UCLA Samueli
Computer Science

CS31 Week 5 Discussion

Fall 2021, Section 1C
Mingyu Derek Ma mdma@ucla.edu

Thanks Muhao Chen, Rosa Garza for their shared content

https://derek.ma/cs31 for slides and other discussion materials

Reminder

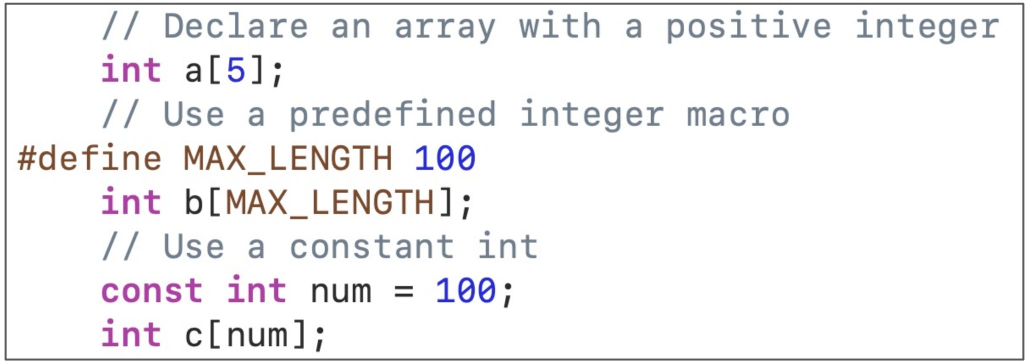
- Project 4, Nov 3, Wed 11pm
- Project 3 solution released

Project 4 Suggestions

- Read spec and FAQ thoroughly
- You don’t need to care about the case that query index is bigger than the number of elements in the array
    - Related description in the spec
- Develop incrementally
    - Implement a simple case, then expand to more conditions
- Save your immediate versions
    - You can compare and revert if something doesn’t work
    - You have something to turn in if you run out of time!

Declare an array

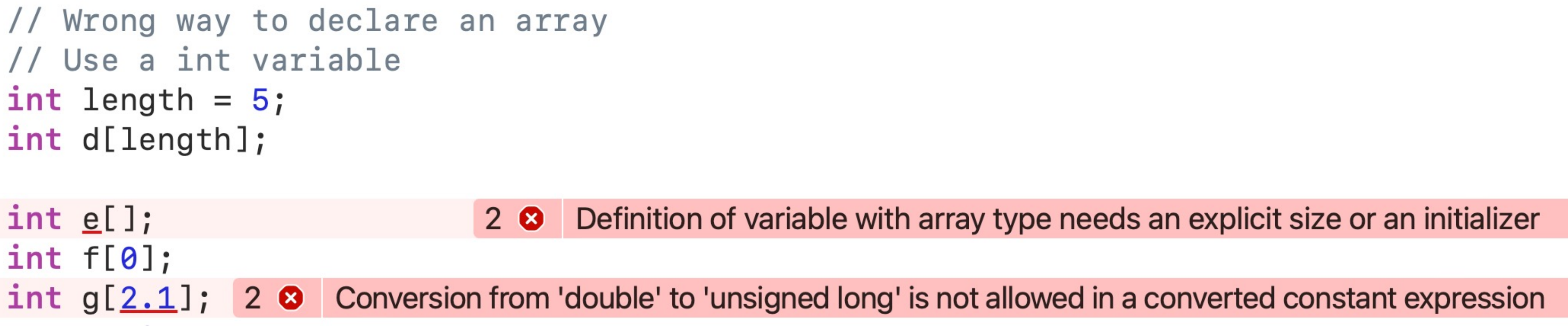
- type name [# of elements];
    - int a[5];
- # of elements can be
    - A positive number
    - A predefined integer macro
    - A constant int



```
    // Declare an array with a positive integer
    int a[5];
    // Use a predefined integer macro
#define MAX_LENGTH 100
    int b[MAX_LENGTH];
    // Use a constant int
    const int num = 100;
    int c[num];
```

Declare an array

- # of elements can NOT be
  - int variable
    - Not allowed in many compilers
  - Empty
  - 0
  - Float number



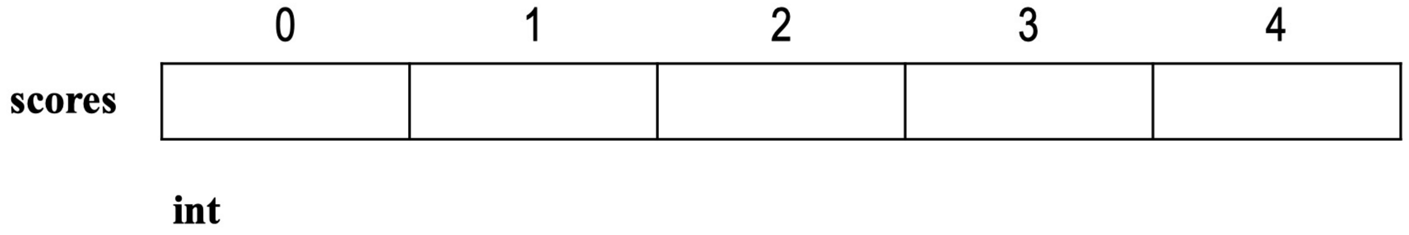
```
// Wrong way to declare an array
// Use a int variable
int length = 5;
int d[length];

int e[];        2 ✖ Definition of variable with array type needs an explicit size or an initializer
int f[0];
int g[2.1];  2 ✖ Conversion from 'double' to 'unsigned long' is not allowed in a converted constant expression
```

Array

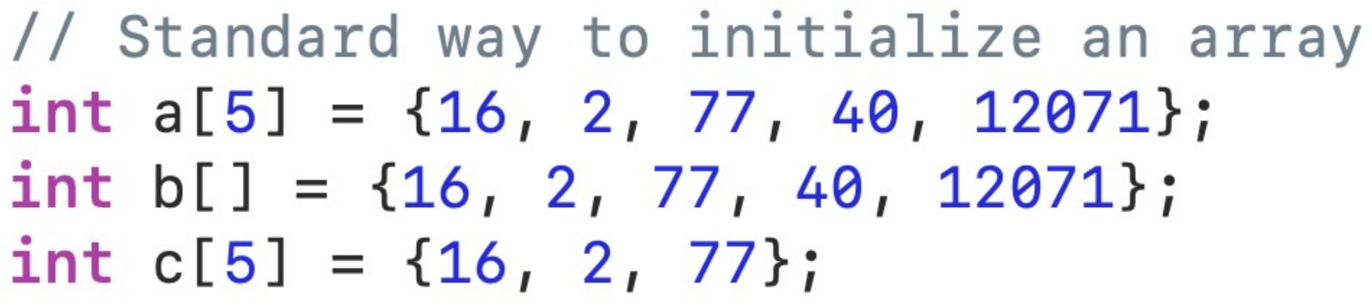
- Must know data type and size ahead of time
- Can only have one type of data
  - Cannot have an array of int and doubles etc
- No bounds checking!
  - The program will still run it we are trying to access out of bound position
  - Need to make sure we’re within valid bound
  - We need to specify a constant number of elements for an array
    - Some newer versions of compilers are removing this constraint
  - `sizeof` cannot provide the size of an array

An array starts from 0

- int scores[5];
- Elements are numbered/indexed from 0 to 4

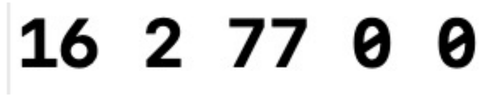


Initialize an array



```
// Standard way to initialize an array
int a[5] = {16, 2, 77, 40, 12071};
int b[] = {16, 2, 77, 40, 12071};
int c[5] = {16, 2, 77};
```

- If the number of initializing values is less than number of elements indicated, the rest will become all 0
- If we print out c:



```
16 2 77 0 0
```

Initialize an array

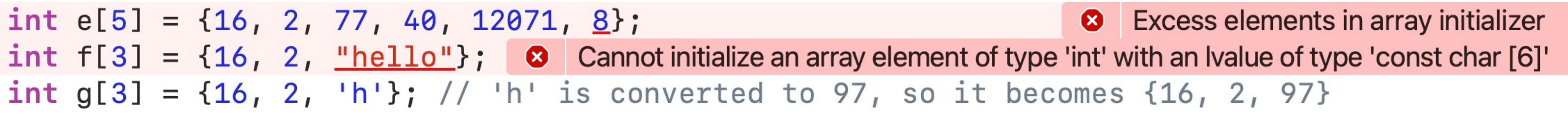
```
int d[4];
cout << d[0] << " " << d[1] << " " << d[2] << " " << d[3] << endl;
```

```
802832 1 -1074793384 32759
```

- Declare an array without initializing it
    - Values are undefined

Unacceptable array initialization

- Number of values > number of elements
- Inconsistent and unconvertible types



```
int e[5] = {16, 2, 77, 40, 12071, 8};          Excess elements in array initializer
int f[3] = {16, 2, "hello"};   Cannot initialize an array element of type 'int' with an lvalue of type 'const char [6]'
int g[3] = {16, 2, 'h'}; // 'h' is converted to 97, so it becomes {16, 2, 97}
```

Initialize an array with many elements

- How to initialize an array with 100 0s

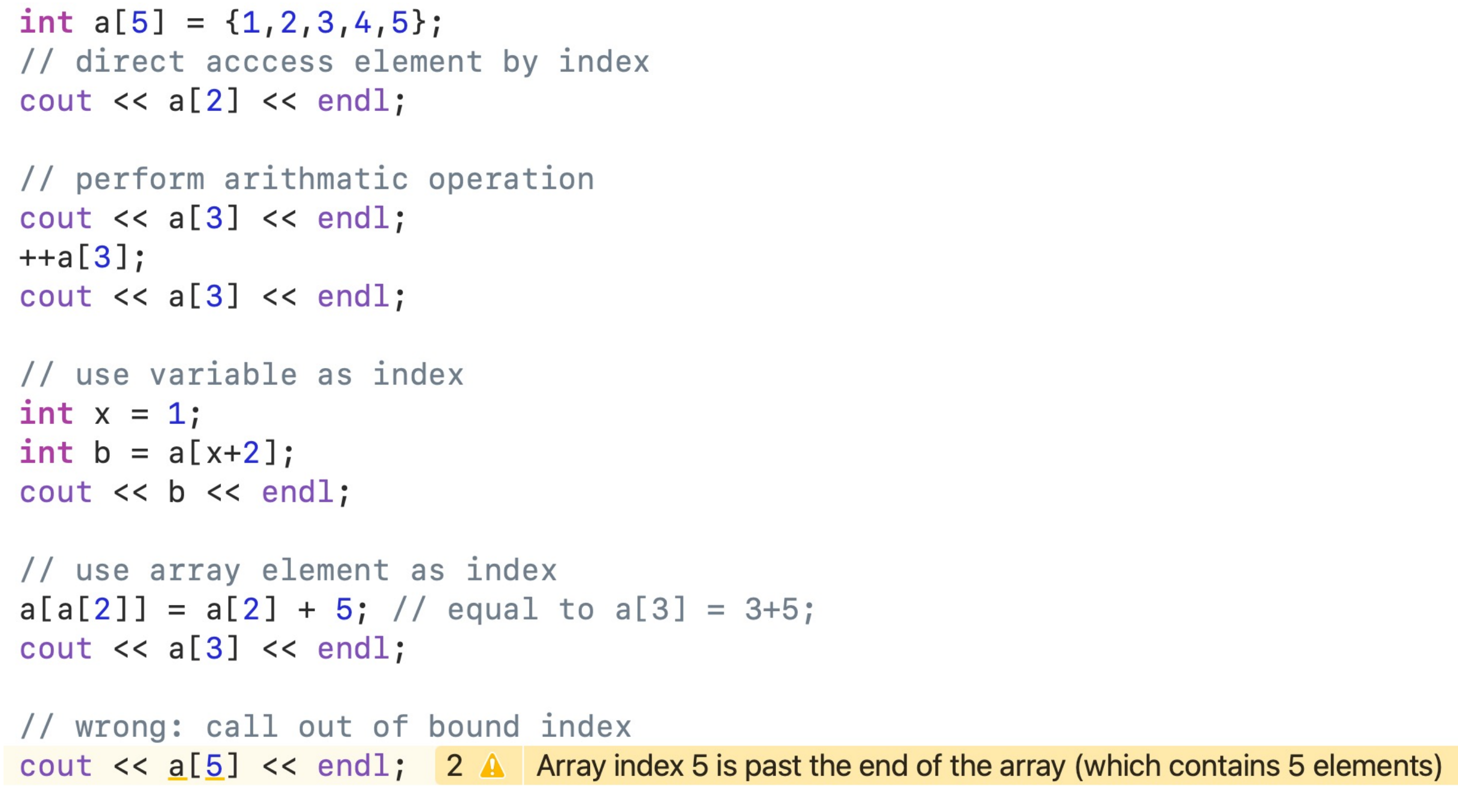
```
int a[100] = {0};
```

- How to initialize an array with 100 1s

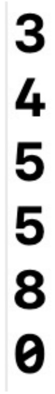
```
int a[100];
for(int i = 0; i < 100; ++i)
     a[i] = 1
```

Access elements of an array

- name[index]



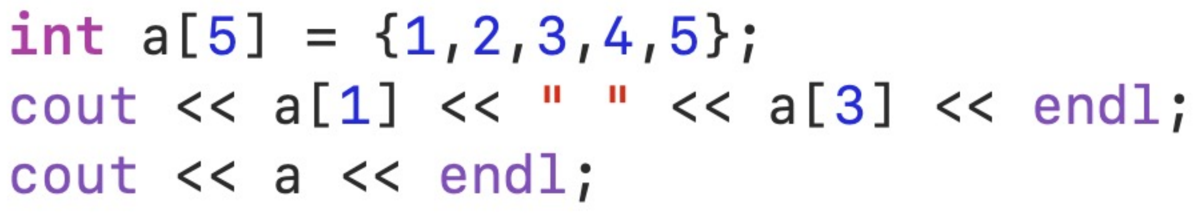
```
int a[5] = {1,2,3,4,5};
// direct acccess element by index
cout << a[2] << endl;

// perform arithmatic operation
cout << a[3] << endl;
++a[3];
cout << a[3] << endl;

// use variable as index
int x = 1;
int b = a[x+2];
cout << b << endl;

// use array element as index
a[a[2]] = a[2] + 5; // equal to a[3] = 3+5;
cout << a[3] << endl;

// wrong: call out of bound index
cout << a[5] << endl;   2 ⚠ Array index 5 is past the end of the array (which contains 5 elements)
```



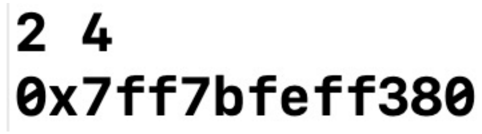
```
3
4
5
5
8
0
```

Print an array

- Print array elements
- If we print the array variable, we will get the starting memory address



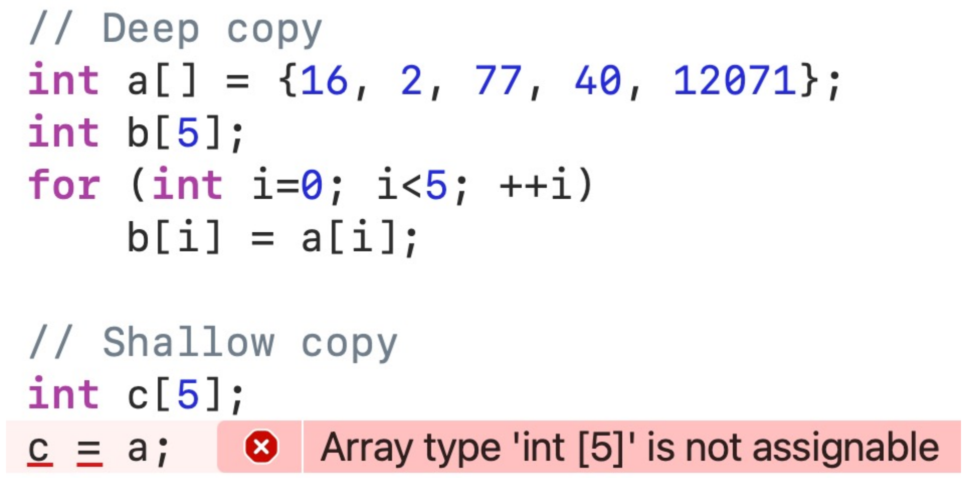
```
int a[5] = {1,2,3,4,5};
cout << a[1] << " " << a[3] << endl;
cout << a << endl;
```



```
2 4
0x7ff7bfeff380
```

Copy an array

- Deep copy: copy the content from an array to another
    - Copy it element by element
- Shallow copy: just make the new name and the old name have the same array, not allowed in some compilers



```
// Deep copy
int a[] = {16, 2, 77, 40, 12071};
int b[5];
for (int i=0; i<5; ++i)
    b[i] = a[i];

// Shallow copy
int c[5];
c = a;    Array type 'int [5]' is not assignable
```

Use arrays in a function

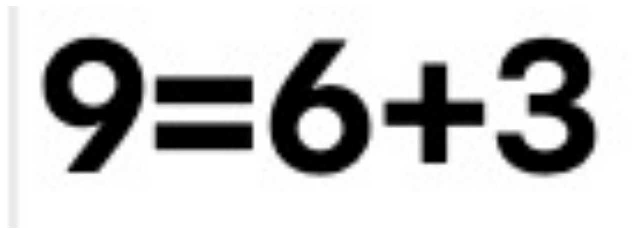
```
void print_array(int a[], int len){
        for (int i=0;i<len;i++)
                cout << "[" << i << "] = " << a[i] << endl;
}
int main(){
        int a[7] = {2, 0, 1, 2, 2, 2, 7};
        print_array(a, 7);
}
```

- Cannot add number of elements to an array argument
- A function will not know the length of an array unless you provide it
- Pass the name of the array to the function

Use arrays in a function

- An array as an argument is always an actual mutable parameter (pass by reference

```
void invert_array (char a[], int len) {
    for (int i=0; i < len / 2; ++i) {
        char tmp = a[i];
        a[i] = a[len - 1 - i];
        a[len - 1 - i] = tmp;
    }
}

int main(){
    char a[6] = {'3', '+', '6', '=', '9'};
    invert_array(a, 5);
    cout << a << endl;
}
```



```
9=6+3
```

Thank You