
  - Groups related identifiers (variables, functions, classes) under a name
  - Helps avoid name conflicts in large projects or libraries
- Declaring a namespace

```
namespace Math {
    int add(int a, int b) { return a + b; }
    int mul(int a, int b) { return a * b; }
}
```

- Accessing namespace members

```
// with scope operator
int x = Math::add(2, 3);   // 5
// Using directive
using namespace Math;
int y = mul(4, 5);   // 20
```

- Standard namespace (std)
  - Most of the C++ standard library is in std.

```
// with scope operator
std::cout << "Hello";
using namespace std;
// after using directive
cout << "Hello";
```

Char Strings in C/C++: Definition

- C-Style Character Strings
  - A character array ending with the null terminator `'\0'`
  - Stored in contiguous memory
- Declaration/Initialization

```
char str1[] = "Hello";          // compiler adds '\0'
char str2[6] = {'H','e','l','l','o','\0'};
```

- Input/Output using cout/cin

```
cout << str1 << endl; // prints Hello
cin >> str1; // reads word (stops at space)
```

- Important Remarks
  - Must always leave room for `'\0'`
  - Example: `"Hello"` needs 6 chars (5 letters + null)

Char Strings in C/C++: Common Functions

- Length
  - Get the number of characters
  - Excludes '0'
- Copy
  - Copy string into another string
  - Target must have enough chars
- Concatenate
  - Concatenate strings
  - Target must have enough chars
- Compare
  - Compare two strings
- Convert
  - Get integer value of char string

```
#include <iostream>
#include <cstring>
#include <cstdlib>   // for atoi
using namespace std;
int main(){
   char str1[] = "Hello";
   // length excluding '\0'
   cout << " length: " << strlen(str1) <<  endl;
   char dest[20];
   strcpy(dest, str1); // copies "Hello" into dest
   strcat(dest, " World"); // dest = "Hello World"
   // equal strings
   if (strcmp(str1, "Hello") == 0)
      cout << " Strings are equal" << endl;
   // convert char numStr[] = "456“ to int
   char numStr[] = "456";     // C-string
   int value = atoi(numStr);
   cout << " number: " << value <<  endl;;
   return 0;
}
```

Code: Lecture2_char_strings

Using Arguments in main

- Forms of main
  - `int main(); // simplest form`
  - `int main(int argc, char* argv[]); // with arguments`
- Parameters
  - Argument count: argc
    → number of command-line arguments
  - Argument vector: argv
    → array of C-strings
- Command Line

```
$ ./bin/myapp.exe hello 123
Number of arguments: 3
arg[0] = ./bin/myapp
arg[1] = hello
arg[2] = 123
```

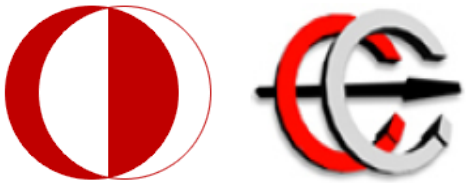
```
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
    cout << "Number of arguments: "
         << argc << endl;

    for (int i = 0; i < argc; i++) {
        cout << "arg[" << i << "] = "
             << argv[i] << endl;
    }
    return 0;
}
```

Using Arguments in main: Example

```
#include <iostream>
#include <cstdlib>   // for atoi
using namespace std;

int main(int argc, char* argv[]) {
    if (argc < 3) {
        cout << "Usage: " << argv[0]
             << " num1 num2 [num3 ...]" << endl;
        return 1; // exit with error
    }
    int sum = 0;
    for (int i = 1; i < argc; i++) {
        sum += atoi(argv[i]);   // convert string → int
    }
    cout << "Sum = " << sum << endl;
    return 0;
}
```

```
$ ./bin/myapp.exe 10 20 30
Sum = 60
```

Code: Lecture2_arguments_main



EE 441 Data Structures

Lecture 2: Arrays, Pointers, Input/Output