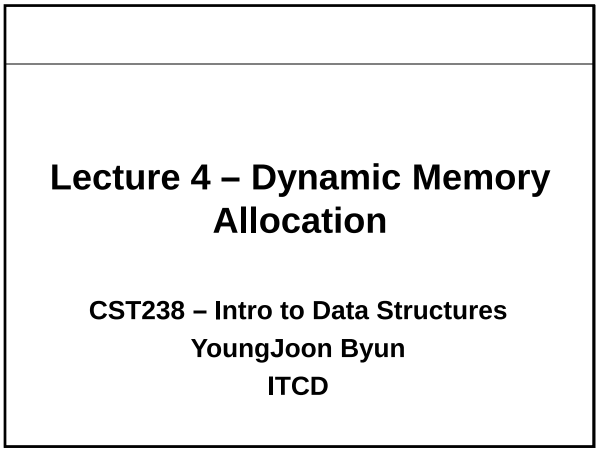
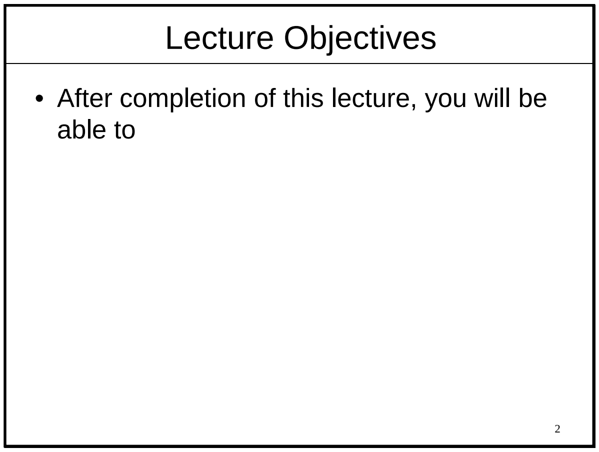


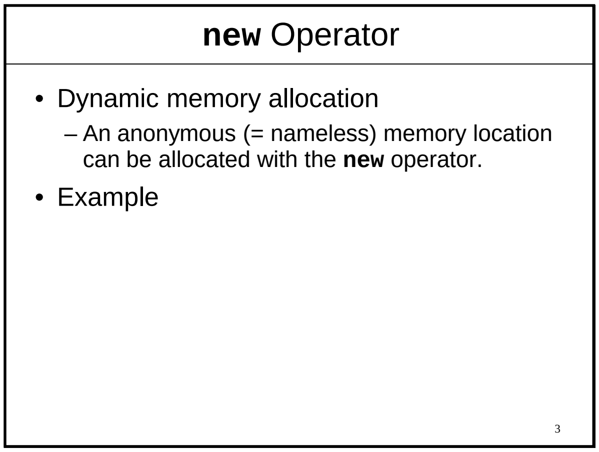
Lecture 4 – Dynamic Memory Allocation

CST238 – Intro to Data Structures
YoungJoon Byun
ITCD



Lecture Objectives

- After completion of this lecture, you will be able to



new Operator

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

Dynamic Memory Allocation

- A programmer can request a memory location during execution time.

```
type * p;
p = new type;
*p = any value of type;
//or
type * p = new type;
*p = any value of type;
```

Sample Program

```
#include <iostream>
using namespace std;
int main( ) {
    int *p1, *p2;

    p1 = new int;
    *p1 = 42;
    p2 = p1;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;

    *p2 = 53;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;

    p1 = new int;
    *p1 = 88;
    cout << "*p1 == " << *p1 << endl;
    cout << "*p2 == " << *p2 << endl;
}
```

Exercise: Determine the execution result of the sample program

Memory Management of Dynamic Variables

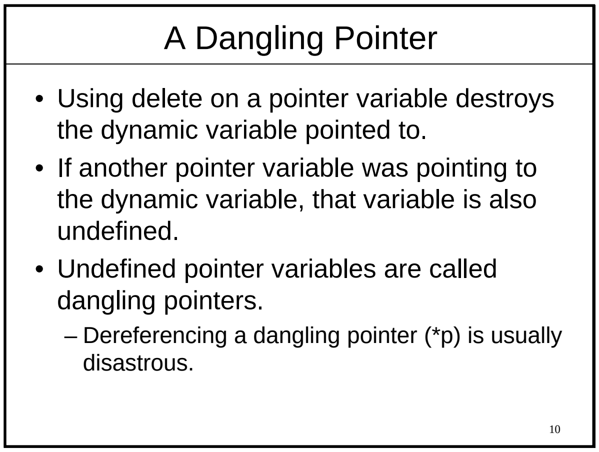
- An area of memory called the **free store (or heap memory)** is reserved for dynamic variables
  - New dynamic variables use memory in the free store.
  - If all of the free store is used, calls to **new** will fail.
- Unneeded memory can be recycled
  - When a variable is no longer needed, it can be released and the memory is returned to the free store.

`delete` Operator (1 of 2)

- When a dynamic variable is no longer needed, release it to return the memory to the free store.
- Example

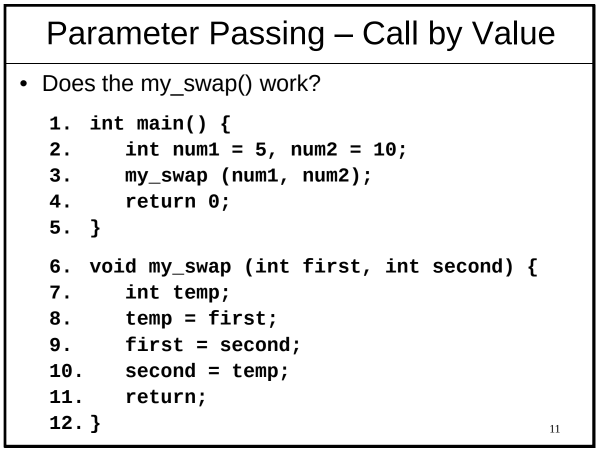
```
delete p;
// The memory used by the variable that p
// pointed to is back in the free store and
// the value of p is now undefined.
```

`delete` Operator (2 of 2)



A Dangling Pointer

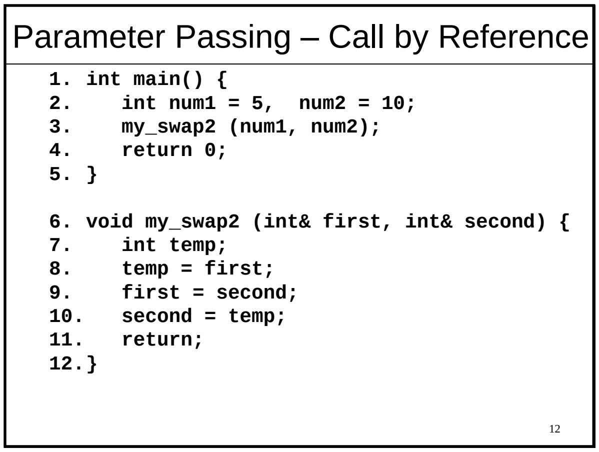
- Using delete on a pointer variable destroys the dynamic variable pointed to.
- If another pointer variable was pointing to the dynamic variable, that variable is also undefined.
- Undefined pointer variables are called dangling pointers.
  - Dereferencing a dangling pointer (*p) is usually disastrous.



Parameter Passing – Call by Value

- Does the my_swap() work?

```
int main() {
   int num1 = 5, num2 = 10;
   my_swap (num1, num2);
   return 0;
}
void my_swap (int first, int second) {
   int temp;
   temp = first;
   first = second;
   second = temp;
   return;
}
```



Parameter Passing – Call by Reference

```
int main() {
   int num1 = 5,  num2 = 10;
   my_swap2 (num1, num2);
   return 0;
}

void my_swap2 (int& first, int& second) {
   int temp;
   temp = first;
   first = second;
   second = temp;
   return;
}
```

What do you think to pass a pointer as an argument?

```
int main() {
   int num1 = 5,  num2 = 10;
   int * p1 = &num1;
   int * p2 = &num2;
   my_swap3 (p1, p2);
   cout << num1 << “ “ << num2 << endl;
   return 0;
}
void my_swap3 (int * first, int * second) {
   int temp;
   temp = *first;
   *first = *second;
   *second = temp;
   return;
}
```

Pointer Arguments

- Pointers can be passed as arguments to functions

Define a Pointer Type with **`typedef`**

- **`typedef int * IntPtr;`**
  - A type, called IntPtr, is the type for a pointer variable that can contain an address to an integer variable.
- Example

  IntPtr p;

Structure of C++

- A collection of multiple values with possibly different types.
- Example

```
struct cst338_score {
    string name;
    int id;
    double average;
    char grade;
};    // Don’t forget semicolon
```

- Reference: Chapter 3.5 of our textbook.

16

Usage of Structures

- Example

```
cst338_score  tom, chris, eric;
cst338_score  joe = {“Joe”, 1234, 88.5, ‘B’};
cst338_score  tyler;
tyler = {“Tyler”, 2345, 98.5, ‘A’};

joe.average = 90.7;
joe.grade = ‘A’;
tyler = joe;
```

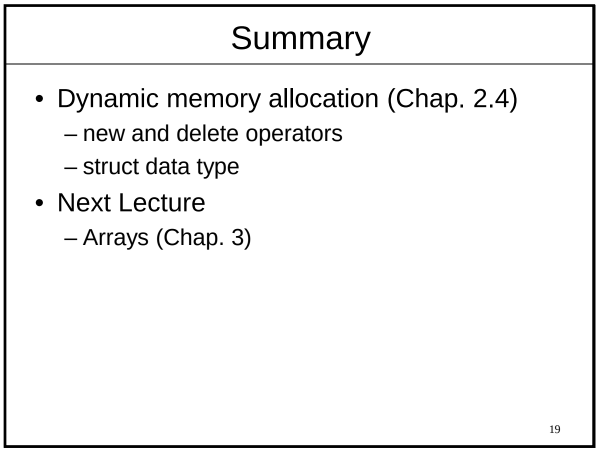
17

A Pointer to Structure Variable

- Declare a pointer variable, ptrStr, to point a variable, strVar, with cst338_score type.

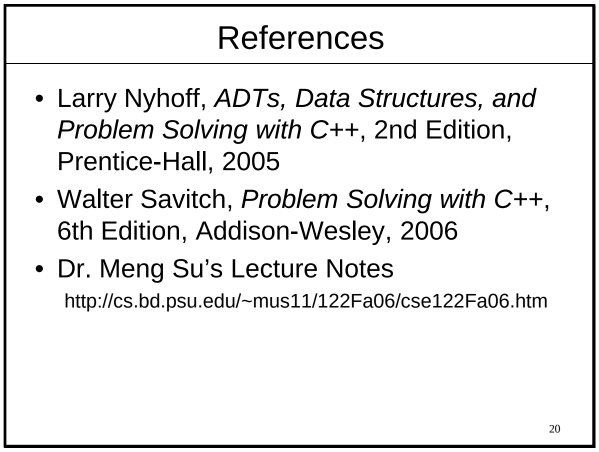
```
cst338_score strVar;
```

18



Summary

- Dynamic memory allocation (Chap. 2.4)
  - new and delete operators
  - struct data type
- Next Lecture
  - Arrays (Chap. 3)



References

- Larry Nyhoff, *ADTs, Data Structures, and Problem Solving with C++*, 2nd Edition, Prentice-Hall, 2005
- Walter Savitch, *Problem Solving with C++*, 6th Edition, Addison-Wesley, 2006
- Dr. Meng Su's Lecture Notes

  http://cs.bd.psu.edu/~mus11/122Fa06/cse122Fa06.htm