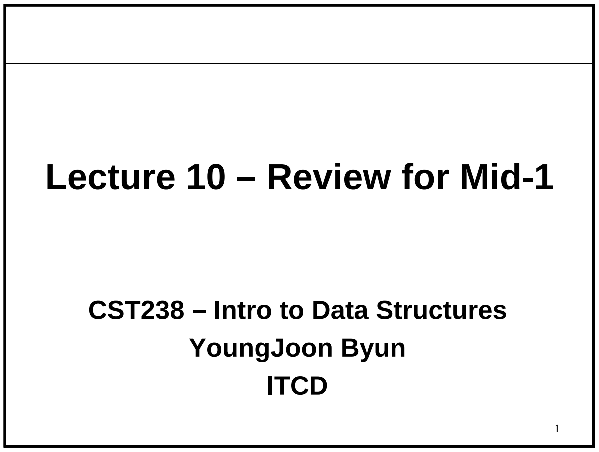
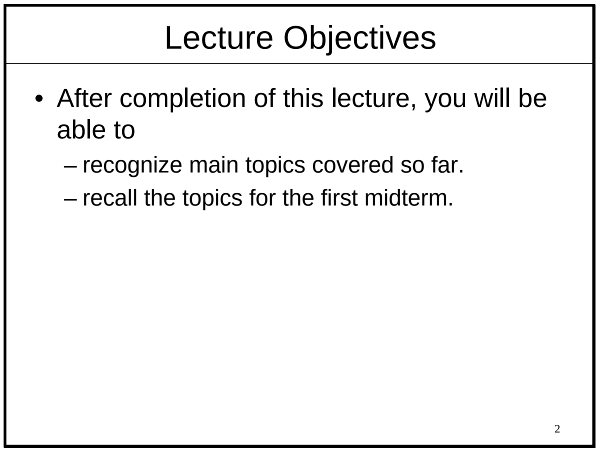


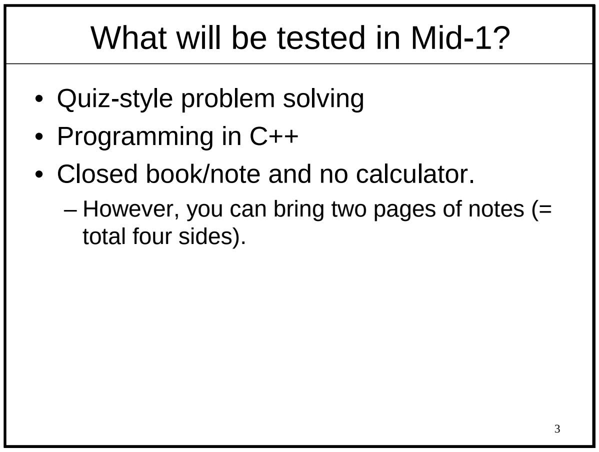
Lecture 10 – Review for Mid-1

CST238 – Intro to Data Structures
YoungJoon Byun
ITCD



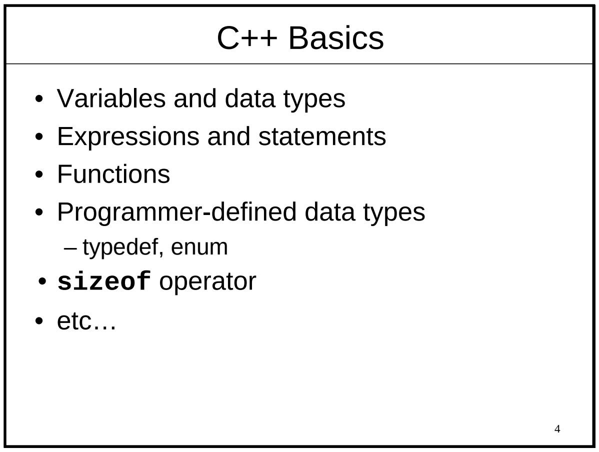
Lecture Objectives

- After completion of this lecture, you will be able to
  - – recognize main topics covered so far.
  - – recall the topics for the first midterm.



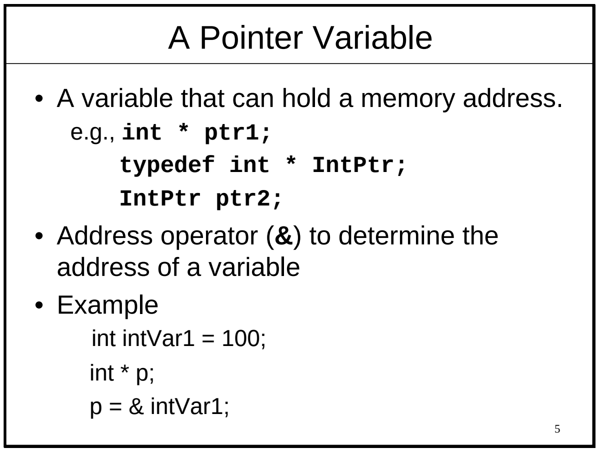
What will be tested in Mid-1?

- Quiz-style problem solving
- Programming in C++
- Closed book/note and no calculator.
  - – However, you can bring two pages of notes (= total four sides).



C++ Basics

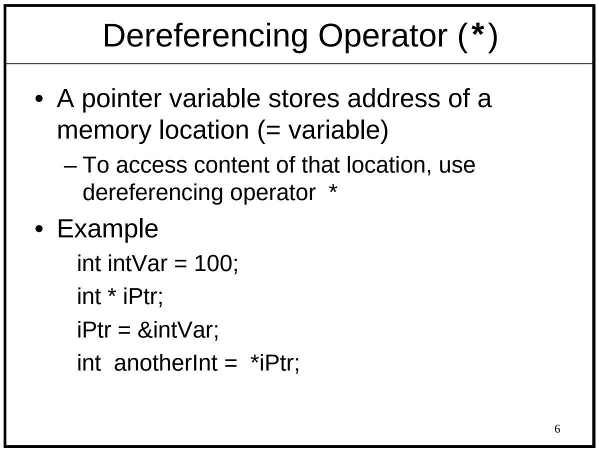
- Variables and data types
- Expressions and statements
- Functions
- Programmer-defined data types
  - – typedef, enum
- **`sizeof`** operator
- etc…



A Pointer Variable

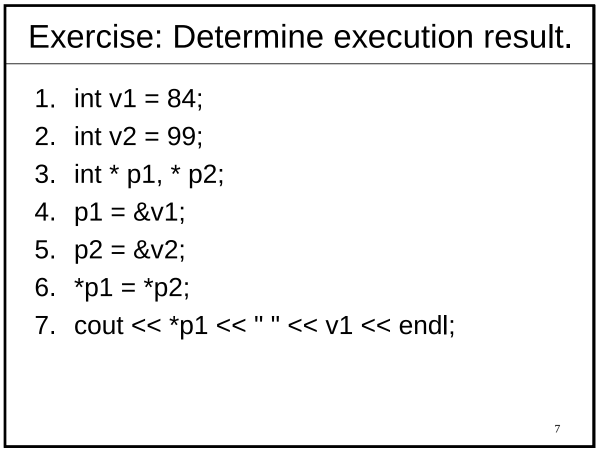
- A variable that can hold a memory address.

  e.g.,
```
int * ptr1;
typedef int * IntPtr;
IntPtr ptr2;
```
- Address operator (**&**) to determine the address of a variable
- Example
```
 int intVar1 = 100;
int * p;
p = & intVar1;
```



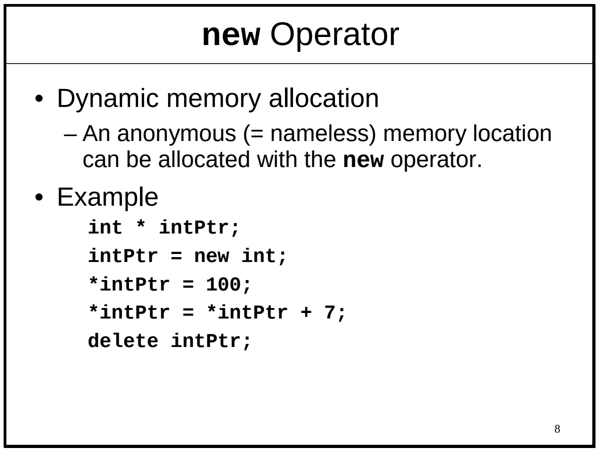
Dereferencing Operator (*****)

- A pointer variable stores address of a memory location (= variable)
  - – To access content of that location, use dereferencing operator *
- Example
```
int intVar = 100;
int * iPtr;
iPtr = &intVar;
int  anotherInt =  *iPtr;
```



Exercise: Determine execution result.

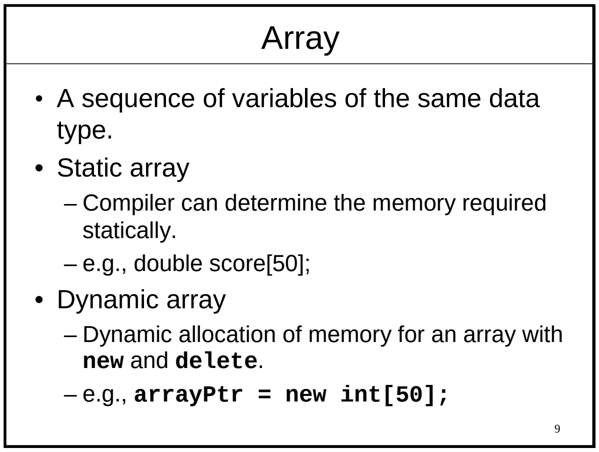
```
int v1 = 84;
int v2 = 99;
int * p1, * p2;
p1 = &v1;
p2 = &v2;
*p1 = *p2;
cout << *p1 << " " << v1 << endl;
```



new Operator

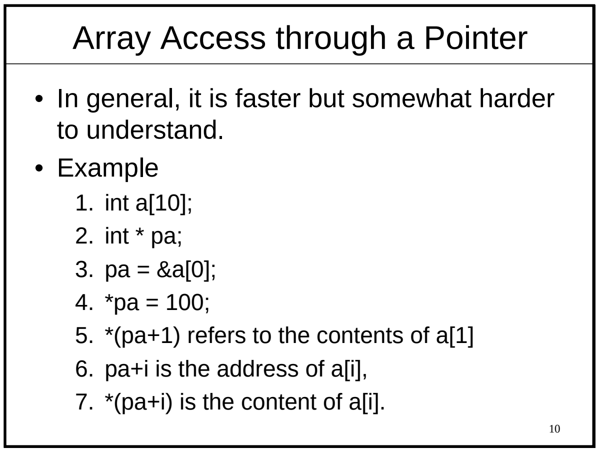
- Dynamic memory allocation
  - An anonymous (= nameless) memory location can be allocated with the **new** operator.
- Example

```
int * intPtr;
intPtr = new int;
*intPtr = 100;
*intPtr = *intPtr + 7;
delete intPtr;
```



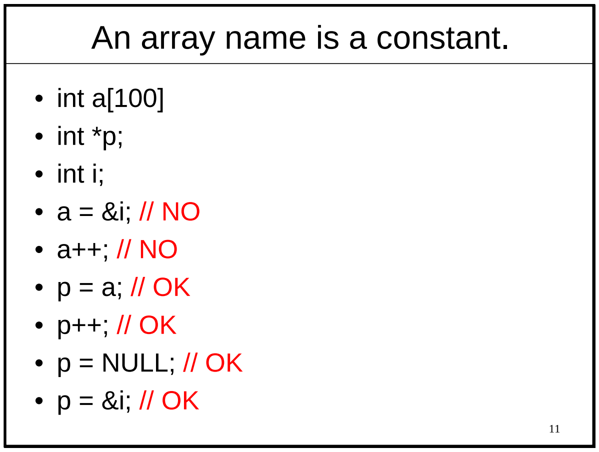
Array

- A sequence of variables of the same data type.
- Static array
  - Compiler can determine the memory required statically.
  - e.g., double score[50];
- Dynamic array
  - Dynamic allocation of memory for an array with **new** and **delete**.
  - e.g., **arrayPtr = new int[50];**



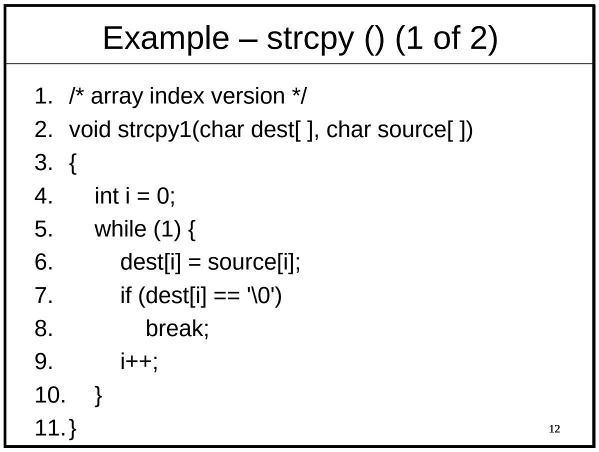
Array Access through a Pointer

- In general, it is faster but somewhat harder to understand.
- Example
  1. int a[10];
  2. int * pa;
  3. pa = &a[0];
  4. *pa = 100;
  5. *(pa+1) refers to the contents of a[1]
  6. pa+i is the address of a[i],
  7. *(pa+i) is the content of a[i].



An array name is a constant.

- int a[100]
- int *p;
- int i;
- a = &i; // NO
- a++; // NO
- p = a; // OK
- p++; // OK
- p = NULL; // OK
- p = &i; // OK



Example – strcpy () (1 of 2)

```
/* array index version */
void strcpy1(char dest[ ], char source[ ])
{
    int i = 0;
    while (1) {
        dest[i] = source[i];
        if (dest[i] == '\0')
            break;
        i++;
    }
}
```

Example – strcpy () (2 of 2)

```
/* pointer version */
void strcpy2(char * dest, char * source)
{
    while ((*dest = *source) != '\0') {
        dest++;
        source++;
    }
}
```

An Array of Pointers

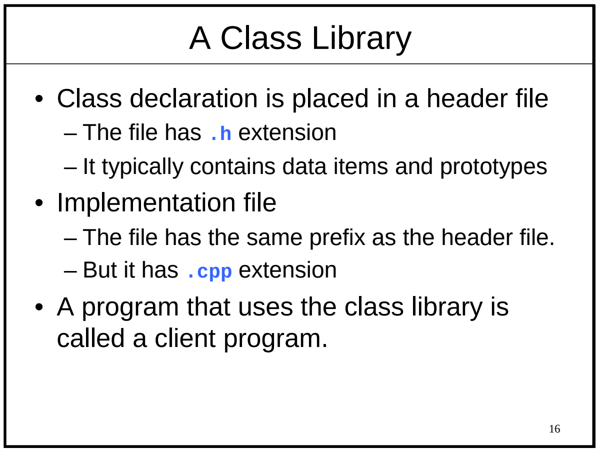
```
#include <iostream>
using namespace std;

int main() {
    int i, j;
    double * arrayPtr[10];

    for (i = 0; i < 10; i++) {
        arrayPtr[i] = new double [5];
        for (j = 0; j < 5; j++) {
            *(arrayPtr[i]+j) = j;
        }
    }
}
```

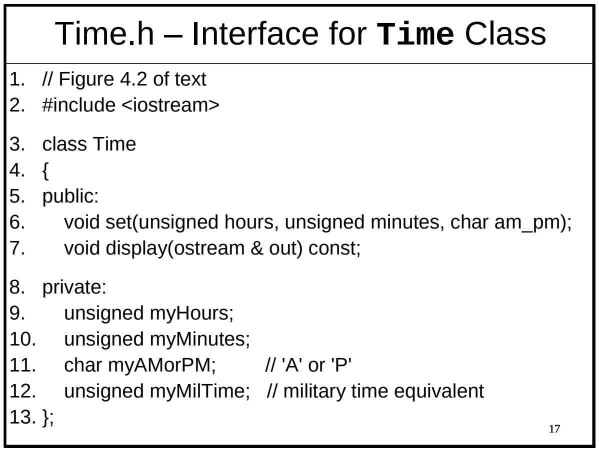
C++ Class

- A C++ class has
  - data members
  - functions (or methods)
- A class is a heart of object oriented programming.



A Class Library

- Class declaration is placed in a header file
  - The file has `.h` extension
  - It typically contains data items and prototypes
- Implementation file
  - The file has the same prefix as the header file.
  - But it has `.cpp` extension
- A program that uses the class library is called a client program.

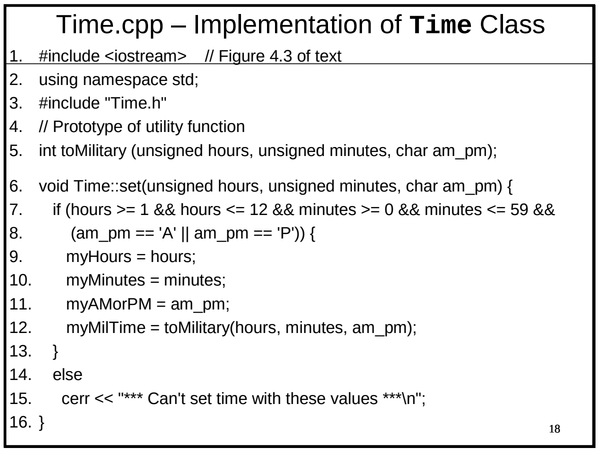


Time.h – Interface for **`Time`** Class

```
// Figure 4.2 of text
#include <iostream>

class Time
{
public:
   void set(unsigned hours, unsigned minutes, char am_pm);
   void display(ostream & out) const;

private:
   unsigned myHours;
   unsigned myMinutes;
   char myAMorPM;         // 'A' or 'P'
   unsigned myMilTime;   // military time equivalent
};
```



Time.cpp – Implementation of **`Time`** Class

```
#include <iostream>    // Figure 4.3 of text
using namespace std;
#include "Time.h"
// Prototype of utility function
int toMilitary (unsigned hours, unsigned minutes, char am_pm);

void Time::set(unsigned hours, unsigned minutes, char am_pm) {
   if (hours >= 1 && hours <= 12 && minutes >= 0 && minutes <= 59 &&
        (am_pm == 'A' || am_pm == 'P')) {
      myHours = hours;
      myMinutes = minutes;
      myAMorPM = am_pm;
      myMilTime = toMilitary(hours, minutes, am_pm);
   }
   else
     cerr << "*** Can't set time with these values ***\n";
}
```

driver.cpp – Test Driver for `Time` Class

```
// Figure 4.4
#include <iostream>
using namespace std;
#include "Time.h"

int main() {
   Time mealTime;

   mealTime.set(5, 30, 'P');

   cout << "We'll be eating at ";
   mealTime.display(cout);
   cout << endl;

   cout << "\nNow trying to set time with illegal hours \n";
   mealTime.set(13, 0, 'A');
}
```

Constructors

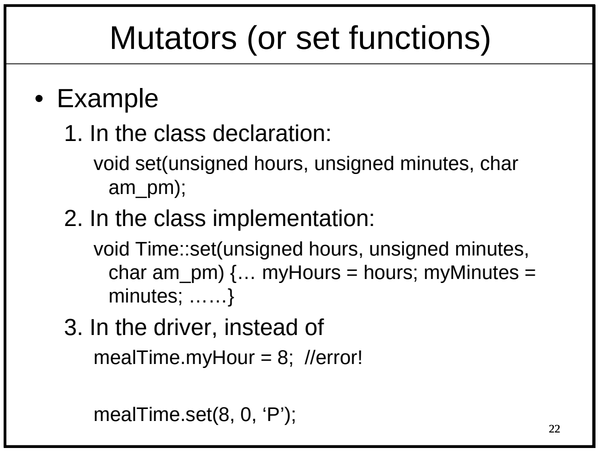
```
Time::Time()
  : myHours(12), myMinutes(0), myAMorPM('A'),
    myMilTime(0)
{
}

Time::Time(unsigned initHours, unsigned initMinutes, char initAMPM)
{
   // Check class invariant
   if (initHours >= 1 && initHours <= 12 &&
       initMinutes >= 0 && initMinutes <= 59 &&
       (initAMPM == 'A' || initAMPM == 'P')) {
         myHours = initHours;
         myMinutes = initMinutes;
         myAMorPM = initAMPM;
         myMilTime = toMilitary(initHours, initMinutes, initAMPM);
   }
   else
      cerr << "*** Invalid initial values ***\n";
}
```

Accessors (or get functions)

- Example: getHours()
  1. In the class declaration:
     unsigned getHours() const;
  2. In the class implementation:
     Time::getHours() const { return myHours; }
  3. In the driver, instead of
     cout << mealTime.myHour; // error! Why?

     cout << mealTime. getHours();



Mutators (or set functions)

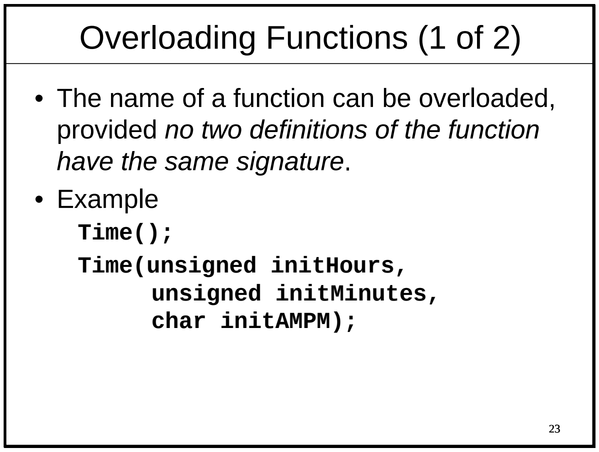
- Example
  1. In the class declaration:

     void set(unsigned hours, unsigned minutes, char am_pm);
  2. In the class implementation:

     void Time::set(unsigned hours, unsigned minutes, char am_pm) {… myHours = hours; myMinutes = minutes; ……}
  3. In the driver, instead of

     mealTime.myHour = 8; //error!

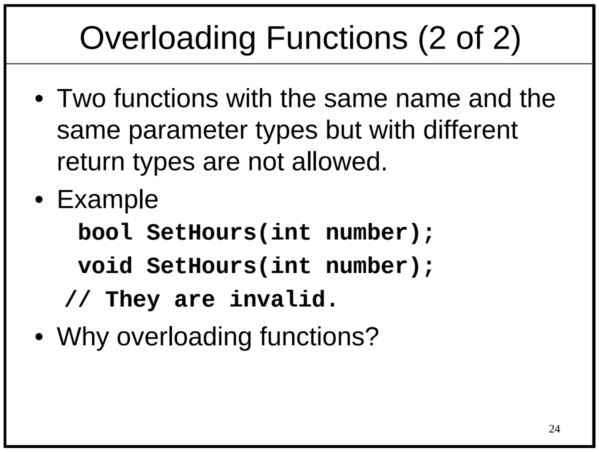
     mealTime.set(8, 0, ‘P’);



Overloading Functions (1 of 2)

- The name of a function can be overloaded, provided *no two definitions of the function have the same signature*.
- Example

```
Time();
Time(unsigned initHours,
     unsigned initMinutes,
     char initAMPM);
```



Overloading Functions (2 of 2)

- Two functions with the same name and the same parameter types but with different return types are not allowed.
- Example

```
 bool SetHours(int number);
 void SetHours(int number);
// They are invalid.
```

- Why overloading functions?