Week 7

CS 150 – C++ Programming I
In-Person Lecture

Partially-Filled Array Review

- Building-blocks used to implement classes like *vector*
  - Max-allocated array and two variables: *size*, *capacity*
  - Append, read, (or *push_back*) element *e*

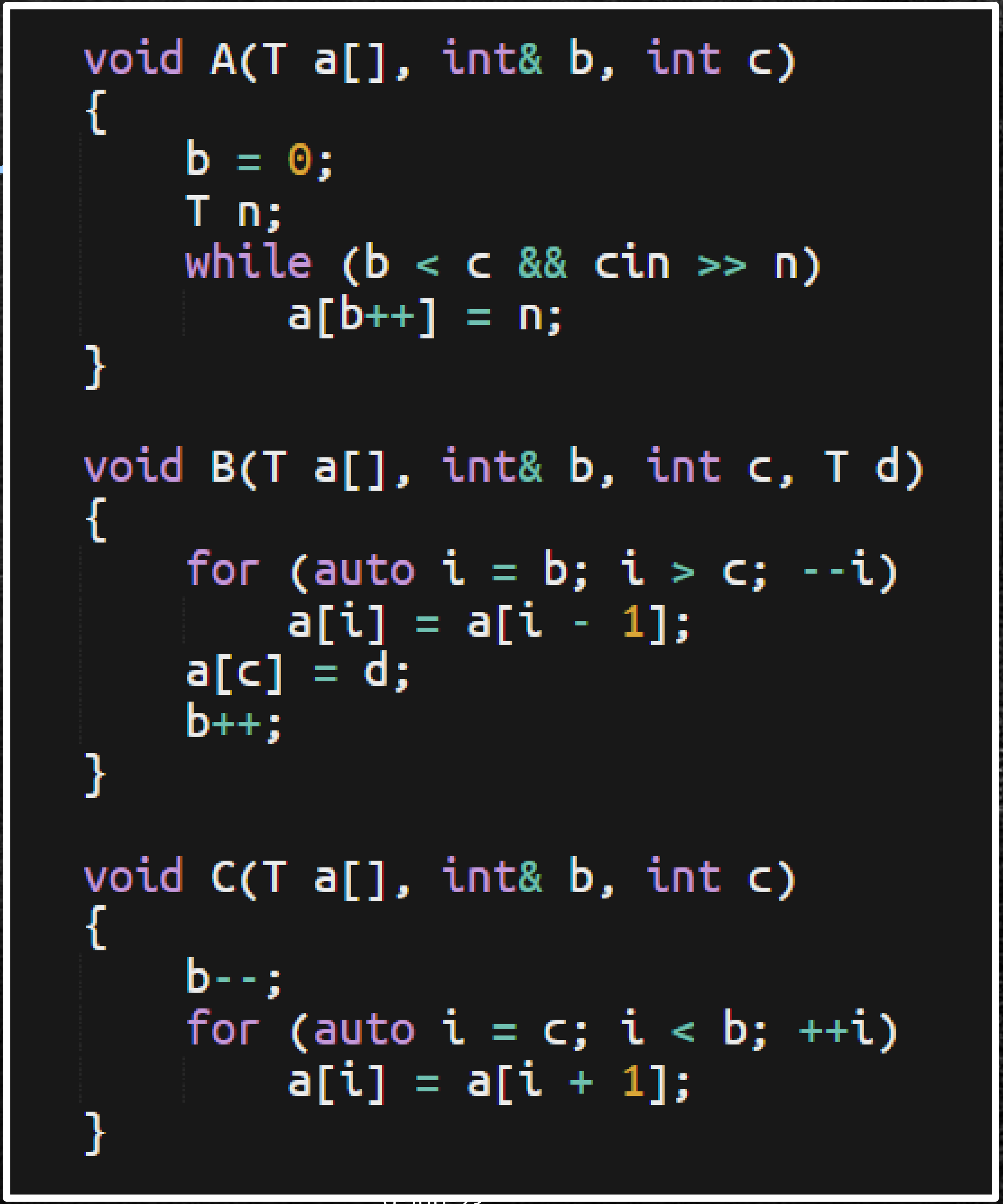
    -
      ```
      while (size < capacity && cin >> e)
        a[size++] = e;
      ```
  - Insert element *e* at position *pos*

    -
      ```
      for (i = ++size; i > pos; i--) a[i] = a[i-1];
      a[pos] = e;
      ```
  - Erase element at position *pos*

  -
    ```
    size--;
    for (i = pos; i < size; i++) a[i] = a[i+1];
    ```

Question

- Which of these algorithms appends elements to the end of the partially-filled array a?
  - A. A( )
  - B. B( )
  - C. C( )
  - E. None of them



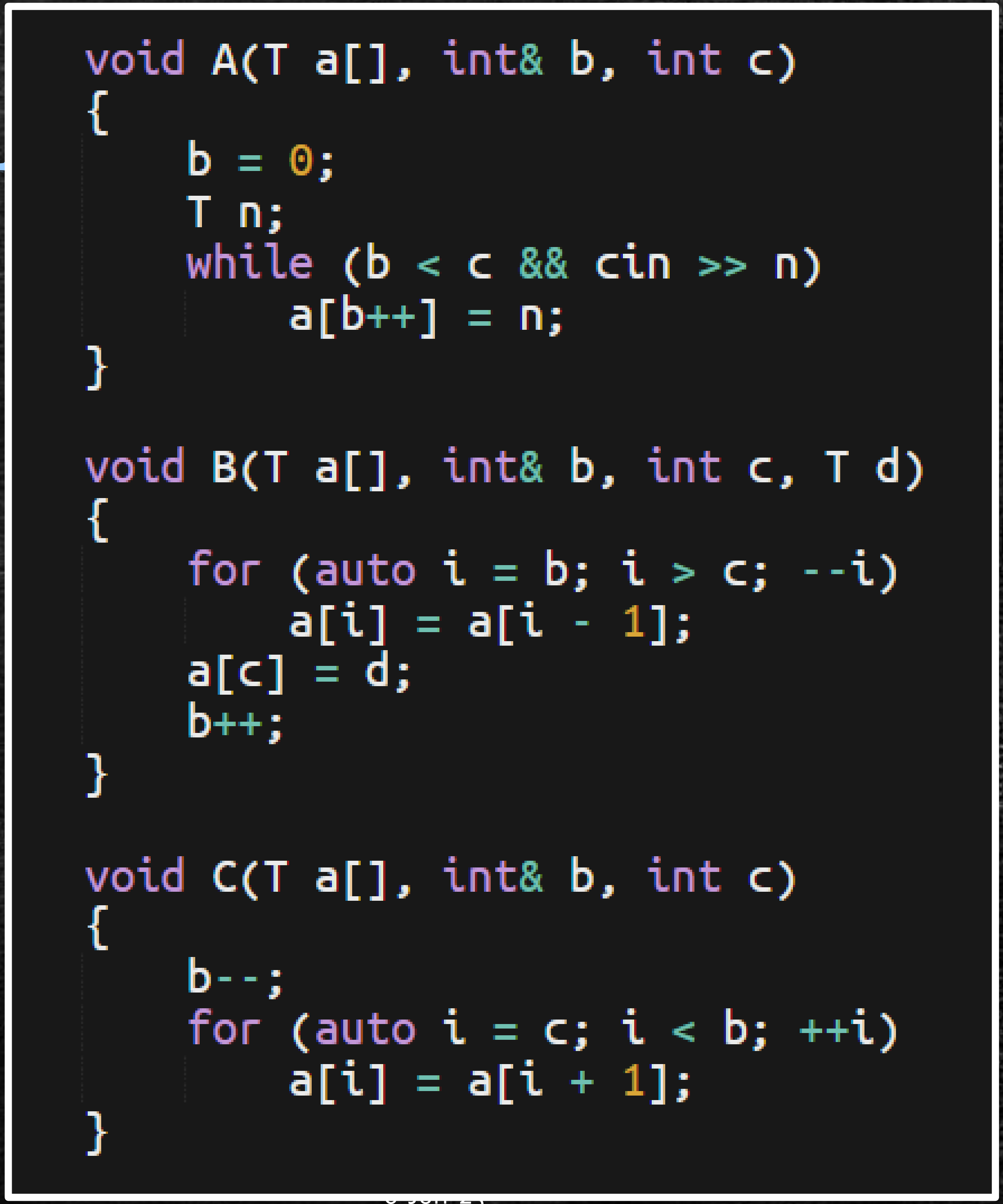
```
void A(T a[], int& b, int c)
{
    b = 0;
    T n;
    while (b < c && cin >> n)
        a[b++] = n;
}

void B(T a[], int& b, int c, T d)
{
    for (auto i = b; i > c; --i)
        a[i] = a[i - 1];
    a[c] = d;
    b++;
}

void C(T a[], int& b, int c)
{
    b--;
    for (auto i = c; i < b; ++i)
        a[i] = a[i + 1];
}
```

Question

- Which of these algorithms inserts a value into the partially-filled array a?
  - A. A( )
  - B. B( )
  - C. C( )
  - E. None of them



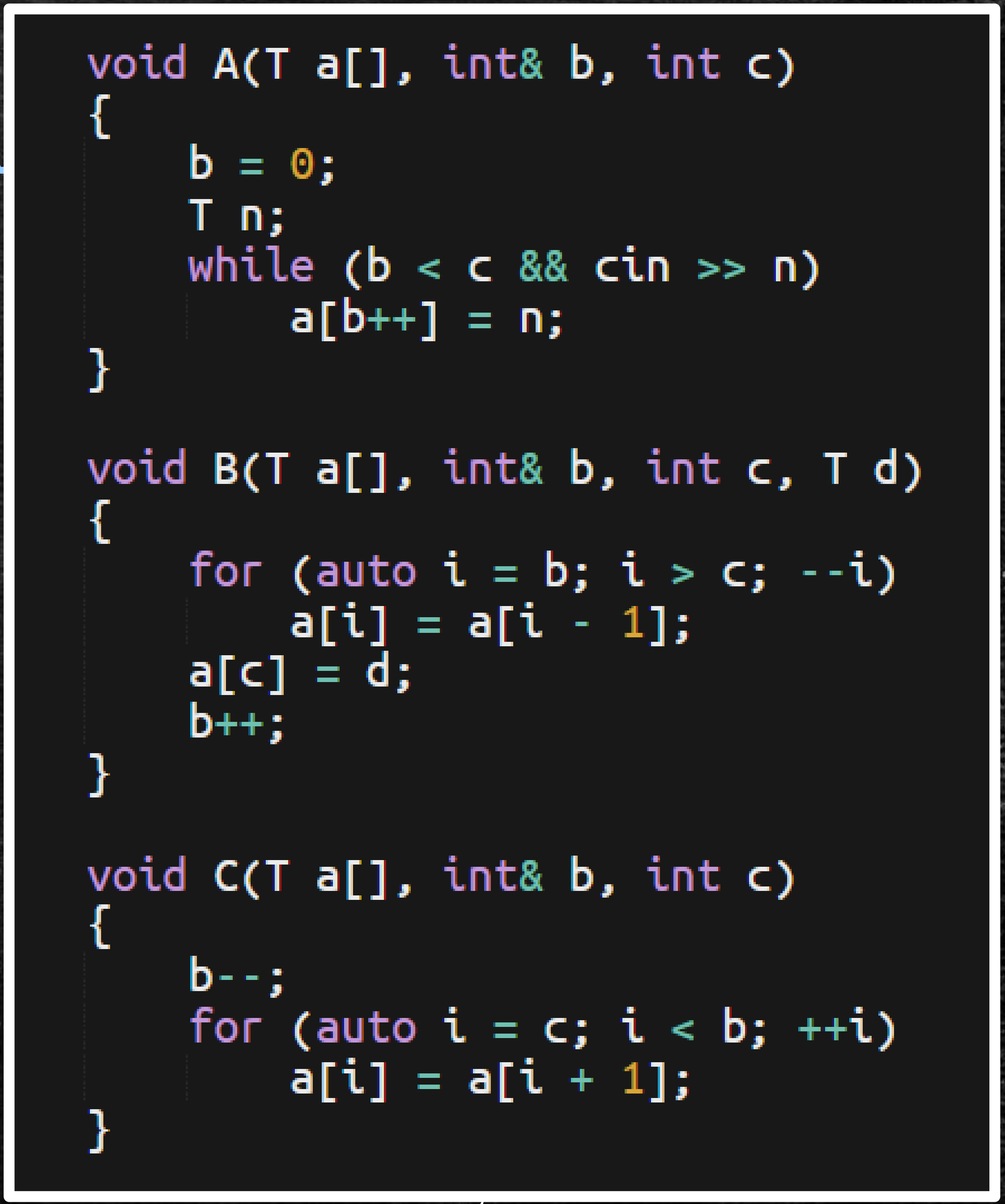
```
void A(T a[], int& b, int c)
{
    b = 0;
    T n;
    while (b < c && cin >> n)
        a[b++] = n;
}

void B(T a[], int& b, int c, T d)
{
    for (auto i = b; i > c; --i)
        a[i] = a[i - 1];
    a[c] = d;
    b++;
}

void C(T a[], int& b, int c)
{
    b--;
    for (auto i = c; i < b; ++i)
        a[i] = a[i + 1];
}
```

Question

- Which of these algorithms erases an elements from the partially-filled array a?
  - A. A( )
  - B. B( )
  - C. C( )
  - E. None of them



```
void A(T a[], int& b, int c)
{
    b = 0;
    T n;
    while (b < c && cin >> n)
        a[b++] = n;
}

void B(T a[], int& b, int c, T d)
{
    for (auto i = b; i > c; --i)
        a[i] = a[i - 1];
    a[c] = d;
    b++;
}

void C(T a[], int& b, int c)
{
    b--;
    for (auto i = c; i < b; ++i)
        a[i] = a[i + 1];
}
```

C-String Review

- A C-string is a NUL terminated character array
  - `char greeting[] = "Hello";`
    `// greeting[] = {'H','e','l','l','o','\0'};`
- The standard library has a collection of functions (inherited from C) in the header `<cstring>`
  - `strlen(str)` counts the characters before the `'\0'`
  - `strcpy(dest, src)` – C-string assignment
  - `strcat(dest, src)` – C-string concatenation
  - `strcmp(str1, str2)` – Comparison: `0`, `<0` && `>0`

Question

```
const char *s1 = "bob", *s2 = "sally";
cout << strcmp(s1, s2) << endl;
```

- *If* one of these prints, which is it?
  - A. `0`
  - B. `1`
  - C. `-1`
  - D. `"bob"`
  - E. `"sally"`

Question

```
const char *s1 = "bob", *s2 = "sally";
cout << strcmp(s2, s1) << endl;
```

- *If* one of these prints, what is it?
  - A. `0`
  - B. `1`
  - C. `-1`
  - D. `"bob"`
  - E. `"sally"`

Question

```
const char *s = "1\020304050";
cout << strlen(s) << endl;
```

- What prints?
  - A. 1
  - B. 10
  - C. 11
  - D. 12
  - E. None of these

Question

```
const char *s = "1/020304050";
cout << strlen(s) << endl;
```

- What prints?
  - A. 1
  - B. 10
  - C. 11
  - D. 12
  - E. None of these

Question

```
const char *src = "bob";
char *dest;
strcpy(dest, src);
```

- What is the error here?
  - A. dest must be const
  - B. src must not be const
  - C. Should be strcpy(src, dest)
  - D. dest is uninitialized
  - E. None of these

Question

```
const char *src = "bob";
char dest[50];
strcpy(dest, src);
```

- What is the error here?
  - A. `dest` must be `const`
  - B. `src` must not be `const`
  - C. Should be `strcpy(src, dest)`
  - D. `dest` is uninitialized
  - E. None of these

Question

```
char buf[50] = strcat("billy", "bob");
cout << buf << endl;
```

- What prints?
  - A. `billybob`
  - B. `billy bob`
  - C. Printing an array, so the address of `buf[0]`
  - D. Does not compile
  - E. Crashes when run

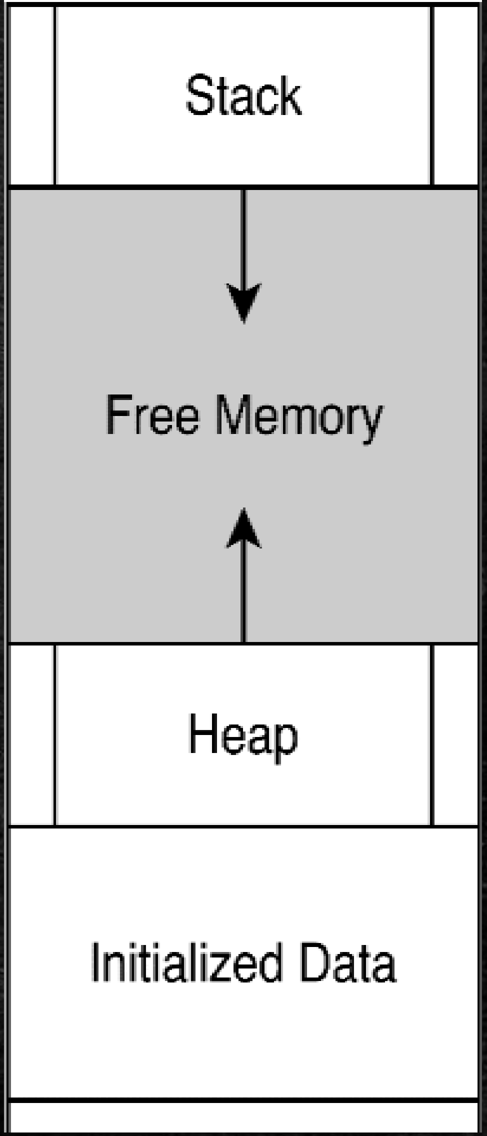
Question

```
char buf[50];
char * s1 = strcpy(buf, "billy");
char * s2 = strcat(s1, "bob");
cout << s2 << endl;
```

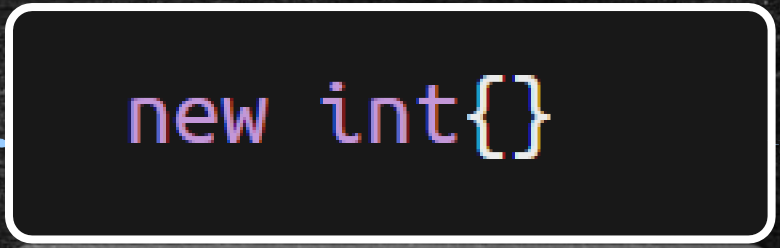
- What prints?
  - A. `billybob`
  - B. `billy bob`
  - C. The address of `buf[0]`
  - D. Does not compile
  - E. Crashes when run

Dynamic Memory Review

- Variables: local (auto) on stack, global in static
- Dynamic: explicitly allocated on the heap using new
  - int *p1 = new int{3}; // initialized int
  - int *ia = new int[3](); // array
- Manual memory management (you)
  - delete p1;   // single object
  - delete[] ia;   // array on heap
- Errors: a) leak, b) 2x delete, c) dangling



Question

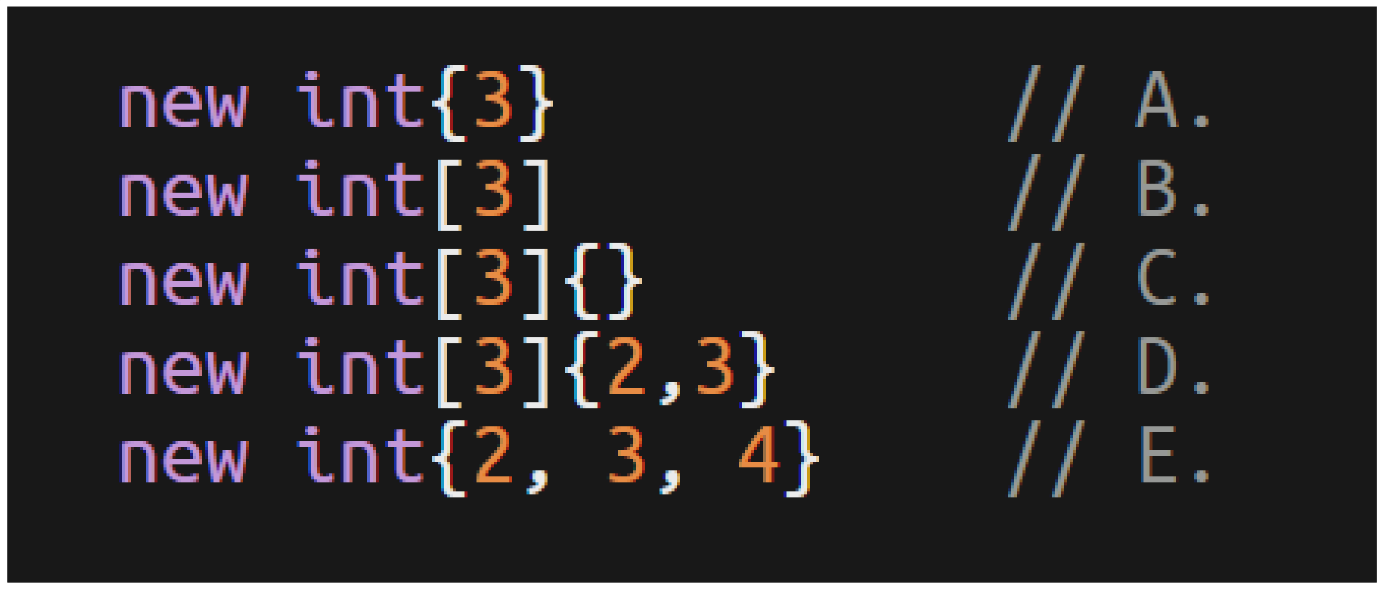


```
new int{}
```

- What is the value of this expression?
  - A. an uninitialized integer
  - B. an integer initialized to `0`
  - C. the address of an uninitialized integer
  - D. the address of an integer initialized to `0`
  - E. An array of uninitialized integers.

Question

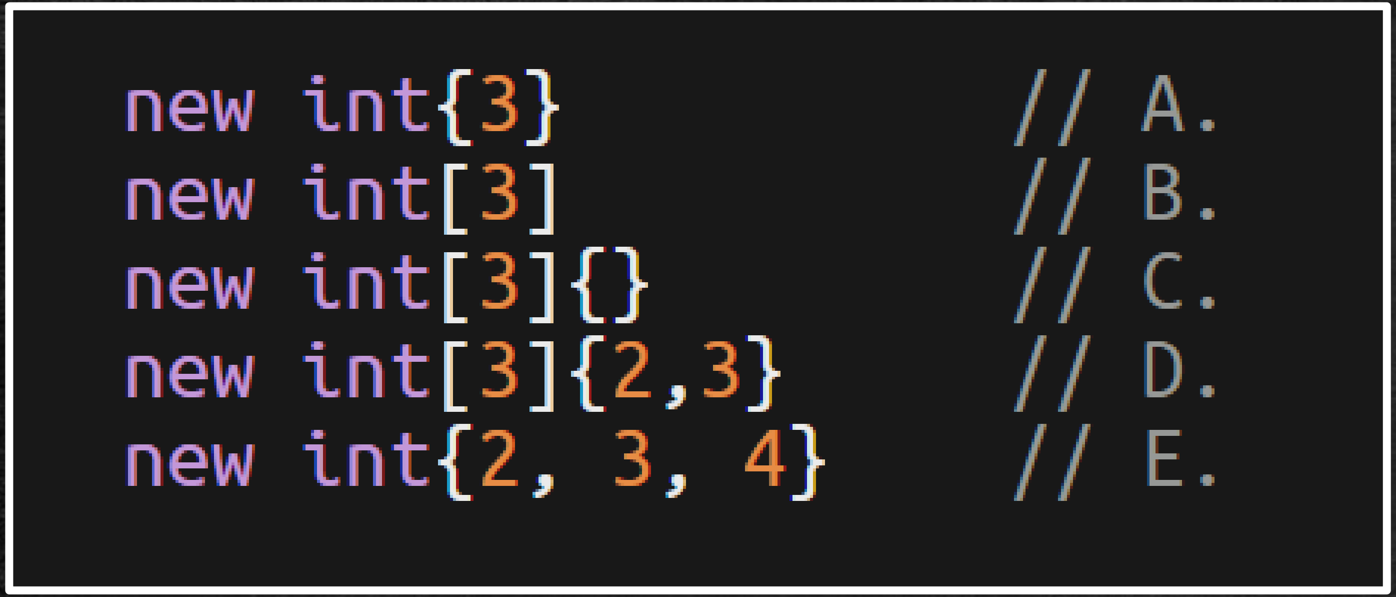
- Which expression below is illegal?



```
new int{3}          // A.
new int[3]          // B.
new int[3]{}        // C.
new int[3]{2,3}     // D.
new int{2, 3, 4}    // E.
```

Question

- Which is the address of the first of three contiguous uninitialized integers allocated on the heap?



```
new int{3}                // A.
new int[3]                // B.
new int[3]{}              // C.
new int[3]{2,3}           // D.
new int{2, 3, 4}          // E.
```

Question

```
double* num = new double{10};
```

- What does the new operator do in this statement?
  - A. It allocates an array of size 10, and yields a pointer to the starting element.
  - B. It allocates enough memory for a double value and initializes it with 10 and returns a pointer to the value
  - C. It allocates enough memory for 10 pointers.
  - D. This is not a legal statement, it will generate a compiler error.

Question

- What is the legal statement to reclaim the memory allocated here?

- A. `delete num;`
- B. `delete[] num;`
- C. `delete num[];`
- D. `delete *num;`

```
int* num = new int;
*num = 10;
```

Question

```
double* deleted;
*deleted = 10;
cout << *deleted;
```

- What is wrong with this code?
  - A. There is a double pointer being used
  - B. There is a deleted pointer being used
  - C. There is an uninitialized pointer being used
  - D. There is a compiler error in the program.
  - E. There is nothing wrong with it

Question

- What is true about this code?
  - A. It's fine.
  - B. It doesn't compile
  - C. It has a memory leak
  - D. It uses a dangling pointer
  - E. It uses an uninitialized pointer

```
int* num = new int;
*num = 10;
cout << num << endl;
delete num;
*num = *num * 2;
cout << num << endl;
```

Question

```
double* num = new double[10];
```

- What does the new operator do in this statement?
  - A. It allocates an array of size 10, and yields a pointer to the starting element.
  - B. It allocates enough memory for a double value and initializes it with 10
  - C. It allocates enough memory for 10 pointers.
  - D. This is not a legal statement, it will generate a compiler error.

Question

- What is the problem with this code?
  - A. Has an off by 1 error
  - B. It dereferences an uninitialized pointer
  - C. It has an array out of bounds error
  - D. It has a memory leak
  - E. It uses a dangling pointer

```
string* mw[20];
for (int i = 0; i < 14; i++) {
   mw[i] = new string("Hello");
}
cout << *mw[15] << endl;
. . . // more code
```

Question

```
int* pArray[10];
```

- Which of the following best describes the nature of `pArray`?
  - A. It is a pointer to an array of 10 integers.
  - B. It is a pointer to an integer initialized with 10.
  - C. It is an array of ten integer pointers on the stack.
  - D. It is an array of ten integer pointers on the heap.
  - E. There is a compilation error.

Question

- What is the problem with this code?

```
string* mw;
for (int i = 0; i < 14; i++) {
   mw = new string("Hello");
}
cout << *mw << endl;
. . . // more code
```

  - A. Has an off by 1 error
  - B. It dereferences an uninitialized pointer
  - C. It has an array out of bounds error
  - D. It has a memory leak
  - E. It uses a dangling pointer

LEC-7A Preview-Objects & Classes

- The wall of abstraction and the class definition
  - The `public` and `private` sections
  - The implicit parameter and the `this` pointer
  - Accessor and mutator member functions
- Using constructors to initialize objects
  - The default or no-argument constructor
  - The working and synthesized default constructors
  - Using assignment vs. using the initializer list
  - Conversion constructors and the `explicit` modifier

LEC-7B Preview-Classes & Inheritance

- Assignment, copying and destruction
- Static members
  - static data members and static member functions
  - static const data members
- Classification and Inheritance
  - The superclass (base class) and subclass (derived class)
  - Using UML diagrams to illustrate inheritance
  - Inherited and private base-class members
  - Writing derived class constructors (using the initializer list)
  - Protected members

LEC-7C-Inheritance & Polymorphism

- The `virtual` keyword and member function overriding
  - Redefining a derived `to_string()` in `Student`
  - Combining or extending `to_string()` in `Student`
  - Using `override` to check for errors at compile time
- Class relationships and stream substitutability
  - Barbara Liskov and the substitution principle
  - The association or uses-a relationship
  - Composition or the has-a relationship
  - Public inheritance and the is-a relationship
- Polymorphic inheritance and polymorphic functions

LEC-7D-Polymorphism & Abstract Classes

- Applications of Inheritance
  - Polymorphic lists of pointers
  - How early and late binding work under the hood
  - Multiple inheritance
  - Contraction and private inheritance
- Specification Inheritance
  - Abstract classes and pure-virtual functions
  - Using an abstract class
  - Final functions and classes

Week 7 Homework Preview

- Week 7 HW due by 1pm July 31th (Mon) or August 1st (Tue)
- H30 - Classes: A Bug's Life
- H31 - Classes: the Image class
- H32 - Inheritance: Point, Circle, Cylinder
- H33 - Abstract Classes: Virtual Workers of the World

Programming Exam 8, 9 & Midterm #3

- Now – Programming Exam #8
  - I will collect your cellphones, watches & electronics
  - Place all books, backpacks, notes at front or back of the room
  - Move to your assigned seat; do not log in
  - I will start PE08 on your computer
  - Log in using your Homework Console credentials
  - When you are done, submit the exam and leave
- Come back by 3pm when PE 09 will start
- Come back by 4pm when Midterm #3 will start