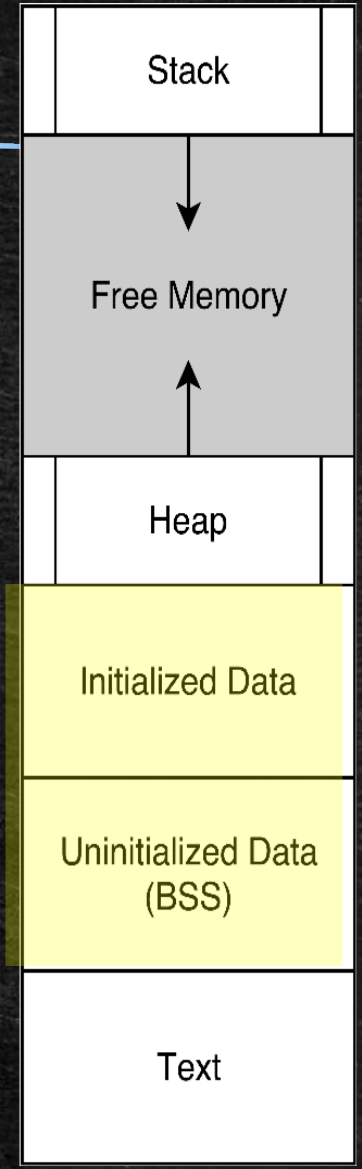
Week 6

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

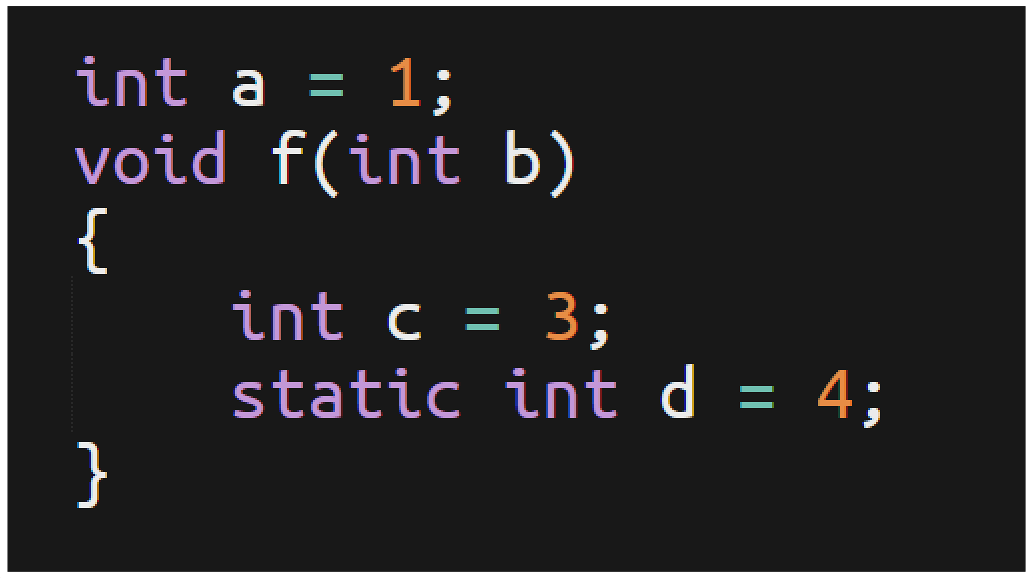
Memory Review

- 3 characteristics of objects, functions, classes
  - Scope: where a name is visible (file, block)
  - Duration: time in memory (auto, static, dynamic)
  - Linkage: in multi-file code (external, internal, none)
- 3 areas of memory: static (text), stack, heap
  - static storage: globals, constants & code
  - stack: locals, parameters, runtime mechanics
  - heap: dynamic variables controlled by programmer



Question

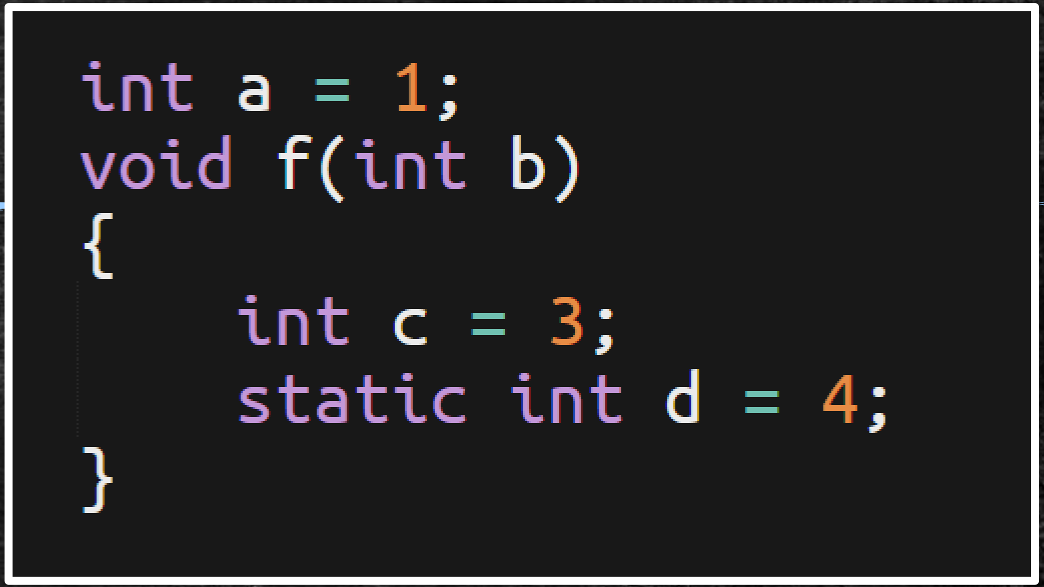
- The variable ***a*** is stored:
  - A. on the stack
  - B. on the heap
  - C. in the static storage area
  - D. You can't tell from this example



```
int a = 1;
void f(int b)
{
    int c = 3;
    static int d = 4;
}
```

Question

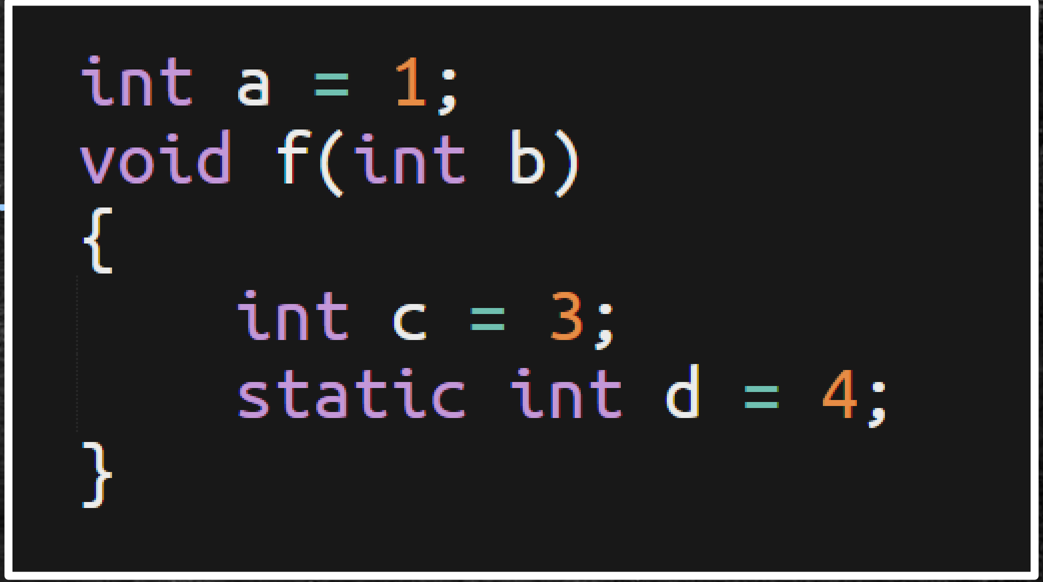
- The variable ***b*** is stored:
  - A. on the stack
  - B. on the heap
  - C. in the static storage area
  - D. You can't tell from this example



```
int a = 1;
void f(int b)
{
    int c = 3;
    static int d = 4;
}
```

Question

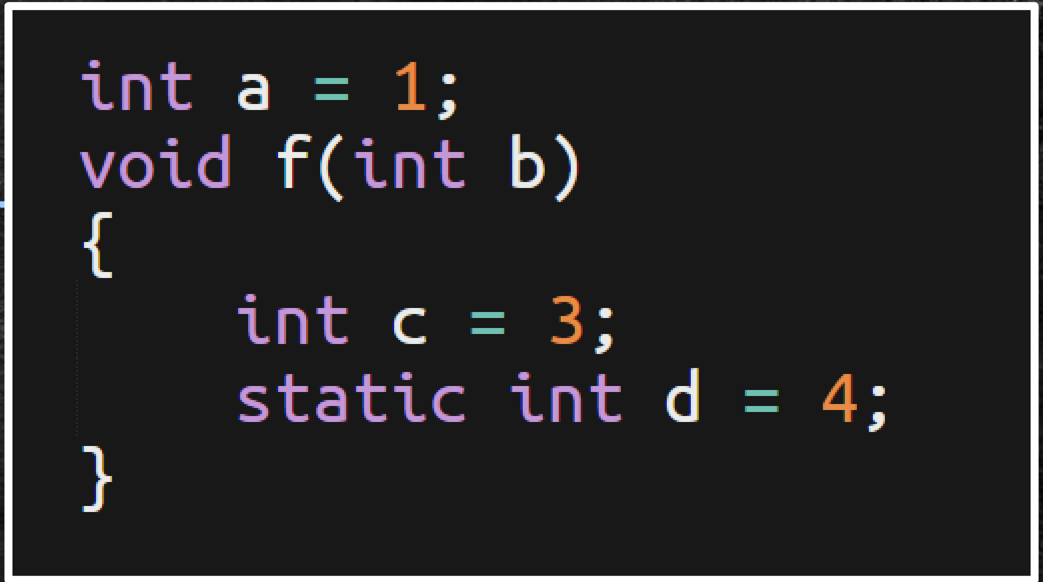
- The variable ***c*** is stored:
  - A. on the stack
  - B. on the heap
  - C. in the static storage area
  - D. You can't tell from this example



```
int a = 1;
void f(int b)
{
    int c = 3;
    static int d = 4;
}
```

Question

- The variable ***d*** is stored:
  - A. on the stack
  - B. on the heap
  - C. in the static storage area
  - D. You can't tell from this example



```
int a = 1;
void f(int b)
{
    int c = 3;
    static int d = 4;
}
```

Question

```
static char c = 'c';
int f() { return 21; }
```

- This code appears in `f1.cpp`.
  What is the linkage of the variable **c**?
  - A. static linkage
  - B. no linkage
  - C. internal linkage
  - D. external linkage

Question

```
char c = 'c';
int f() { return 21; }
```

- This code appears in `f1.cpp`.
  What is the duration of the variable **c**?
  - A. static duration
  - B. automatic duration
  - C. dynamic (programmer defined) duration
  - D. internal duration

Question

```
char c = 'c';
int f() { return 21; }
```

- This code appears in `f1.cpp`.
  What is the scope of the variable **c**?
  - A. internal scope
  - B. local scope
  - C. block scope
  - D. file scope
  - E. global scope

Pointer Review

- A pointer is a variable that contains an address
  - A pointer will be in one of 4 "states"
  - a) valid, b) one-past a sequence, c) null, d) invalid
  - Only valid pointers can be dereferenced
- Valid pointers may be initialized with:
  - a) address operator, b) new operator c) name of an array, d) another (valid) pointer, e) name of a function
- Skills: define, initialize, assign, dereference
- A pointer may be `const` or point to `const` or both

Question

```
int* ptr = nullptr;
cout << &ptr << endl;
```

- What is the output?
  - A. No output; a compiler error
  - B. `0 0`
  - C. The address where *`ptr`* is stored
  - D. "nullptr"
  - E. Compiles. Undefined behavior when run

Question

```
int* ptr = nullptr;
cout << *ptr << endl;
```

- What is the output?
  - A. No output; a compiler error
  - B. `0` `0`
  - C. The address value where *`ptr`* is stored
  - D. "nullptr"
  - E. Compiles fine. Undefined behavior when run.

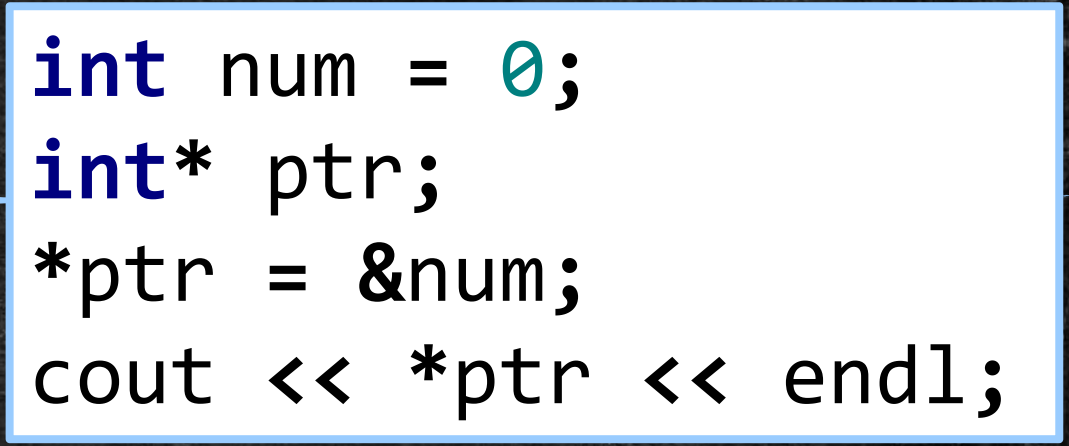
Question

```
int* ptr = nullptr;
cout << ptr << endl;
```

- What is the output?
  - A. No output; a compiler error
  - B. `0` `0`
  - C. The address value where *ptr* is stored
  - D. "nullptr"
  - E. Compiles fine. Undefined behavior when run.

Question

- What is the output?
  - A. 0
  - B. The address of *num*
  - C. Undefined behavior when run.
  - D. Will not compile



```
int num = 0;
int* ptr;
*ptr = &num;
cout << *ptr << endl;
```

Question

```
int* ptr = &0;
cout << *ptr << endl;
```

- What is the output?
  - A. 0
  - B. The address of 0 in memory
  - C. Undefined behavior
  - D. Will not compile

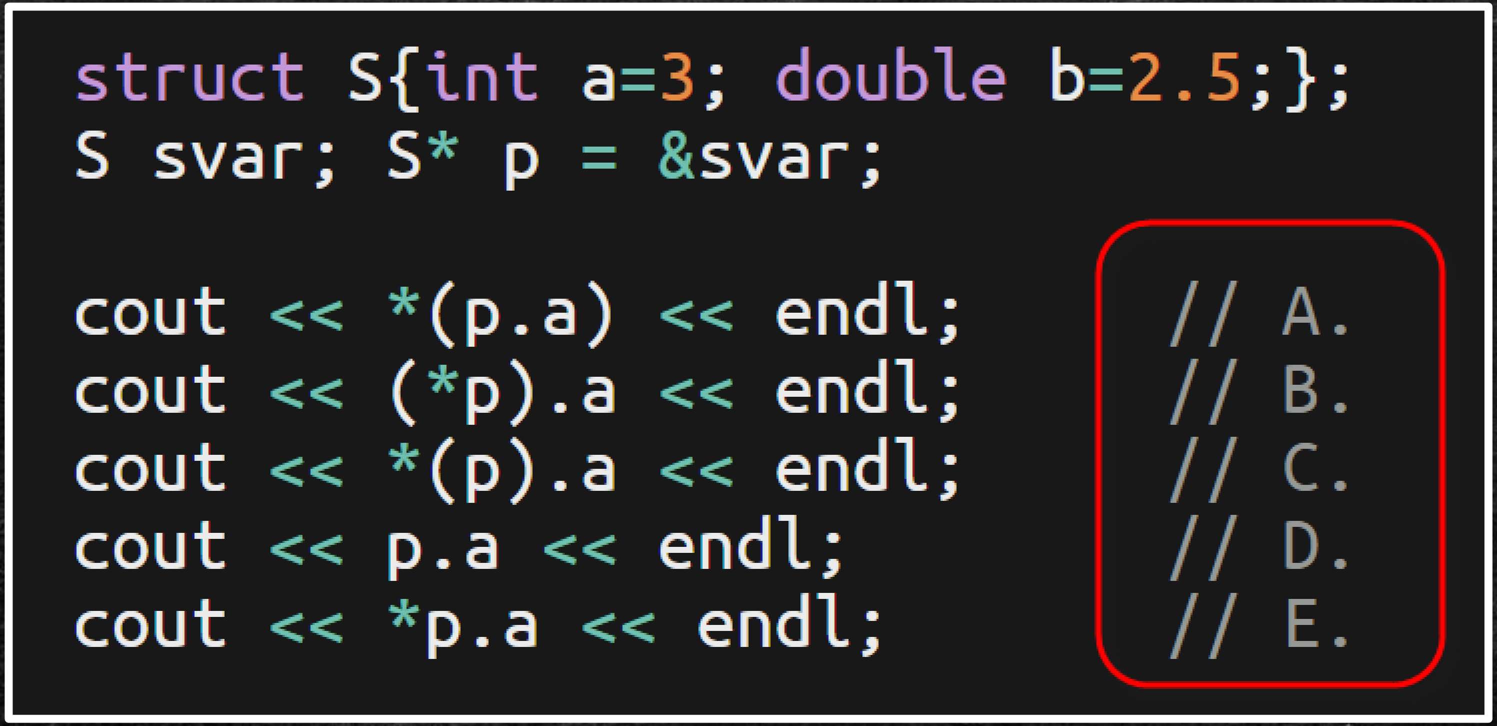
Pointer & Structure Review

- Pointers to structures

```
struct Point {int x, y; }; // define structure  type
Point pt{3,  4}; // define & init a structure variable
Point *p = &pt;   // define & init a pointer to struct
cout << (*p).x << ",  "      // dereference & select x
  << p->y << endl;           // dereference & select y
```

Question

- Which line below is correct?

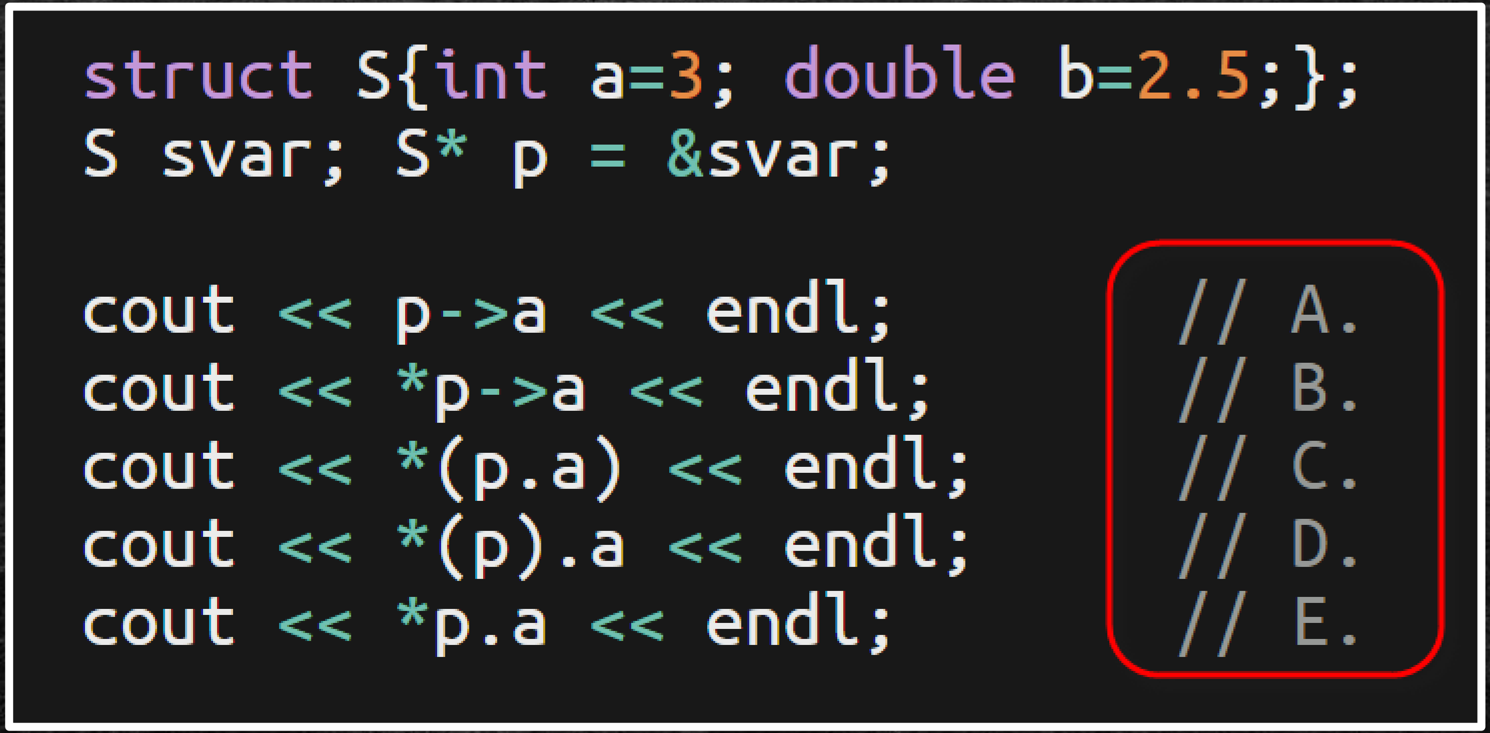


```
struct S{int a=3; double b=2.5;};
S svar; S* p = &svar;

cout << *(p.a) << endl;          // A.
cout << (*p).a << endl;          // B.
cout << *(p).a << endl;          // C.
cout << p.a << endl;             // D.
cout << *p.a << endl;            // E.
```

Question

- Which line below is correct?



```
struct S{int a=3; double b=2.5;};
S svar; S* p = &svar;

cout << p->a << endl;                   // A.
cout << *p->a << endl;                  // B.
cout << *(p.a) << endl;                 // C.
cout << *(p).a << endl;                 // D.
cout << *p.a << endl;                   // E.
```

Pointers & Graphics Review

- C-language stb image libraries process graphics files

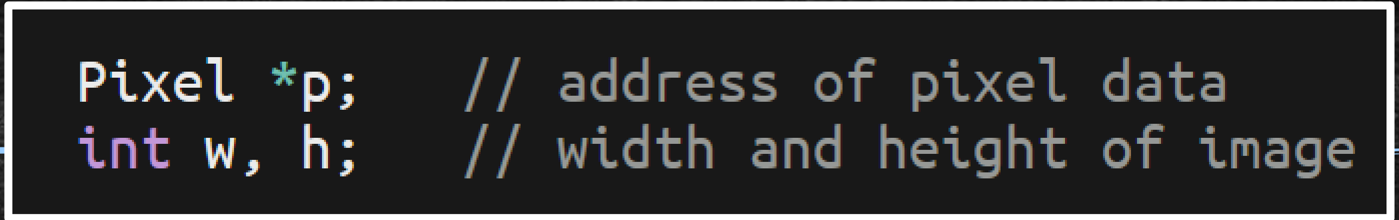
```
– using uc = unsigned char;  // a type alias
  int w, h, bpp;             // filled in through function
  uc* const data = stbi_load("cat.png", &w, &h, &bpp, 4);
```

- Returns a pointer to first byte of image data allocated on heap
  - Create a pair of pointers to traverse all of the data
  - Can't use data pointer, so create a beg pointer: `uc* beg = data;`
  - Create an end pointer like this: `uc* end = data + w * h * 4`
  - Move the beg pointer using increment to reach the next byte

Pixels & Structures Review

- With *stbi_load()*, *data* points to a single *unsigned char*
  - Each pixel in the image has 4 of these (red, green, blue, alpha).
- Process a whole *Pixel* by creating a structure with 4 members
  - Initialize *beg* pointer with *reinterpret_cast<Pixel*>(data)*
  - Now *beg* pointer will look at image data as *Pixel*
- Address arithmetic: pointer + n = new address
  - *n* expressed in element size (*Pixel* in our case)
  - `Pixel* end = beg + w * h; // don't need the 4`

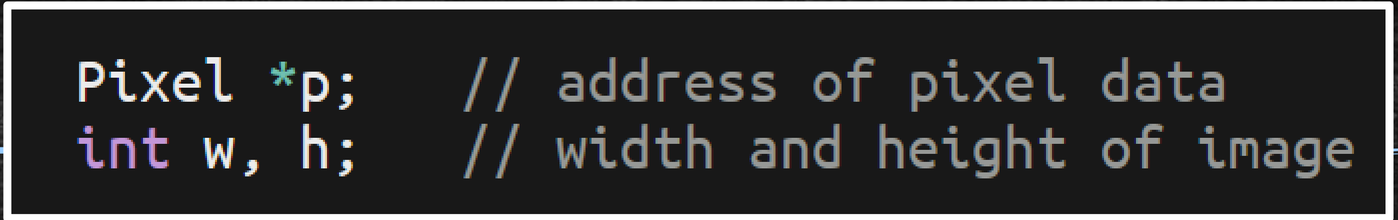
Question



```
Pixel *p;    // address of pixel data
int w, h;    // width and height of image
```

- What is the address of the last row?
  - A. `p + w * h`
  - B. `p + w * (h - 1)`
  - C. `p + w + h`
  - D. `p + w + (h - 1)`
  - E. None of these are correct

Question

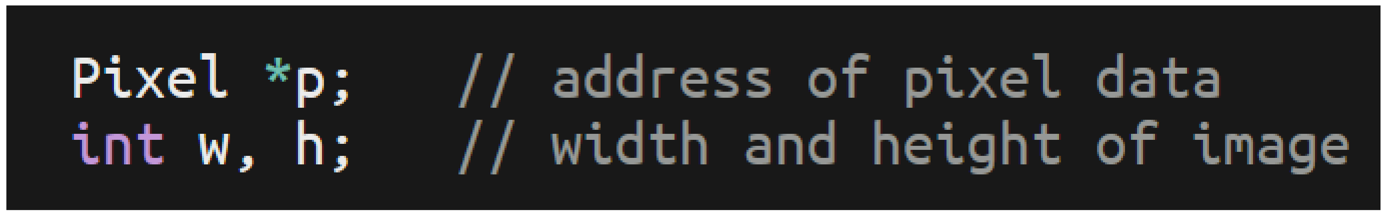


```
Pixel *p;    // address of pixel data
int w, h;    // width and height of image
```

- What returns the last pixel on the first row?

  - A. `p + w - 1`
  - B. `*p + w - 1`
  - C. `*(p + w) - 1`
  - D. `*(p + w - 1)`
  - E. None of these are correct

Question



```
Pixel *p;    // address of pixel data
int w, h;    // width and height of image
```

- What returns the last pixel on the last row?

  – A. `p + w * h - 1`
  – B. `*p + w * h - 1`
  – C. `*(p + w * h) - 1`
  – D. `*(p + w * h - 1)`
  – E. None of these are correct

Array Review

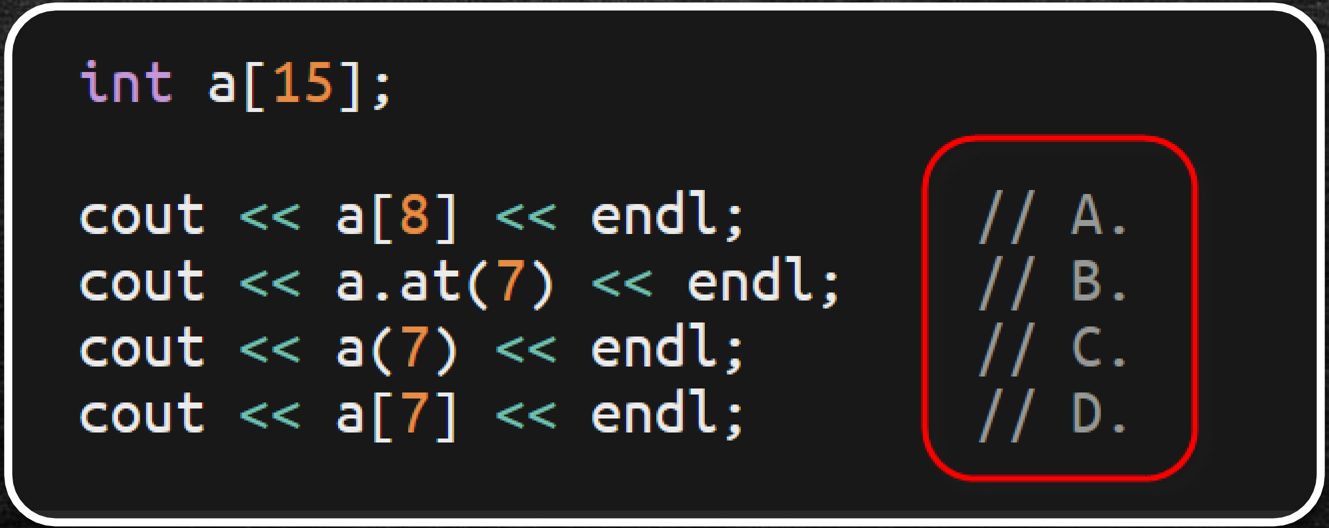
- Array: built-in list of elements (homogenous)
  - `int a[5], b[] = {1, 2, 3};`
  - Name: address of first element, not a variable
  - Use subscript to access: `a[0]`
  - No range checking or exceptions
- Dereferencing: `*` and `[]` operators: any address
  - Combination of address and offset

```
address[offset] ⇔ *(address + offset)
```

  - ptr + 2 is address expression (adds two elements to address)

Question

- Which displays the eighth element of a?



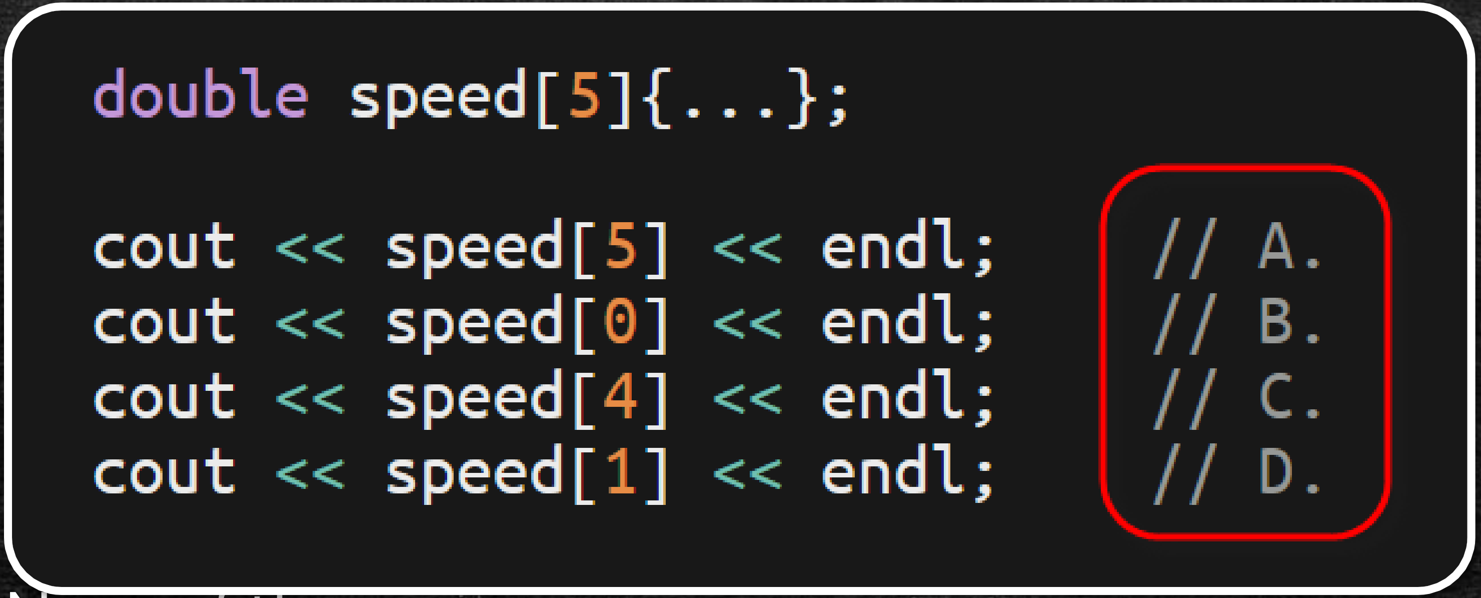
```
int a[15];

cout << a[8] << endl;              // A.
cout << a.at(7) << endl;           // B.
cout << a(7) << endl;              // C.
cout << a[7] << endl;              // D.
```

  - E. Runtime error because a is uninitialized

Question

- Which line has undefined output?

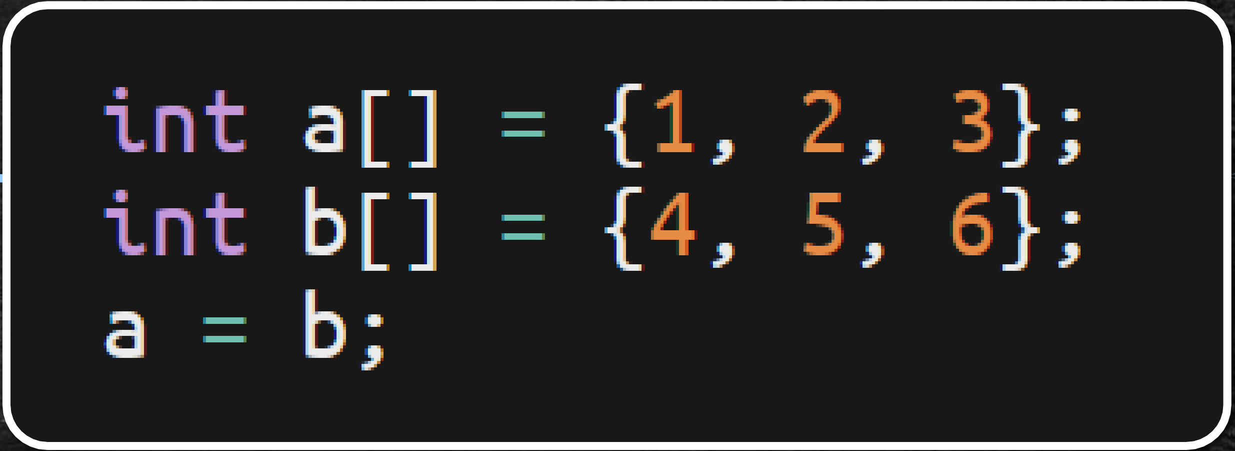


```
double speed[5]{...};

cout << speed[5] << endl;      // A.
cout << speed[0] << endl;      // B.
cout << speed[4] << endl;      // C.
cout << speed[1] << endl;      // D.
```

  - E. None of these

Question

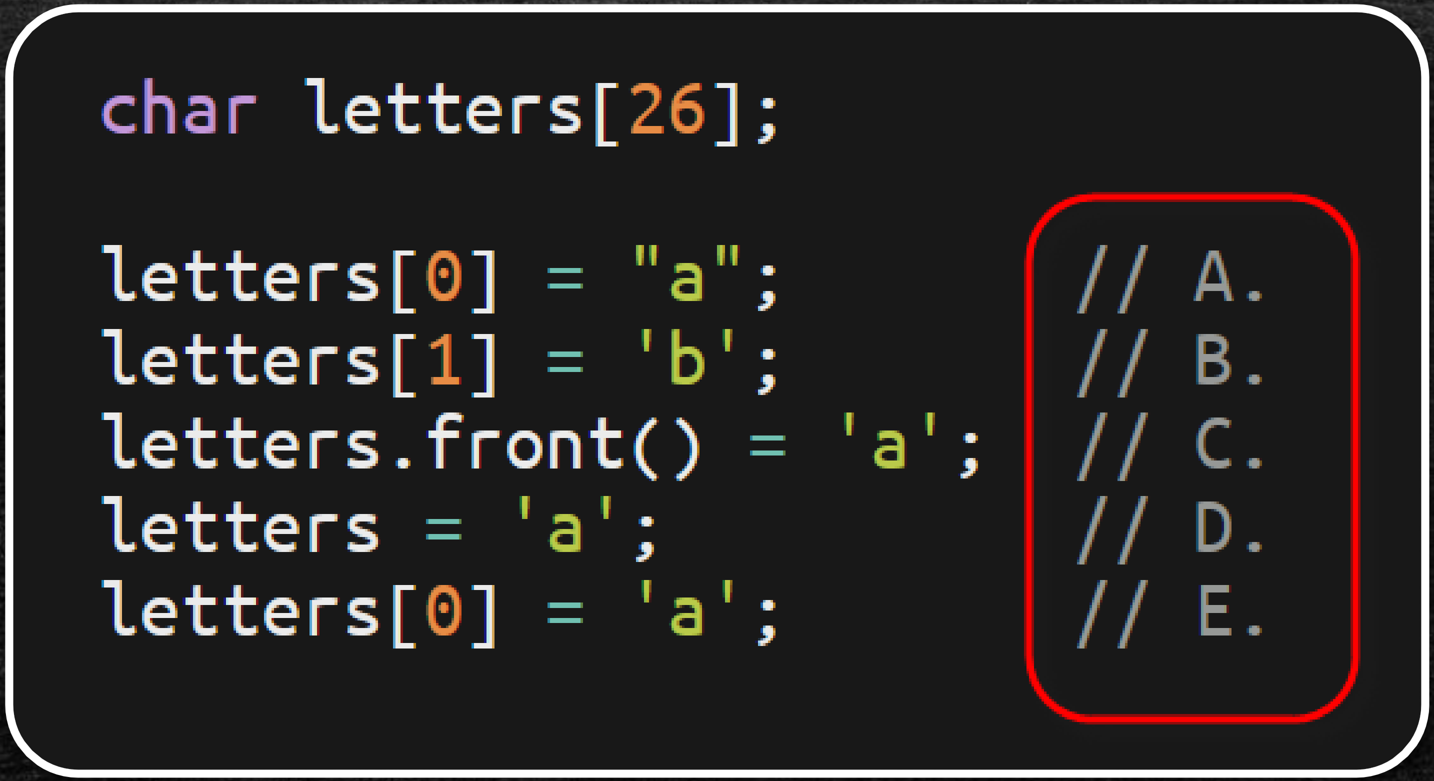


```
int a[] = {1, 2, 3};
int b[] = {4, 5, 6};
a = b;
```

- What does the array a contain after this runs?
  - A. {1, 2, 3}
  - B. {4, 5, 6}
  - C. Undefined behavior
  - D. Syntax error - code will not compile

Question

- Which assigns to the first position in letters?



```
char letters[26];

letters[0] = "a";          // A.
letters[1] = 'b';          // B.
letters.front() = 'a';     // C.
letters = 'a';             // D.
letters[0] = 'a';          // E.
```

Question

- Assume `int dates[10];` What is the equivalent array notation for: `*(dates + 2);`
  - A. `dates[2]`
  - B. `dates[0] + 2`
  - C. `dates[2] + 2`
  - D. `&dates[2]`
  - E. `dates[0] + 4`

Question

- Assume `int dates[10];` What is the equivalent array notation for: `*dates + 2;`

  - A. `dates[2]`
  - B. `dates[0] + 2`
  - C. `dates[2] + 2`
  - D. `&dates[2]`
  - E. `dates[0] + 4`

Question

- Assume `int dates[10];` What is the equivalent array notation for: `(*dates + 2) + 2;`

  - A. `dates[2]`
  - B. `dates[0] + 2`
  - C. `dates[2] + 2`
  - D. `&dates[2]`
  - E. `dates[0] + 4`

Question

- Assume the following code. What prints?

  -
    ```
    int ar[] = {1, 2, 3, 4, 5};
    int *p = ar + 2;
    cout << *p++ << ",";
    cout << *p << endl;
    ```

  - A. `2, 3`
  - B. `3, 4`
  - C. `4, 4`
  - D. `4, 5`

Question

- Assume the following code. What prints?

  -
    ```
    int ar[] = {1, 2, 3, 4, 5};
    int *p = ar + 2;
    cout << *++p << ",";
    cout << *p << endl;
    ```
  - A. `2, 3`
  - B. `3, 4`
  - C. `4, 4`
  - D. `4, 5`

Question

- Assume the following code. What prints?

```
int ar[] = {1, 2, 3, 4, 5};
int *p = ar + 2;
cout << ++*p << ",";
cout << *p << endl;
```

- A. 2, 3
- B. 3, 4
- C. 4, 4
- D. 4, 5

Arrays & Function Review

- Define an array: explicitly allocate or initialize
  - `int a[5], b[] = { 1, 2, 3 }, c[3]{};`
  - The array name is the address of first element
- Declare the function (do not put the size in brackets)
  - `double avg(const int ar[], size_t size);`
  - `void avg(int ar[], size_t size);`
- Call the function: `result = average(a, 5);`
- Loops: range, iterator, counter-controlled, sentinel
- Algorithms: count, cumulative, extremes, fencepost

Question

- What does this print?
  - A. 15
  - B. 20
  - C. 12
  - D. 3
  - E. 35

```cpp
int a[] = {6, 2, 1, 9, 5, 12}, x = 0;
auto p = begin(a);
while (p != end(a)) x += *p++;
cout << x << endl;
```

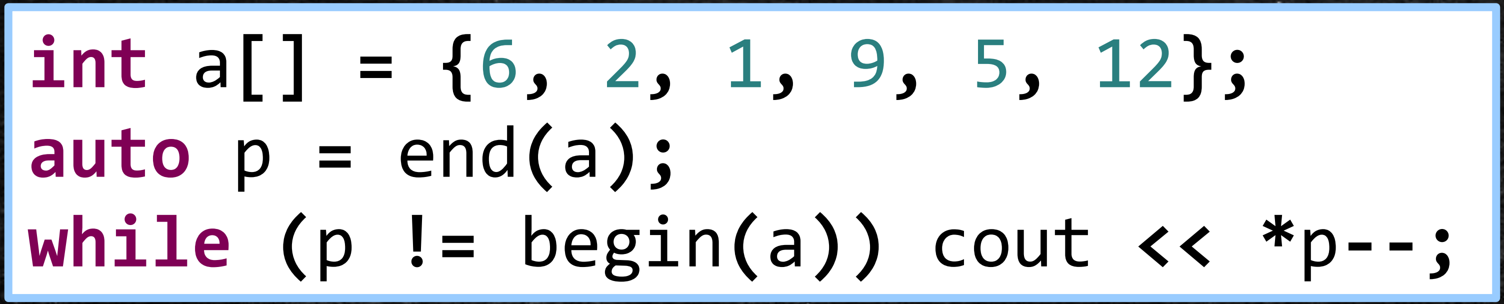
Question

- What does this print?
  - A. 1259126
  - B. 125912
  - C. 59126
  - D. Undefined behavior

```
int a[] = {6, 2, 1, 9, 5, 12};
auto p = end(a);
while (p-- != begin(a)) cout << *p;
```

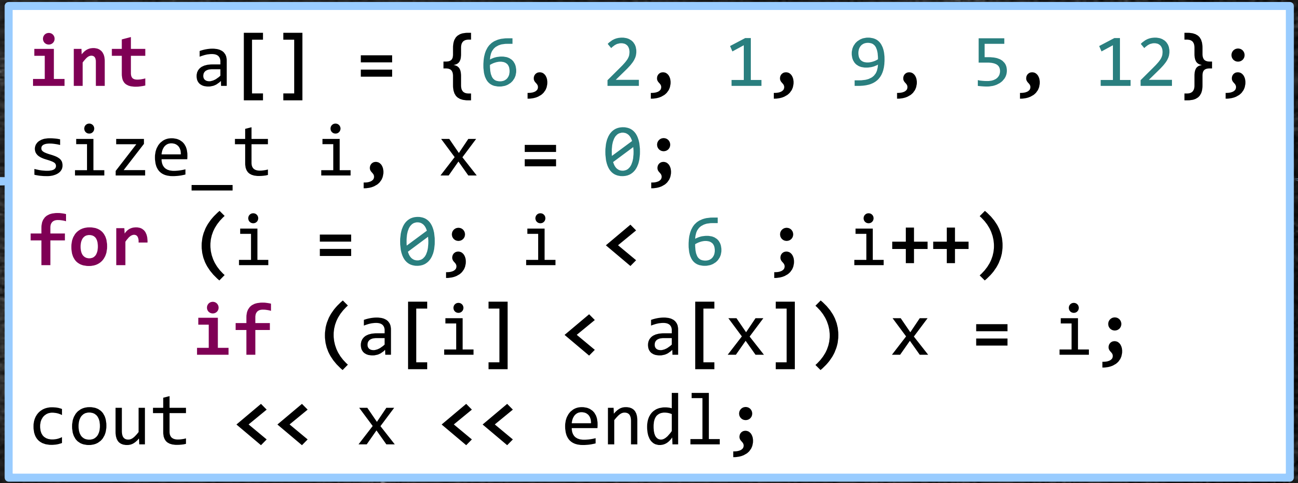
Question

- What does this print?
  - A. 1259126
  - B. 125912
  - C. 59126
  - D. Undefined behavior



```
int a[] = {6, 2, 1, 9, 5, 12};
auto p = end(a);
while (p != begin(a)) cout << *p--;
```

Question



```
int a[] = {6, 2, 1, 9, 5, 12};
size_t i, x = 0;
for (i = 0; i < 6 ; i++)
    if (a[i] < a[x]) x = i;
cout << x << endl;
```

- What does this print?
  - A. 0
  - B. 1
  - C. 2
  - D. 12
  - E. Undefined behavior

Question

- What is the correct prototype for this call?
  - A. `void f(size_t n, int a[]);`
  - B. `void f(int[] a, size_t n);`
  - C. `void f(int a[], size_t n);`
  - D. `void f(int a*, size_t n);`
  - E. More than one of these will work

```
const size_t capacity = 100;
int a[capacity];
f(a, capacity / 2);
```

Question

- What does this code do?
  - A. Sums the elements in a
  - B. Counts the elements in a
  - C. Changes the elements in a
  - D. Counts the last value in a
  - E. Nothing; no effect

```
int a[] = {6, 2, 1, 9, 5, 12}, x = 0;
for (auto e : a) x++;
cout << x << endl;
```

Question

- What does this code do?
  - A. Sums the elements in a
  - B. Counts the elements in a
  - C. Changes the elements in a
  - D. Syntax error; does not compile.
  - E. Nothing; no effect

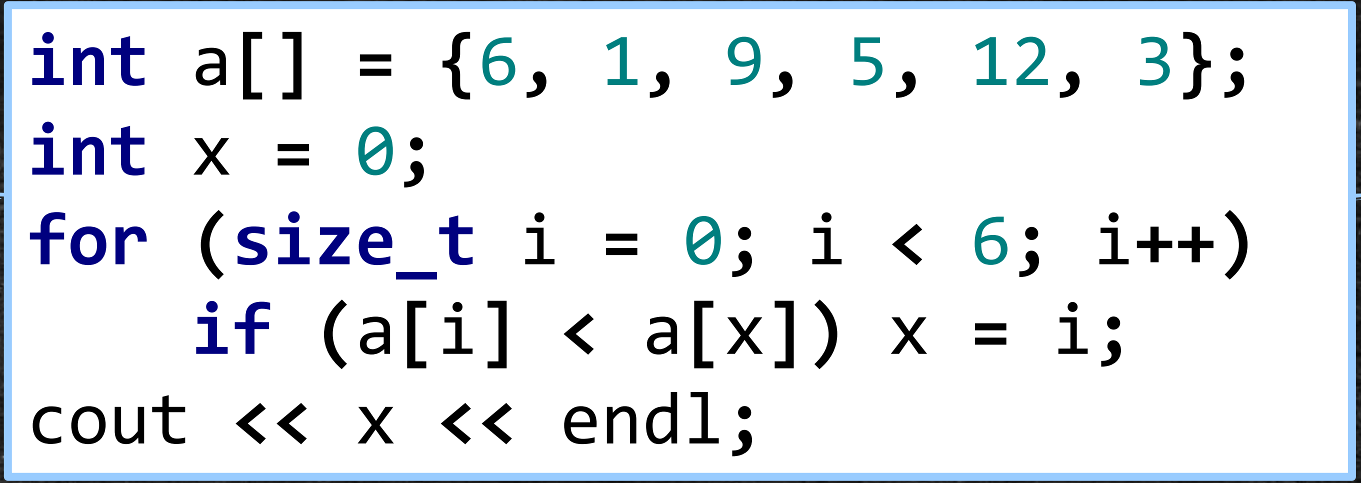
```cpp
int a[] = {6, 2, 1, 9, 5, 12}, x = 0;
for (auto e : a) x += a;
cout << x << endl;
```

Question

```cpp
int a[] = {6, 1, 9, 5, 12, 3};
int x = a[0];
for (size_t i = 0; i < 6; i++)
    if (a[i] > x) x = a[i];
cout << x << endl;
```

- What does this loop do?
  - A. Counts the elements in a
  - B. Sums the elements in a
  - C. Finds the largest value in a
  - D. Finds the smallest value in a
  - E. Finds the last element in a

Question



```
int a[] = {6, 1, 9, 5, 12, 3};
int x = 0;
for (size_t i = 0; i < 6; i++)
    if (a[i] < a[x]) x = i;
cout << x << endl;
```

- What does this loop print?
  - A. 6
  - B. 1
  - C. 12
  - D. 2
  - E. 4

Question

```
int a[] = {6, 1, 9, 5, 12, 3};
int x = 0;
for (size_t i = 0; i < 6; i++)
    if (a[i] % 2 == 0) x++;
cout << x << endl;
```

- What does this loop do?
  - A. Counts the even elements in a
  - B. Sums the even elements in a
  - C. Counts the odd elements in a
  - D. Finds the largest value in a
  - E. Finds the smallest value in a

Question

```cpp
int a[] = {6, 1, 9, 5, 12, 3};
int x = 0;
for (size_t i = 0; i < 6; i++)
    if (a[i] % 2 == 1) x += a[i];
cout << x << endl;
```

- What does this loop print?
  - A. 18
  - B. 22
  - C. 2
  - D. 4
  - E. 36

LEC-6A Preview-More on Arrays

- Partially-filled Arrays
  - Why use partially-filled arrays?
  - Creating a partially-filled array (size and capacity)
  - Appending elements and traversing the array
  - Inserting and deleting elements
- 2D Arrays
  - Creating and initializing a 2D array
  - Passing 2D arrays to functions
- C-Strings
  - Creating and initializing C-Strings (array-based and pointer)

LEC-6B Preview-C-style Strings

- C++ strings (`<string>`) vs C-style Strings (`<cstring>`)
  - Using `strlen()` instead of `str.size()`
  - C-String assignment with `strcpy()` & `strncpy()`
  - Concatenation with `strcat()` & `strncat()`
  - Comparing C-Strings (lexicographically) with `strcmp()`
- Implementing C-String functions
  - Three versions of `strlen()` & a common C++ idiom
  - The `strcpy()` and `strcmp()` functions (GNU/Apple)
- Writing your own C-String functions
  - `find_first()`, `find_last()`, `first_of_any()` and `find_target()`

LEC-6C Preview-Dynamic Memory

- The shell and the command-line
  - Processing and converting command-line arguments
- Dynamic memory and the heap
  - Using the `new` operator to allocate variables on the heap
  - Using `new` to allocate dynamic arrays
  - Using `new` to allocate structure and object variables
- Manual memory management
  - Freeing memory with `delete` and `delete[]`
  - Memory leaks (forgetting to free memory)
  - Dangling pointers (accessing memory that is freed)
  - Double delete (deleting an object twice)

LEC-6D Preview-Information Hiding

- The `Time` type as a structure
  - Structures cannot enforce type invariants
  - Structures are tightly coupled with their implementation
- David Parnas and the theory of information hiding
  - Creating a public interface for the `Time` type
  - Adding member functions to the interface
  - Implementing member functions with the scope operator
- OOP Concepts
  - Objects: identity, state and behavior
  - Classes: specifications for a particular type
  - Encapsulation: hide the implementation behind the interface

Week 6 Homework Preview

- Week 6 HW due by 1pm July 24th (Mon) or 25th (Tue)
- H27 - C-String functions: `reverse()` & `findStr()`
- H28 - Dynamic Memory: the `FlexArray` type
- H29 - Information Hiding: `Time` & member functions

Programming Exam 6, 7 & Retakes

- Now – Programming Exam #6
  - 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 PE06 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 07 will start
- Come back by 4pm when PE Retakes will start