We are 183

L10: Week 6 - Wednesday

Reminders

- Assignment 3 due Friday
  - Announcement – Arrays due following Friday!
  - See the course website

- Exam 1 on Tuesday
  - Exam Review: Sunday 2/14 at 6pm in CHEM 1800

Last Time… on EECS 183

Pass by reference

Pass by Value

- What you're used to now
- Variables hold values
- Values are passed into functions
- **And** the original variable remains unchanged

Function Call – Pass by Value

```
int square(int a) {
    int sq = a * a;
    return sq;
}
```

Execution

```
int main(void) {
    int x = 3;
    int y;
    y = square(x);
    cout << y;
}
```

Function Call – Pass by Value

```
int square(int a) {
    int sq = a * a;
    return sq;
}
```

```
int main(void) {
    int x = 3;
    int y;
    y = square(x);
    cout << y;
}
```

Execution

Function Call – Pass by Value

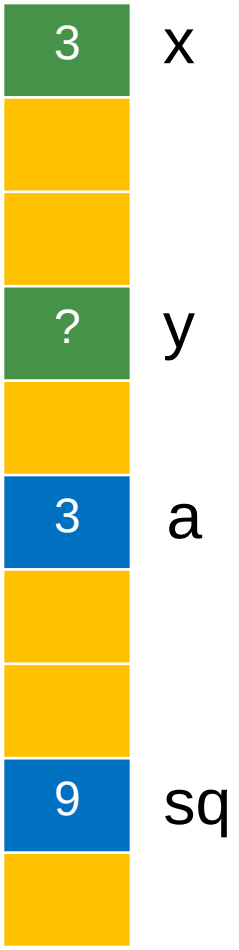
Execution

```
int square(int a) {
    int sq = a * a;
    return sq;
}



int main(void) {
    int x = 3;
    int y;
    y = square(x);
    cout << y;
}
```

Function Call – Pass by Value

Execution

```
int square(int a) {
    int sq = a * a;
    return sq;
}


int main(void) {
    int x = 3;
    int y;
    y = square(x);
    cout << y;
}
```

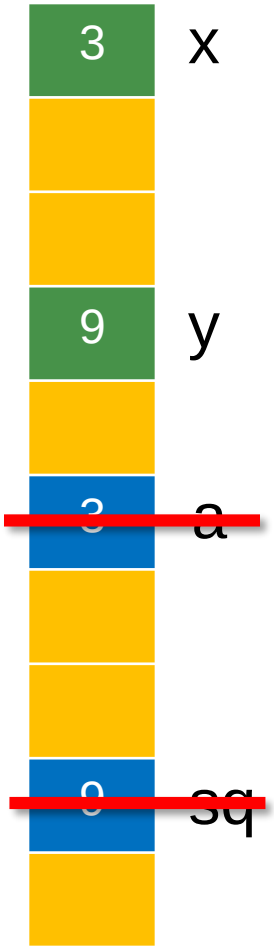


Function Call – Pass by Value

```
int square(int a) {
    int sq = a * a;
    return sq;
}

int main(void) {
    int x = 3;
    int y;
    y = square(x);
    cout << y;
}
```

Execution

9



Function Call – Pass by Value

```
int square(int a) {
    int sq = a * a;
    return sq;
}
```

```
int main(void) {
    int x = 3;
    int y;
    y = square(x);
    cout << y;
}
```

Execution

Console

9

Pass by Value

- Doesn't work well when:
  - Dealing with something very large in memory
  - We **want** to alter the original variable
- In these cases, use **Pass by Reference**

Pass by Reference

```
// Pass by Value
int squareByValue(int number) {
    return number * number;
}

// Pass by Reference
void squareByReference(int &number) {
    number *= number;
}
```

The **&** indicates *pass by reference*

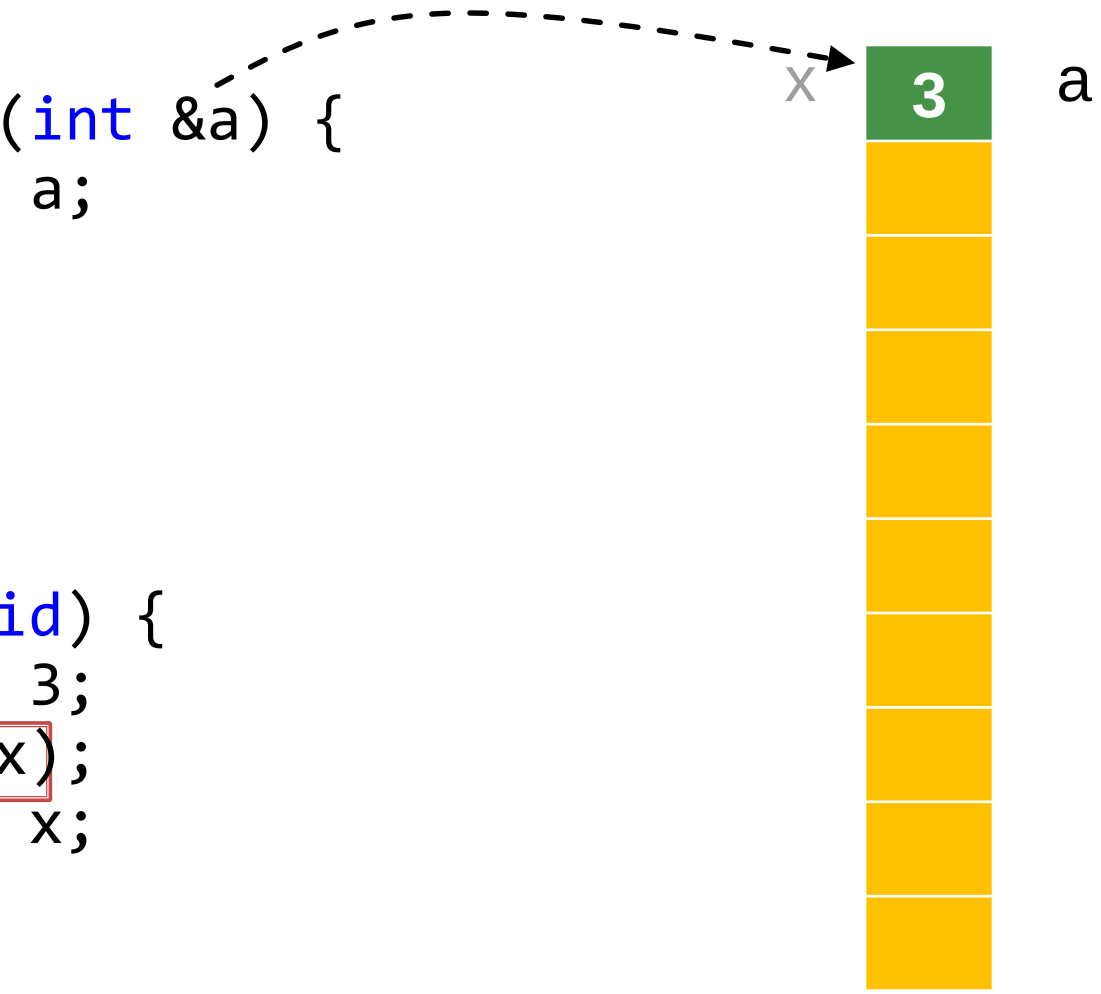
Pass by Reference

- '&' means ***address*** of variable is passed
  - address is the location in memory
- Modify variables directly

Function Call – Pass by Reference

Execution

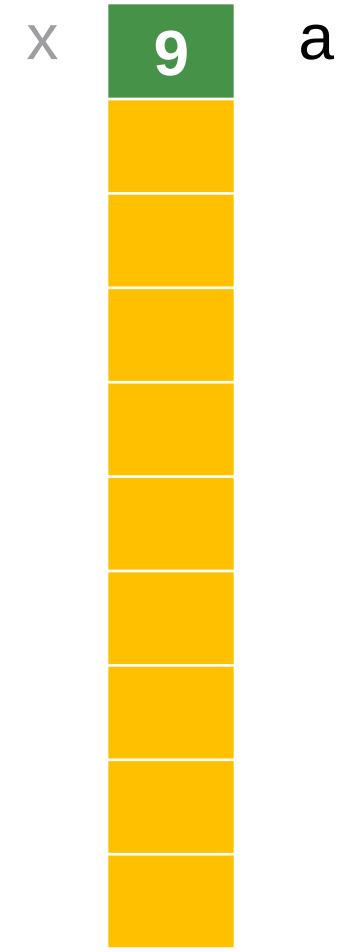
```
void square(int &a) {
    a = a * a;
}



int main(void) {
    int x = 3;
    square(x);
    cout << x;
}
```



Function Call – Pass by Reference

Execution

```
void square(int &a) {
    a = a * a;
}



int main(void) {
    int x = 3;
    square(x);
    cout << x;
}
```



Function Call – Pass by Reference

```
void square(int &a) {
    a = a * a;
}



int main(void) {
    int x = 3;
    square(x);
    cout << x;
}
```

Execution

x

Console

```
9
```

Function Parameters - Summary

Value Parameters

- Receives a copy
- Single value, or nothing returned
- Type casting is allowed
- Can be a variable, constant, or expression

Reference Parameters

- Receives a **reference** to the memory location
- **Multiple values** may be passed back *in effect*
- **Types must match** parameter declaration
- Must be named **storage**

i>Clicker #1

```
void increment(int& value) {
    value++;
}

int main(void) {
    int x = 0;

    increment(x);
    cout << x;
}
```

What gets printed?

A. 0
B. 1
C. 2
D. Error

i>Clicker #1

```
void increment(int& value) {
    value++;
}

int main(void) {
    int x = 0;

    increment(x);
    cout << x;
}
```

What gets printed?

A. 0
B. 1
C. 2
D. Error

i>Clicker #2

```
void increment(int& value1, int value2) {
    value1++;
    value2++;
}

int main(void) {
    int x = 0;
    int y = 0;
    increment(x, y);
    cout << x << y;
}
```

What gets printed?

A. 00
B. 01
C. 10
D. 11

i>Clicker #2

```
void increment(int& value1, int value2) {
    value1++;
    value2++;
}

int main(void) {
    int x = 0;
    int y = 0;
    increment(x, y);
    cout << x << y;
}
```

What gets printed?

A. 00
B. 01
C. 10
D. 11

i>Clicker #3

```
void increment(int& value1, int value2) {
    value1++;
    value2++;
    cout << value1 << value2;
}

int main(void) {
    int x = 0;

    increment(x, x);
}
```

What gets printed?

A. 11
B. 12
C. 21
D. 22

i>Clicker #3

```
void increment(int& value1, int value2) {
    value1++;
    value2++;
    cout << value1 << value2;
}

int main(void) {
    int x = 0;

    increment(x, x);
}
```

What gets printed?

A. 11
B. 12
C. 21
D. 22

Today

1-Dimensional arrays
Passing arrays to functions

Print grades > average

- Write a program that reads from standard input the grades of 5 students and prints all the grades that are higher than average.

- Sample run:

Console

```
60 70 80 90 100<enter>
90
100
```

Print grades > average

- **Pseudocode**
  1. Declare 5 variables
  2. Use 5 *cin* statements
  3. Compute average
  4. Use 5 *if* statements to print grades that are greater than avg

Print grades > average

Step 1

Step 2

Step 3

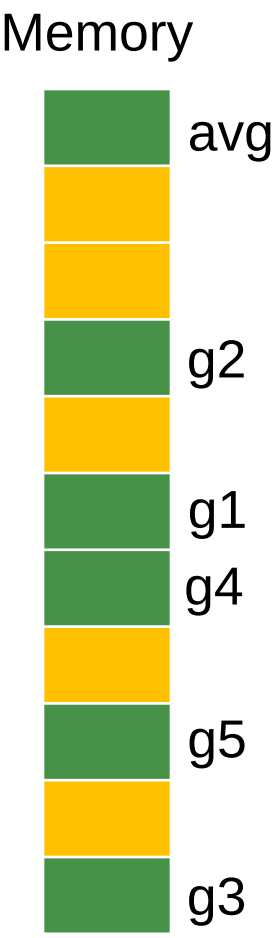
Step 4

Print grades > average

```
double g1, g2, g3, g4, g5;
cin >> g1 >> g2 >> g3 >> g4 >> g5;
double avg = (g1 + g2 + g3 + g4 + g5) / 5;
if (g1 > avg) { cout << g1 << endl; }
if (g2 > avg) { cout << g2 << endl; }
if (g3 > avg) { cout << g3 << endl; }
if (g4 > avg) { cout << g4 << endl; }
if (g5 > avg) { cout << g5 << endl; }
```

Print grades > average

```
double g1, g2, g3, g4, g5;
cin >> g1 >> g2 >> g3 >> g4 >> g5;
double avg = (g1 + g2 + g3 + g4 + g5) / 5;
if (g1 > avg) { cout << g1 << endl; }
if (g2 > avg) { cout << g2 << endl; }
if (g3 > avg) { cout << g3 << endl; }
if (g4 > avg) { cout << g4 << endl; }
if (g5 > avg) { cout << g5 << endl; }
```



Print grades > average

- What if we do this small change?
  - Write a program that reads from standard input the grades of **100** students and prints all the grades that are higher than average.

Print grades > average

- **Pseudocode**
  1. Declare 100 variables
  2. Use 100 *cin* statements
  3. Compute average
  4. Use 100 *if* statements to print grades that are > avg

1. Declare 100 variables

```
double g1, g2, g3, g4, g5, g6, g7, g8, g9, g10,
g11, g12, g13, g14, g15, g16, g17, g18, g19,
g20, g21, g22, g23, g24, g25, g26, g27, g28,
g29, g30, g31, g32, g33, g34, g35, g36, g37,
g38, g39, g40, g41, g42, g43, g44, g45, g46,
g47, g48, g49, g50, g51, g52, g53, g54, g55,
g56, g57, g58, g59, g60, g61, g62, g63, g64,
g65, g66, g67, g68, g69, g70, g71, g72, g73,
g74, g75, g76, g77, g78, g79, … continued on next slide
```

1. Declare 100 variables

```
… g80, g81, g82, g83, g84, g85, g86, g87, g88,
g89, g90, g91, g92, g93, g94, g95, g96, g97,
g98, g99, g100;
```

… continued on next slide

2. Read-in 100 grades

```
cin >> g1 >> g2 >> g3 >> g4 >> g5 >> g6 >> g7
>> g8 >> g9 >> g10 >> g11 >> g12 >> g13 >> g14
>> g15 >> g16 >> g17 >> g18 >> g19 >> g20 >>
g21 >> g22 >> g23 >> g24 >> g25 >> g26 >> g27
>> g28 >> g29 >> g30 >> g31 >> g32 >> g33 >>
g34 >> g35 >> g36 >> g37 … continued on next slide
```

2. Read-in 100 grades

```
… >> g38 >> g39 >> g40 >> g41 >> g42 >> g43 >>
g44 >> g45 >> g46 >> g47 >> g48 >> g49 >> g50
>> g51 >> g52 >> g53 >> g54 >> g55 >> g56 >>
g57 >> g58 >> g59 >> g60 >> g61 >> g62 >> g63
>> g64 >> g65 >> g66 >> g67 >> g68 >> g69 >>
g70 >> g71 >> g72 >> g73 >> g74 >> g75 >> g76
>> g77 >> g78 >> g79 >> g80 >> g81 >> g82 >>
g83 >> g84 >> g85 >> g86 >> g87 >> g88 >> g89
>> g90 >> g91 >> g92 >> … continued on next slide
```

2. Read-in 100 grades

```
… g93 >> g94 >> g95 >> g96 >> g97 >> g98 >> g99
>> g100;
```

… continued on next slide

3. 100 *if* statements

```
if (g1 > avg) { cout << g1 << endl; }
if (g2 > avg) { cout << g2 << endl; }
if (g3 > avg) { cout << g3 << endl; }
if (g4 > avg) { cout << g4 << endl; }
…

…
if (g100 > avg) { cout << g100 << endl; }
```

3. 100 *if* statements

```
if (g1 > avg) { cout << g1 << endl; }
if (g2 > avg) { cout << g2 << endl; }
if (g3 > avg) { cout << g3 << endl; }
if (g4 > avg) { cout << g4 << endl; }
```

There must be a more efficient way to do it!

```
if (g100 > avg) { cout << g100 << endl; }
```

Arrays

An **array** is a systematic arrangement of objects

--Wikipedia

Allen Telescope Array (ATA)

Wind Turbines

Arrays

An **array** is a collection of items of the **same datatype** stored **consecutively** in memory under **one name**.

Arrays

An **array** is a collection of items of the **same datatype** stored **consecutively** in memory under **one name**.

What other **collection** have we seen in C++?

Arrays

An **array** is a collection of items of the **same datatype** stored **consecutively** in memory under **one name**.

What other **collection** have we seen in C++?

`string`

Arrays

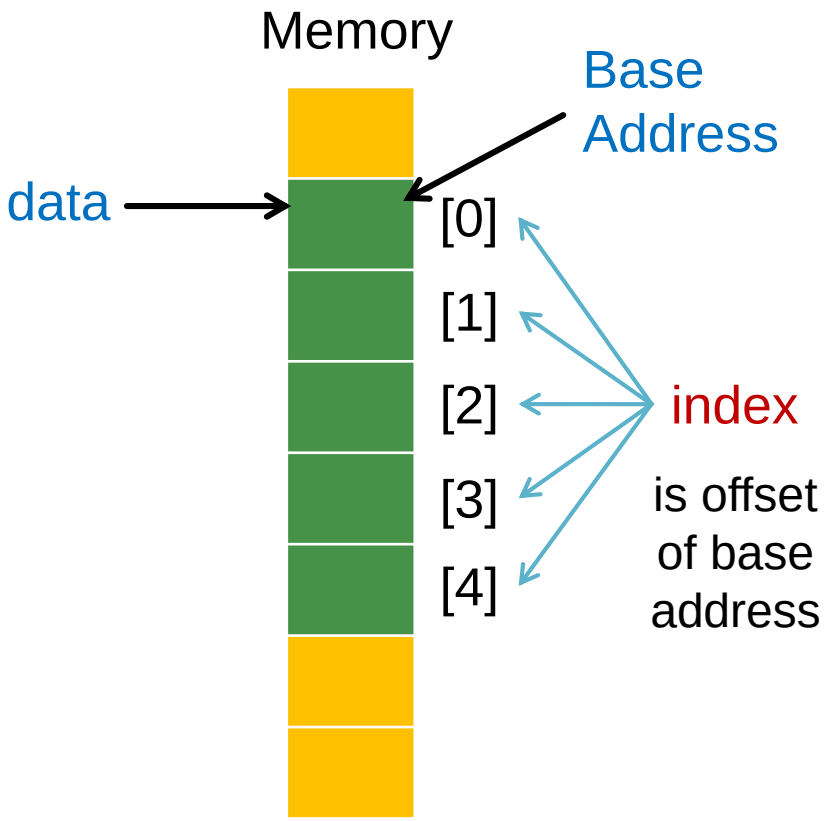
- Excellent choice for storing and processing a **sequence of values**

```
1, 2, 3, 4, …

1.2, 7.4, -5.1, 99.3, …

"Hello World!"

true, false, true, true, …
```

Array Declaration

type name (# of elements) size

```
double data[5];

// Allocates 5 consecutive
// memory locations suitable
// for storing double values
```



Static Allocation

- Array size must be known at compile time
- Array size must be a **constant integer value**

Static Allocation

- Array size must be known at compile time
- Array size must be a **constant integer value**

```
int data[5];
```

Static Allocation

- Array size must be known at compile time
- Array size must be a **constant integer value**

```
int data[5];
```

```
const int SIZE = 5;
int data[SIZE];
```

Static Allocation

- Array size must be known at compile time
- Array size must be a **constant integer value**

```
int data[5];
```

```
const int SIZE = 5;
int data[SIZE];
```

```
int size = 5;
int data[size];
```

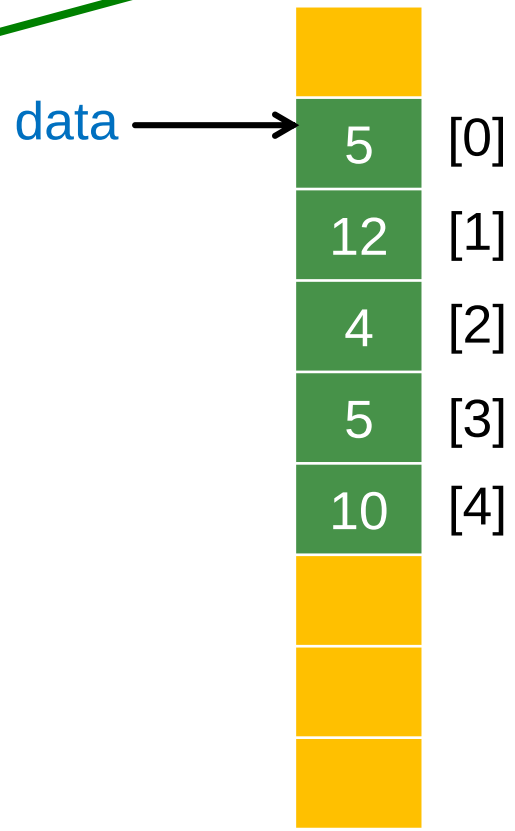
Compile error

Cannot use a variable

Array Initialization

```
int data[5] = {5, 12, 4, 5, 10};
```

Braces can be used to initialize array values.

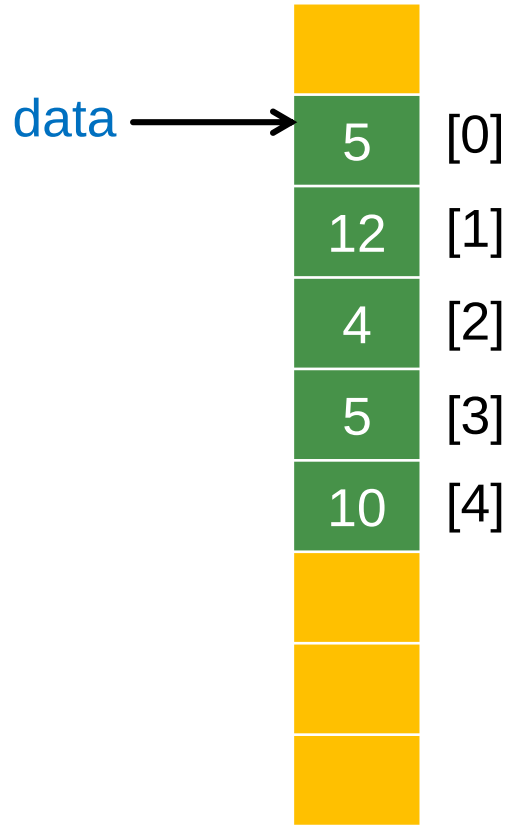


Array Initialization

```
int data[ ] = {5, 12, 4, 5, 10};
```

Size can be omitted in this case

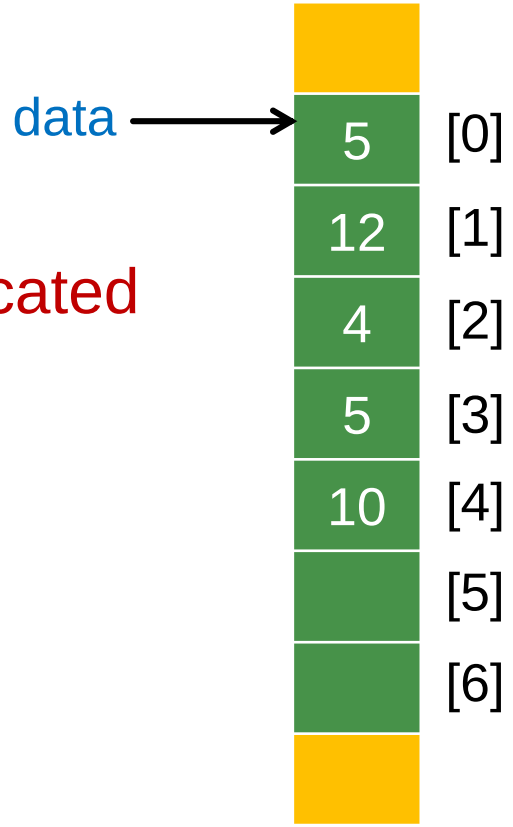
5 memory locations will be allocated by the compiler



Array Initialization

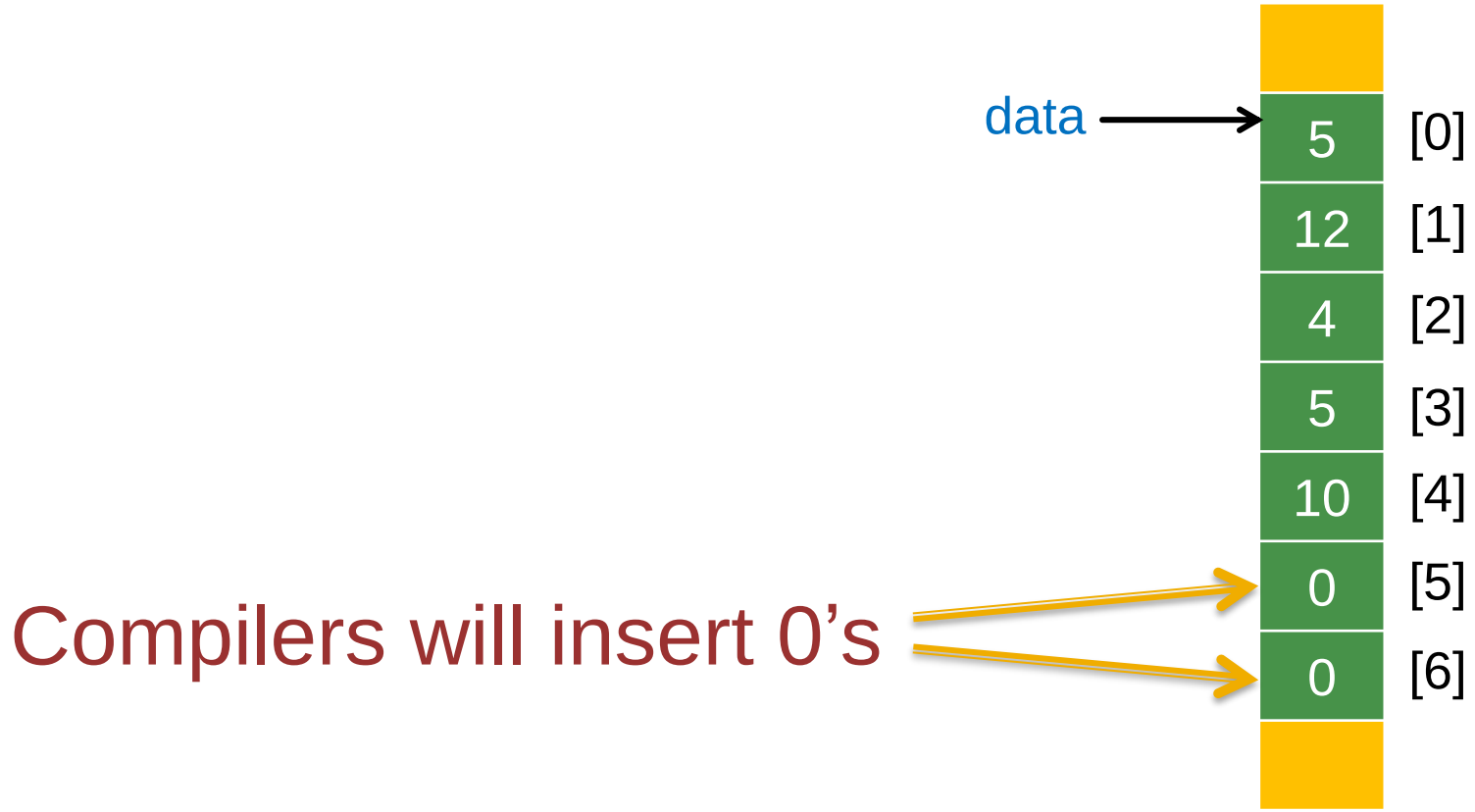
```
int data[7] = {5, 12, 4, 5, 10};
```

More memory locations can be allocated



Array Initialization

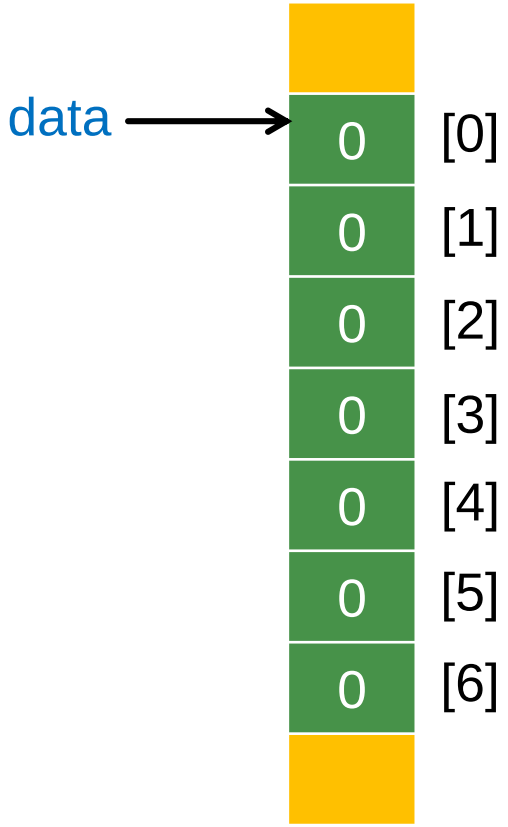
```
int data[7] = {5, 12, 4, 5, 10};
```



Array Initialization

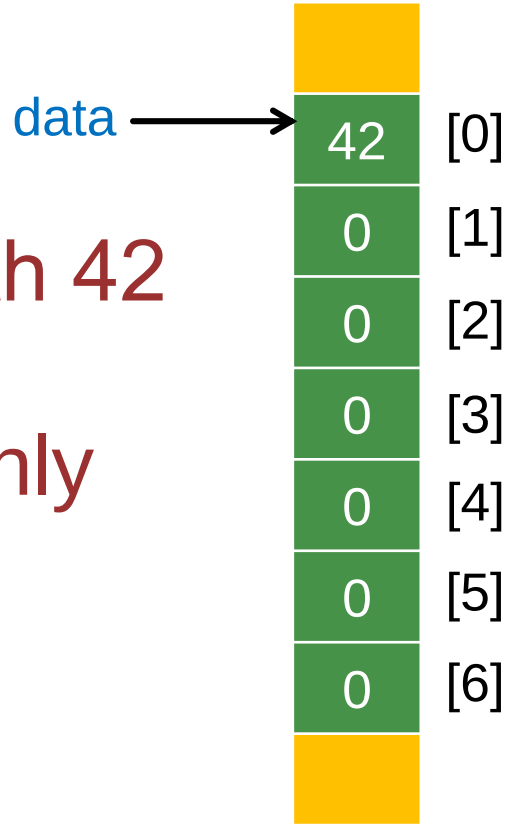
```
int data[7] = { };
```

If you put braces in the definition,
compiler fill with 0's



Array Initialization

```
int data[7] = {42};
```



Don’t expect it to load all with 42

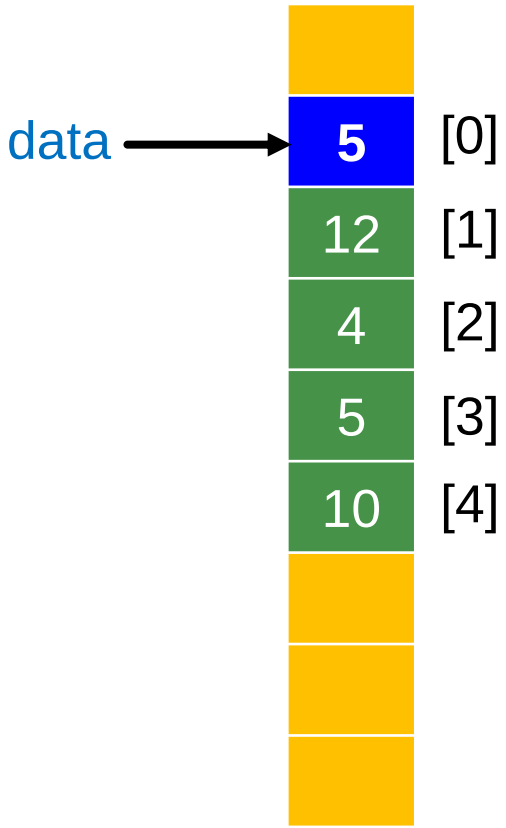
Puts 42 into 1st element only

0’s in rest

Accessing array elements

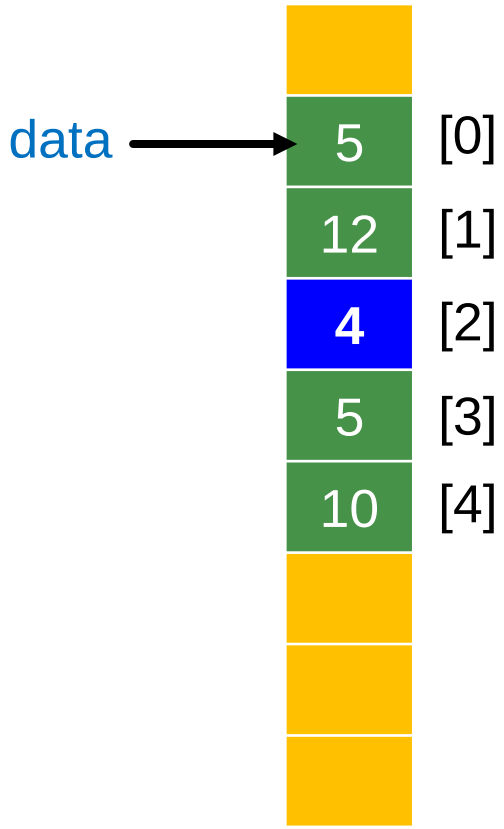
```
cout << data[0];
// prints 5
```

Bracket access, just like a string!



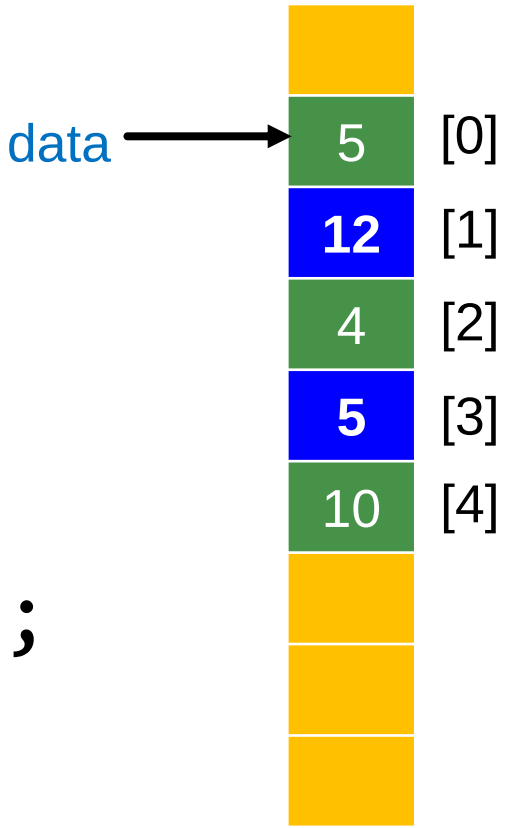
Accessing array elements

```
cout << data[0];
// prints 5

cout << data[2];
// prints 4
```



Accessing array elements

```
cout << data[0];
// prints 5

cout << data[2];
// prints 4

cout << data[1] + data[3];
// prints 17
```



Accessing array elements

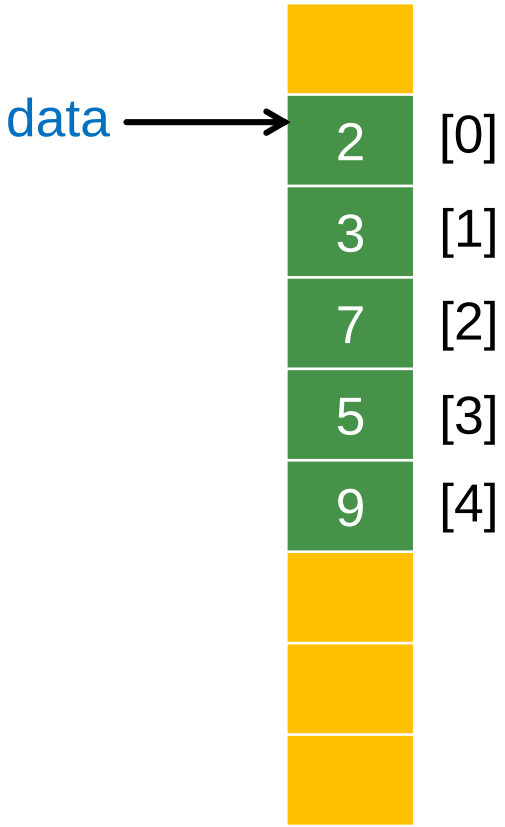
Execution

```
int data[] = {2, 3, 7, 5, 9};

data[2] = -1;

data[3] = data[4];

data[1]++;
```

Accessing array elements

Execution

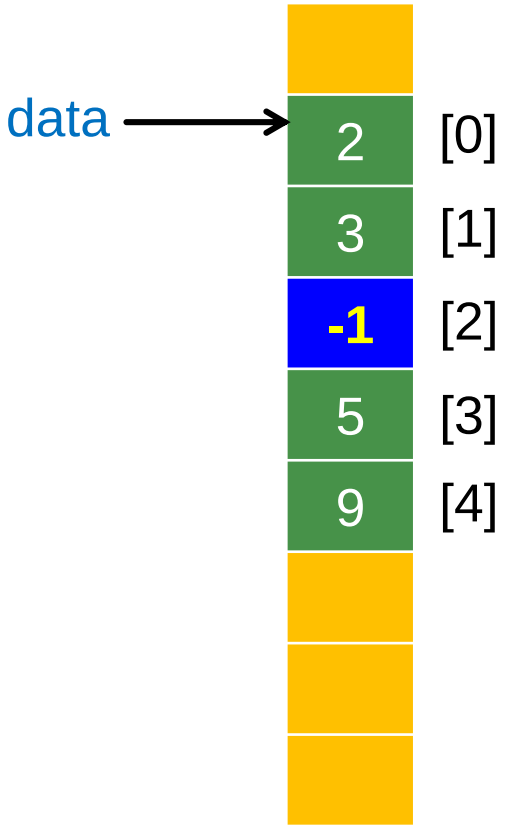
```
int data[] = {2, 3, 7, 5, 9};

data[2] = -1;

data[3] = data[4];

data[1]++;
```



Accessing array elements

```
int data[] = {2, 3, 7, 5, 9};

data[2] = -1;

data[3] = data[4];

data[1]++;
```

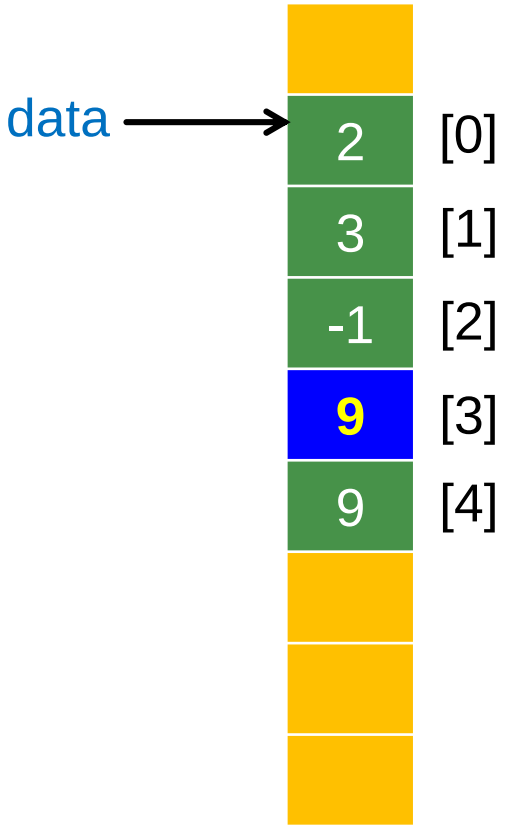
Execution



Accessing array elements

```
int data[] = {2, 3, 7, 5, 9};

data[2] = -1;

data[3] = data[4];

data[1]++;
```

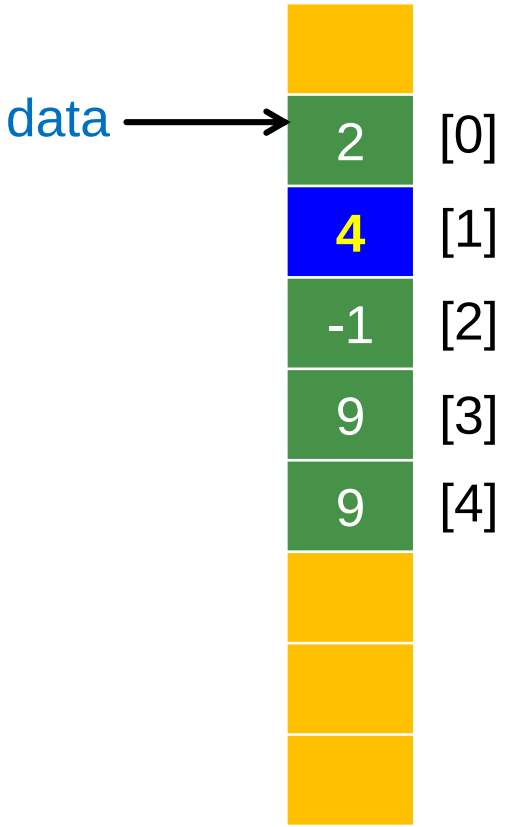
Execution



Accessing array elements

```
int data[] = {2, 3, 7, 5, 9};

data[2] = -1;

data[3] = data[4];

data[1]++;
```

Execution



Creating Arrays with a Size

- **Can only use a constant for size**
- **Cannot use a variable**

```
int size;
cout << "Enter a size";
cin >> size;

int data[size];
```

Compiler Error!!!

i>Clicker #4

Which of the following declarations will not compile?

```
A. int a[100];

B. const int N = 100;
   int a[N];

C. const int N = 100;
   int a[N+1];

D. int a[-100];
```

i>Clicker #4

Which of the following declarations will not compile?

A.
```
int a[100];
```

B.
```
const int N = 100;
int a[N];
```

C.
```
const int N = 100;
int a[N+1];
```

D.
```
int a[-100];
```

i>Clicker #5

Which of the following are invalid array declarations?

```
A. int a[] = {1,2,3};

B. int a[4] = {1};

C. int a[3] = {1,2,3};

D. int a[3] = {0,1,2,3};

E. Both b and d are invalid
```

i>Clicker #5

Which of the following are invalid array declarations?

```
A. int a[] = {1,2,3};

B. int a[4] = {1};

C. int a[3] = {1,2,3};

D. int a[3] = {0,1,2,3};

E. Both b and d are invalid
```

Print grades > average

- Write a program that reads from standard input the grades of **100** students and prints all the grades that are higher than average.

Can we do it more efficiently now?

Print grades > average

- **Pseudocode**
  1. Declare 100 variables
  2. Use 100 *cin* statements
  3. Compute average
  4. Use 100 *if* statements to print grades that are > avg

Print grades > average

- **Pseudocode**
  1. ~~Declare 100 variables~~

     Declare an array of size 100
  2. Use 100 *cin* statements
  3. Compute average
  4. Use 100 *if* statements to print grades that are > avg

Print grades > average

- **Pseudocode**
  1. ~~Declare 100 variables~~

     Declare an array of size 100
  2. ~~Use 100 *cin* statements~~

     Use a loop (1 **cin** in its body) to read in values
  3. Compute average
  4. Use 100 *if* statements to print grades that are > avg

Print grades > average

- **Pseudocode**
  1. ~~Declare 100 variables~~

     Declare an array of size 100
  2. ~~Use 100 *cin* statements~~

     Use a loop (1 **cin** in its body) to read in values
  3. Compute average ✓
  4. Use 100 *if* statements to print grades that are > avg

Print grades > average

- **Pseudocode**
  1. ~~Declare 100 variables~~

     Declare an array of size 100
  2. ~~Use 100 *cin* statements~~

     Use a loop (1 **cin** in its body) to read in values
  3. Compute average ✓
  4. ~~Use 100 *if* statements to print grades that are > avg~~

     Use a loop to check each grade (array element)

Step 1: Declare an array of size 100

Step 2: Use a loop to read in values

Step 3: Compute average

Step 4: Use a loop to check each grade

```
const int SIZE = 100; double grades[SIZE];
```

Step 2: Use a loop to read in values

Step 3: Compute average

Step 4: Use a loop to check each grade

```
const int SIZE = 100; double grades[SIZE];
```

```
double total = 0;
for (int i = 0; i < SIZE; i++) {
    cin >> grades[i];
    total += grades[i];
}
```

Step 3: Compute average

Step 4: Use a loop to check each grade

```
const int SIZE = 100; double grades[SIZE];
```

```
double total = 0;
for (int i = 0; i < SIZE; i++) {
    cin >> grades[i];
    total += grades[i];
}
```

```
double average = total / SIZE;
```

Step 4: Use a loop to check each grade

```
const int SIZE = 100; double grades[SIZE];
```

```
double total = 0;
for (int i = 0; i < SIZE; i++) {
    cin >> grades[i];
    total += grades[i];
}
```

```
double average = total / SIZE;
```

```
for (int i = 0; i < SIZE; i++) {
    if (grades[i] > average) {
        cout << grades[i] << endl;
    }
}
```

i>Clicker #6

What are the contents of the array `arr` after executing the following code?

```
const int SIZE = 1;
int arr[SIZE];
for (int i = 1; i <= SIZE; i++)
    arr[i] = i + 1;
```

A. {1, 2}
B. {1}
C. {2}
D. Indeterminate (depends on the compiler)
E. The code does not compile

i>Clicker #6

What are the contents of the array `arr` after executing the following code?

```
const int SIZE = 1;
int arr[SIZE];
for (int i = 1; i <= SIZE; i++)
    arr[i] = i + 1;
```

A. {1, 2}
B. {1}
C. {2}
D. Indeterminate (depends on the compiler)
E. The code does not compile

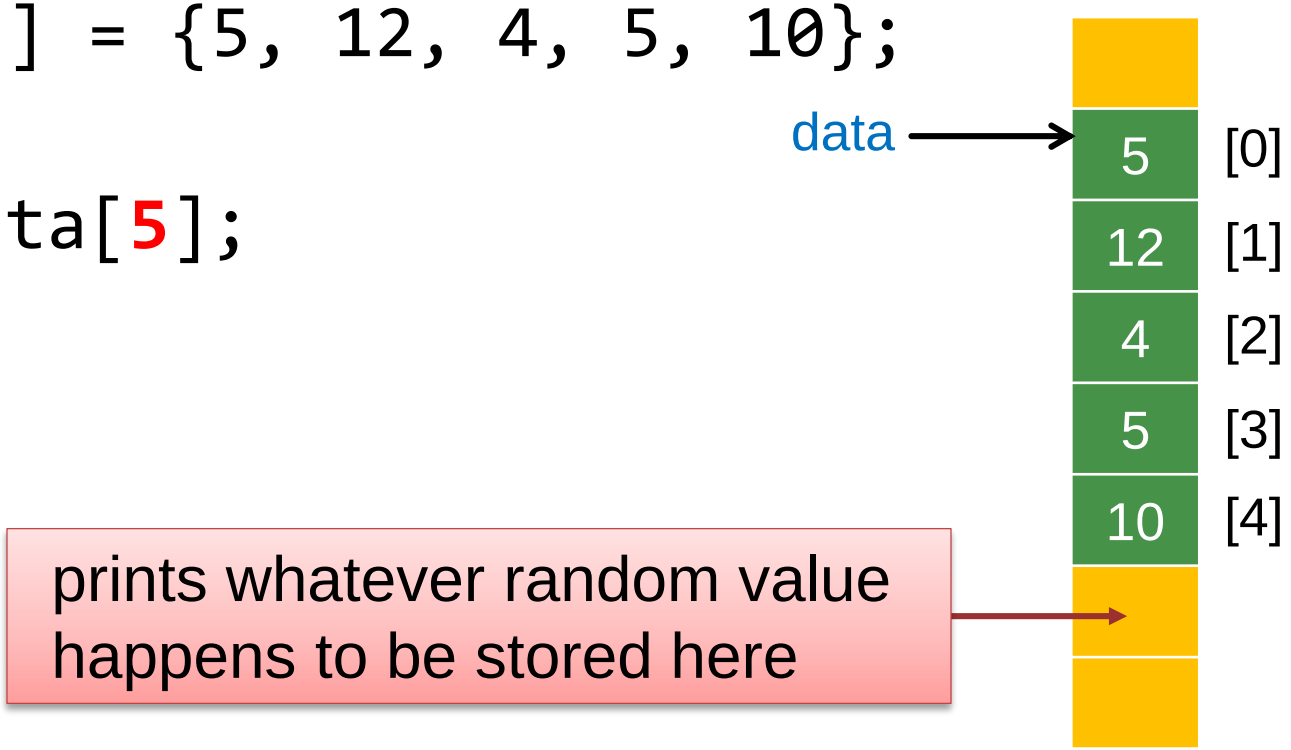
Goes **outside bounds** of arr**!!!**

Common Errors

1- Range Errors

- C++ does **NOT** check array indices for validity

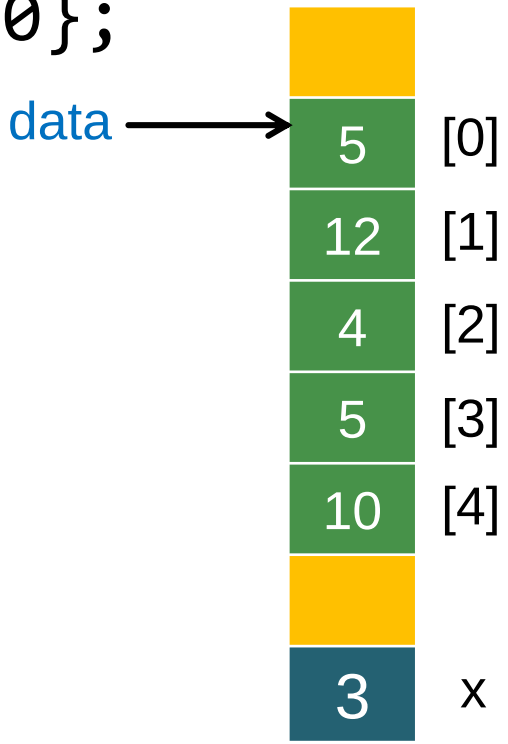
```
int data[5] = {5, 12, 4, 5, 10};

cout << data[5];
```



1- Range Errors

- Range errors can: overwrite data

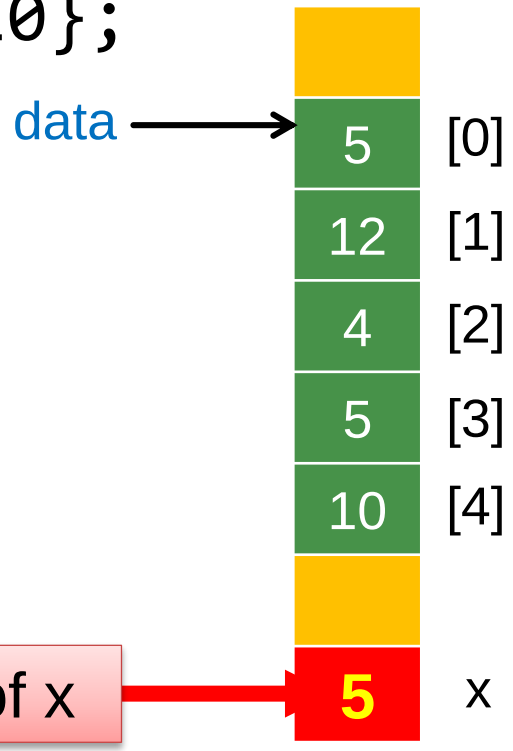
```
int data[5] = {5, 12, 4, 5, 10};

int x = 3;

data[6] = 5;
```



1- Range Errors

- Range errors can: overwrite data

```
int data[5] = {5, 12, 4, 5, 10};

int x = 3;

data[6] = 5;
```



Can **overwrite** the value of x

2- OFF-BY-ONE Error

- C++ does **NOT** check array indices for validity

```
const int SIZE = 10;

int data[SIZE];

for (int i = 0; i <= SIZE; i++) {
    cin >> data[i];
}
```

Array index range: 0 → **9**

i : 0 → **10**

3- Operations on Arrays

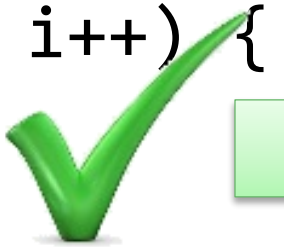
- I/O errors

```
const int SIZE = 10;

int data[SIZE];

cin >> data;
```

3- Operations on Arrays

- I/O errors

```
const int SIZE = 10;

int data[SIZE];

for (int i = 0; i < SIZE; i++) {
    cin >> data[i];
}
```



Use a loop

3- Operations on Arrays

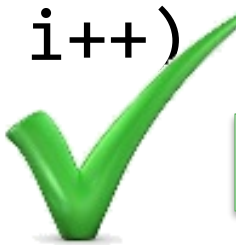
- Assignment errors

```
const int SIZE = 10;

int arr1[SIZE];
int arr2[SIZE];

...

arr2 = arr1;
```

3- Operations on Arrays

- Assignment errors

```
const int SIZE = 10;

int arr1[SIZE];
int arr2[SIZE];

for (int i = 0; i < SIZE; i++) {
    arr2[i] = arr1[i];
}
```



Use a loop

3- Operations on Arrays

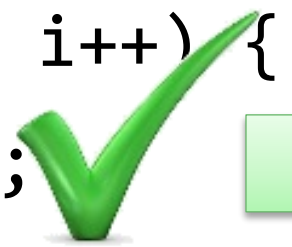
- Arithmetic Operations on Arrays

```
const int SIZE = 10;

int arr1[SIZE];
int arr2[SIZE];

arr1 = arr2 + 6;
```

3- Operations on Arrays

- Arithmetic Operations on Arrays

```
const int SIZE = 10;

int arr1[SIZE];
int arr2[SIZE];

for (int i = 0; i < SIZE; i++) {
    arr1[i] = arr2[i] + 6;
}
```



Use a loop

4- Comparison of Arrays

- Comparing Elements of Arrays

```
const int SIZE = 10;

int arr1[SIZE];
int arr2[SIZE];

if (arr1 < arr2) {
    // Do something
}
```

4- Comparison of Arrays

- Comparing Elements of Arrays

```
const int SIZE = 10;

int arr1[SIZE];
int arr2[SIZE];

if (arr1 < arr2) {
    // Do something
}
```

Bad Idea

Compares ***addresses*** of **arr1** and **arr2**

4- Comparison of Arrays

- Comparing Elements of Arrays

```
const int SIZE = 10;

int arr1[SIZE];
int arr2[SIZE];

for (int i = 0; i < SIZE; i++) {
    if (arr1[i] < arr2[i]) {
        // Do something
    }
}
```

Use a loop

4- Comparison of Arrays

- Comparing Elements of Arrays

```
const int SIZE = 10;

int arr1[SIZE];
int arr2[SIZE];

if (arr1 == arr2) {
    // Do something
}
```

4- Comparison of Arrays

- Comparing Elements of Arrays

```
const int SIZE = 10;

int arr1[SIZE];
int arr2[SIZE];

if (arr1 == arr2) {
    // Do something
}
```

Bad Idea

Compares ***addresses*** of **arr1** and **arr2**

4- Comparison of Arrays

- Comparing Elements of Arrays

```
const int SIZE = 10;

int arr1[SIZE];
int arr2[SIZE];

for (int i = 0; i < SIZE; i++) {
    if (arr1[i] == arr2[i]) {
        // Do something
    }
}
```

Use a loop

Common Solution!

Use a loop!

It's best to deal with array elements **individually**.

Passing Arrays to Functions

Example: Print Array

- Write a function that accepts an array of integers and prints the content of the array
- Prototype:

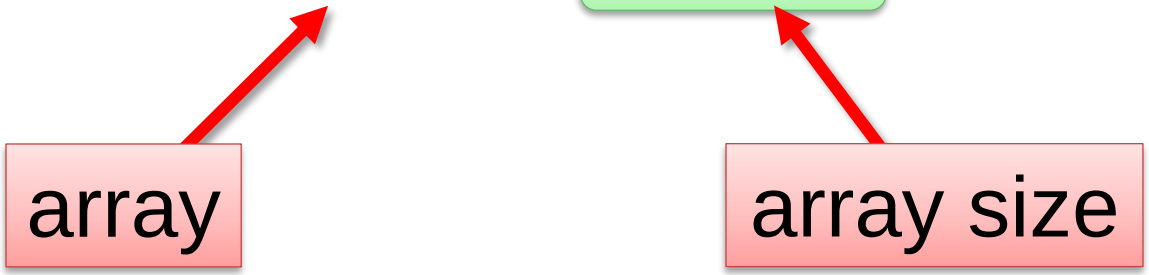
```
          printArray(                                  );
```

Example: Print Array

- Write a function that accepts an array of integers and prints the content of the array

- Prototype:

```
void printArray(int arr[], 2 );
```



Example: Print Array

- Write a function that accepts an array of integers and prints the content of the array

- Prototype:

```
void printArray(int arr[], int size);
```

Always passed by reference.
No **&** needed

Example: Print Array

Test:

```
int main() {
    // test printArray
    const int SIZE = 5;
    int data[SIZE] = {1, 2, 3, 4, 5};
    printArray(data, SIZE);
}
```

Example: Print Array

Implementation:

```
void printArray(int arr[], int size) {

}
```

Example: Print Array

Implementation:

```
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << endl;
    }
}
```

Passing Arrays to functions

C++ arrays are:

- **Always** passed by **reference**
  - No **&** needed
- Saves memory space
  - array is not copied
- Saves processing time
  - array elements are not copied

i>Clicker #7

```
const int SIZE = 5;

void increment(int a[], int size) {
    for (int i = 0; i < size; i++) {
        a[i] = a[i] + 1;
    }
}

int main() {
    int a[SIZE] = {1, 2, 3, 4, 5};
    cout << a[1] << " ";
    increment(a, SIZE);
    cout << a[1];
}
```

What prints?

A. 1 1
B. 2 2
C. 2 3
D. 3 3
E. Error

i>Clicker #7

```
const int SIZE = 5;

void increment(int a[], int siz
    for (int i = 0; i < size; i++) {
        a[i] = a[i] + 1;
    }
}

int main() {
    int a[SIZE] = {1, 2, 3, 4, 5};
    cout << a[1] << " ";
    increment(a, SIZE);
    cout << a[1];
}
```

Arrays are **ALWAYS** passed by reference

What prints?

A. 1 1
B. 2 2
C. 2 3
D. 3 3
E. Error

Example: Square Array Elements

- Write a function that accepts an array of integers and squares all the elements of the array.

```
void squareArray(int arr[], int size);
```

Arrays are passed by reference – modified directly

Example: Square Array Elements

- Write a function that accepts an array of integers and squares all the elements of the array.

```
int[] squareArray(int arr[], int size);
```

Compile Error – Cannot return an array

Example: Square Array Elements

```
int main() {
    // test squareArray
    const int SIZE = 5;
    int data[SIZE] = {1, 2, 3, 4, 5}
    squareArray(data, SIZE);
    // array contents become 1, 4, 9, 16, 25
}
```

Example: Square Array Elements

```
void squareArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        arr[i] = arr[i] * arr[i];
    }
}
```

the Name of an Array ...

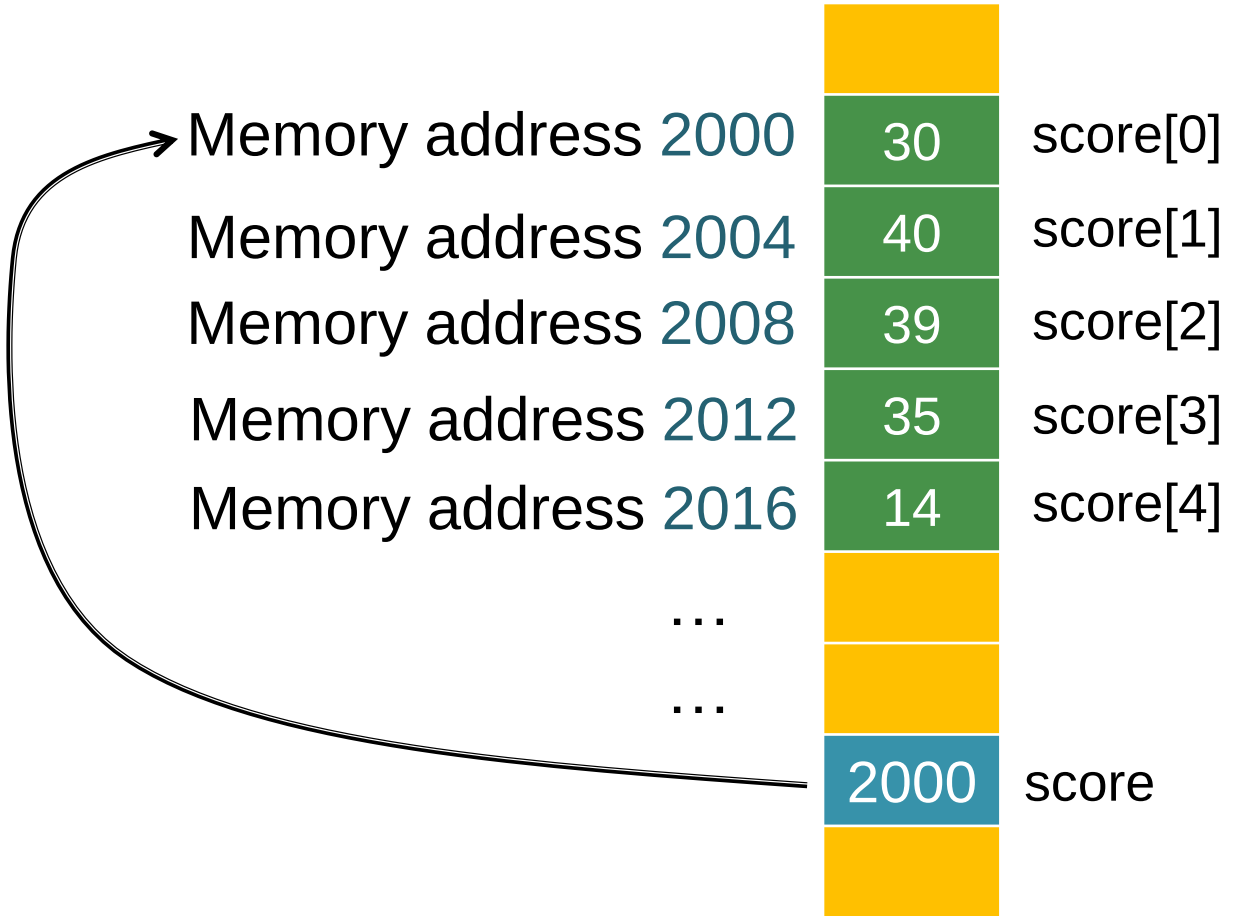
- Refers to the ***entire*** data structure

```
int score[5] = {30, 40, 39, 35, 14};
```

- **`score`** holds the memory address of the 1st array element (`score[0]`)

the Name of an Array ...

```
int score[5] = {30, 40, 39, 35, 14};
```



Prevent function from modifying an array

```
// allows function to alter data
// all arrays are passed by reference
void printResults(int data[ ], int size);


// With const, function can't alter array
void printResults(const int data[ ], int size);
```

i>Clicker #8

```
void increment(int x) {
    x = x + 1;
}

int main() {
    const int SIZE = 3;
    int arr[SIZE] = {1, 2, 3};
    cout << arr[1] << " ";
    increment(arr[1]);
    cout << arr[1];
    return 0;
}
```

What prints?

A. 1 1
B. 2 2
C. 2 3
D. 3 3
E. Error

i>Clicker #8

```
void increment(int x) {
    x = x + 1;
}

int main() {
    const int SIZE = 3;
    int arr[SIZE] = {1, 2, 3};
    cout << arr[1] << " ";
    increment(arr[1]);
    cout << arr[1];
    return 0;
}
```

What prints?

A. 1 1
B. 2 2
C. 2 3
D. 3 3
E. Error

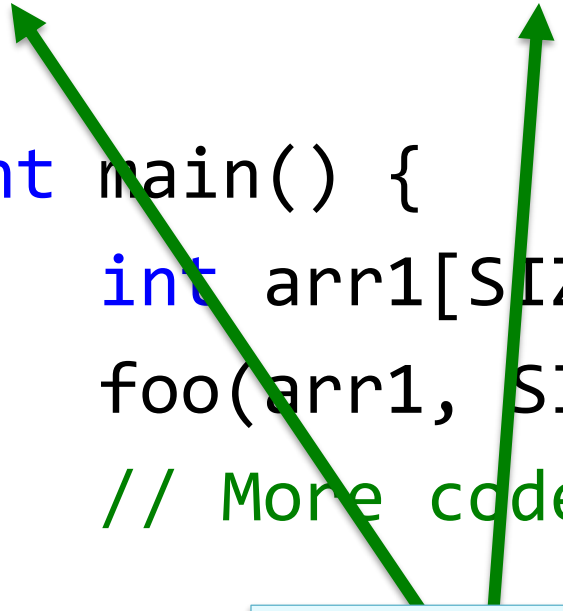
A single array element is ***also*** passed by value

Return from a function

```
const int SIZE = 5;
void foo(int arr[], int size);

int main() {
    int arr1[SIZE];
    foo(arr1, SIZE);
    // More code...
}
```

Return from a function

```
const int SIZE = 5;
void foo(int arr[], int size);

int main() {
    int arr1[SIZE];
    foo(arr1, SIZE);
    // More code...
}
```



Passed by **reference**
no need to "return" the array

Calls & Prototypes

Example Call (assume main is caller):

```
readScoreList(data, SIZE);
```

Possible Corresponding Prototypes:

```
void readScoreList(int data[SIZE], int size);
void readScoreList(int data[ ], int size);
void readScoreList(int * data, int size);
```

Example

on your own

Example

- Read in values from keyboard
  - stop reading when a `0` is entered
- Find the largest value
- Print out all values with largest marked

  32
  54
  67.5 ← largest
  29
  35

int main()

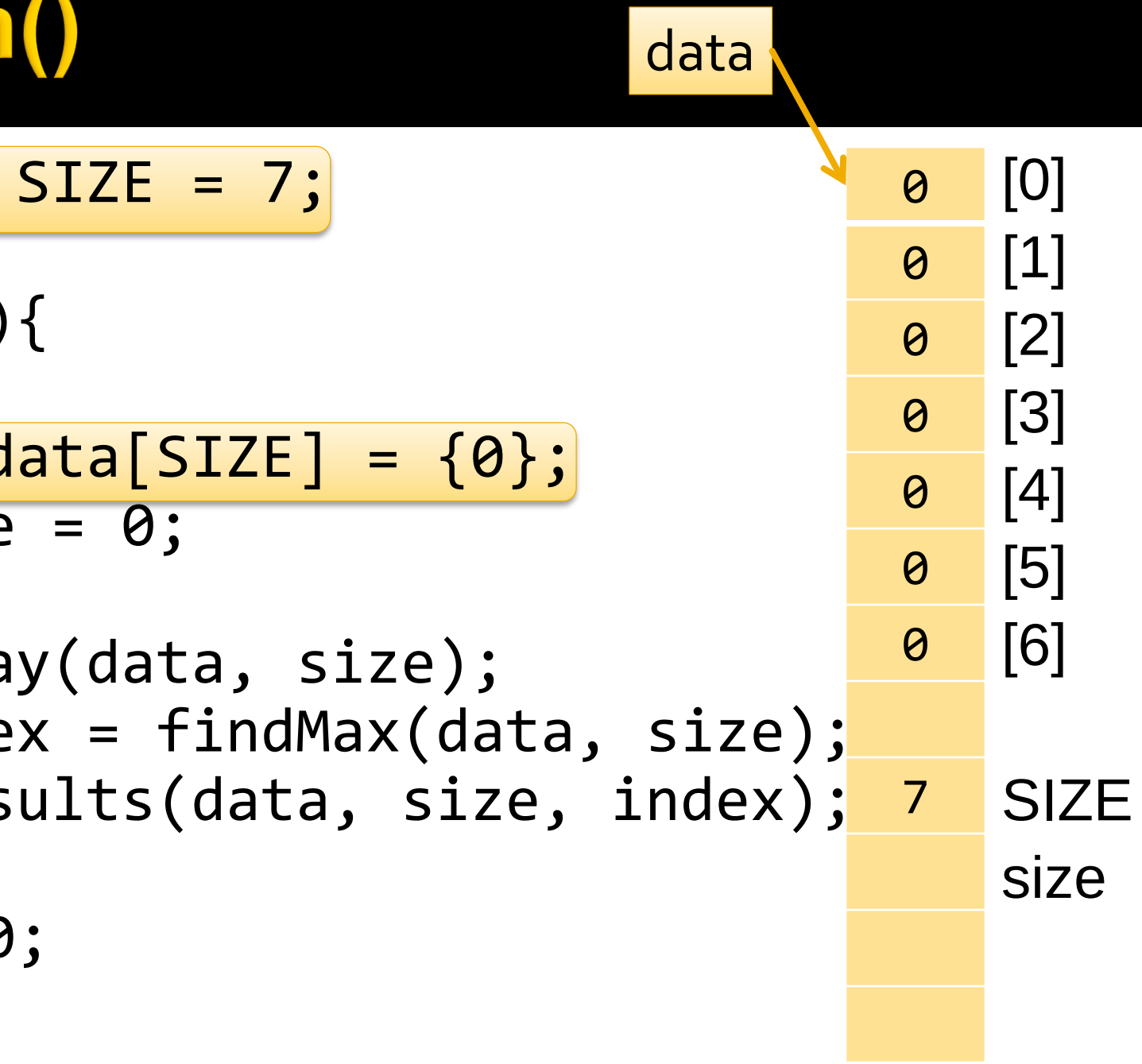
1) Read in values from keyboard
loadArray()
2) Find index of the largest value
findMax()
3)Print values with largest marked
printResults()

SIZE is the max size of the array

```
const int SIZE = 7;

int main(){

   double data[SIZE] = {0};
   int size = 0;

   loadArray(data, size);
   int index = findMax(data, size);
   printResults(data, size, index);

   return 0;
}
```

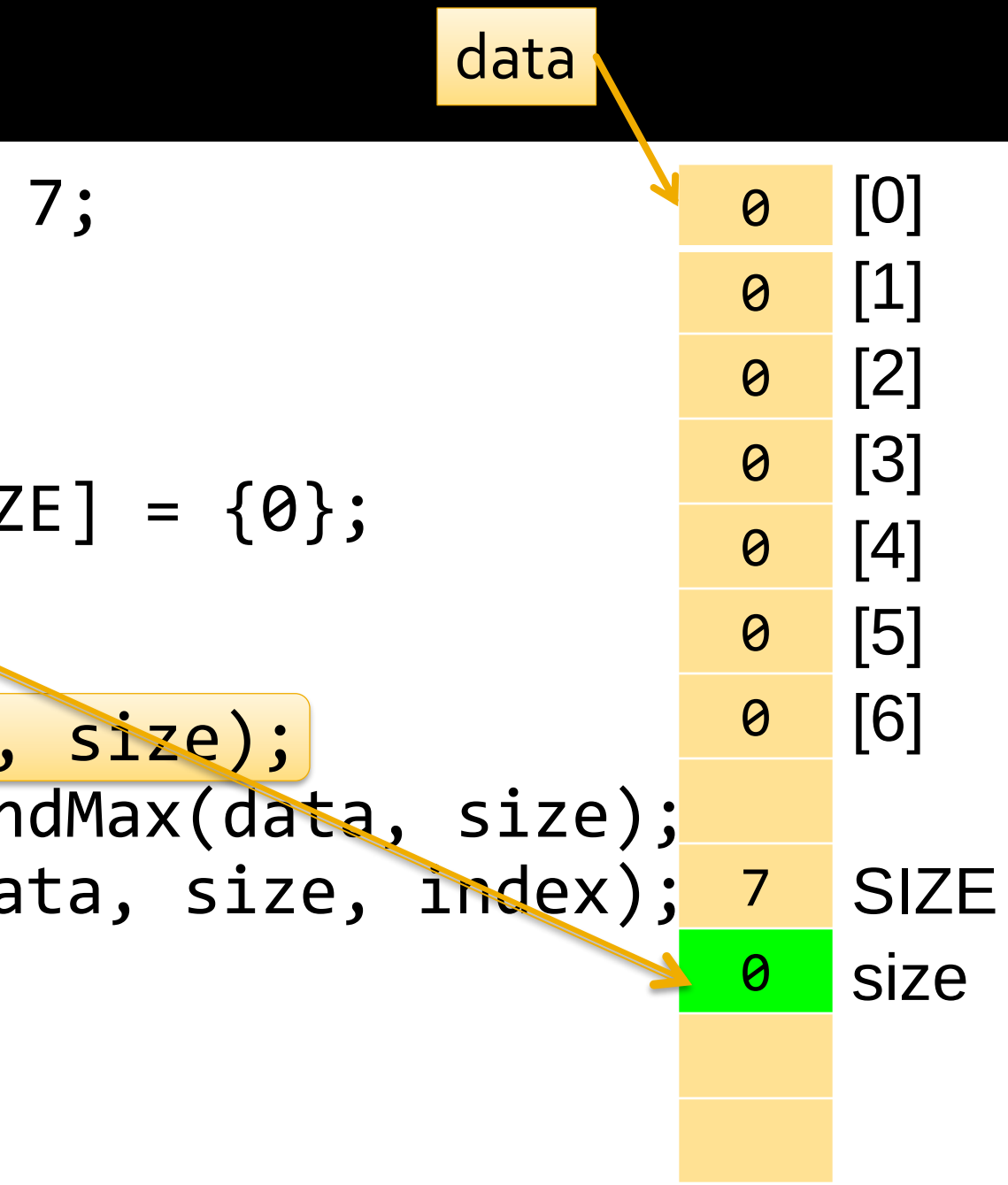
int main()

```
const int SIZE = 7;

int main(){

   double data[SIZE] = {0};
   int size = 0;

   loadArray(data, size);
   int index = findMax(data, size);
   printResults(data, size, index);

   return 0;
}
```



int main()

```
const int SIZE = 7;

int main(){

   double data[SIZE] = {0};
   int size = 0;

   loadArray(data, size);
   int index = findMax(data, size);
   printResults(data, size, index);

   return 0;
}
```

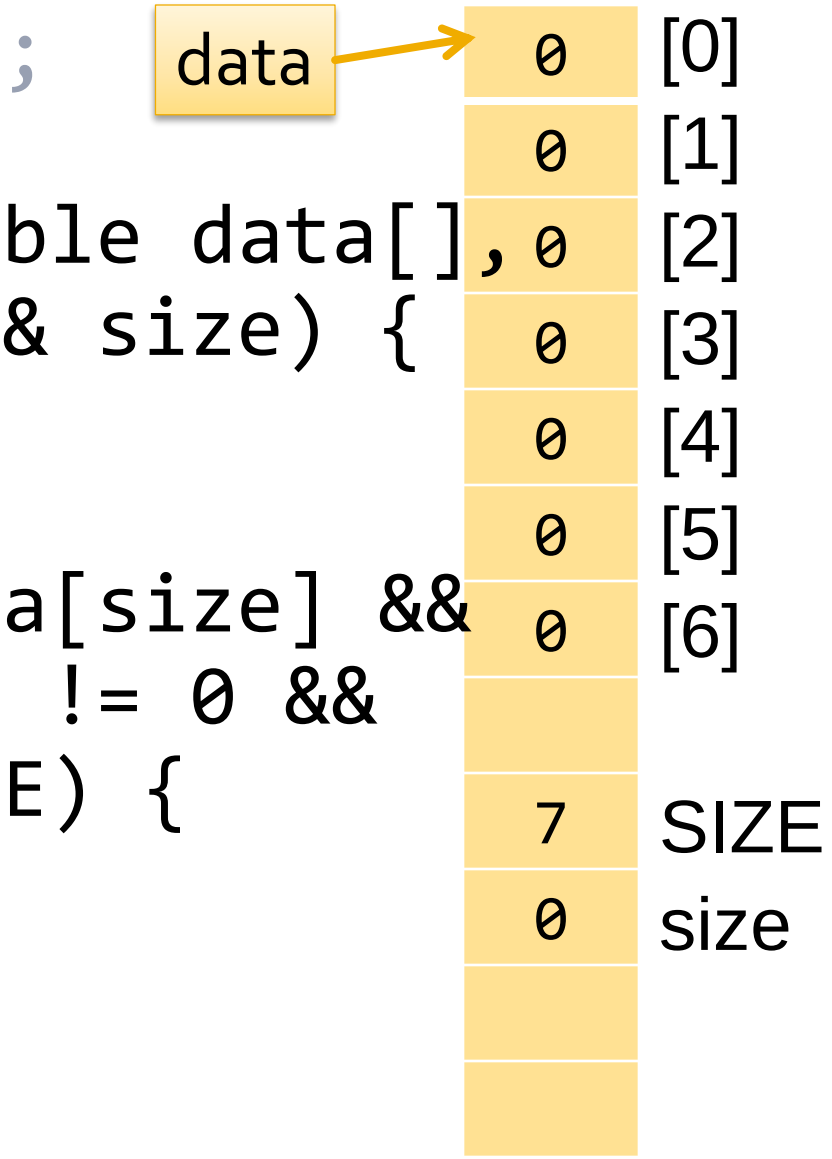


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

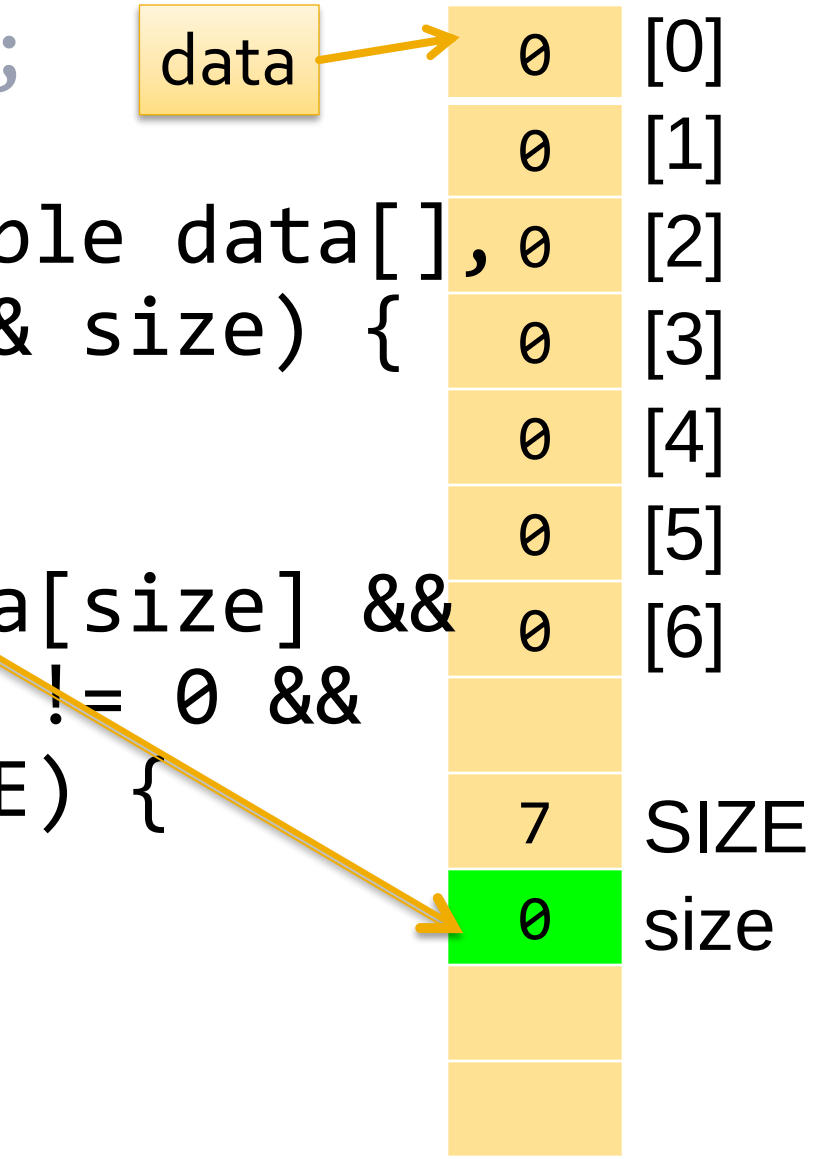


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

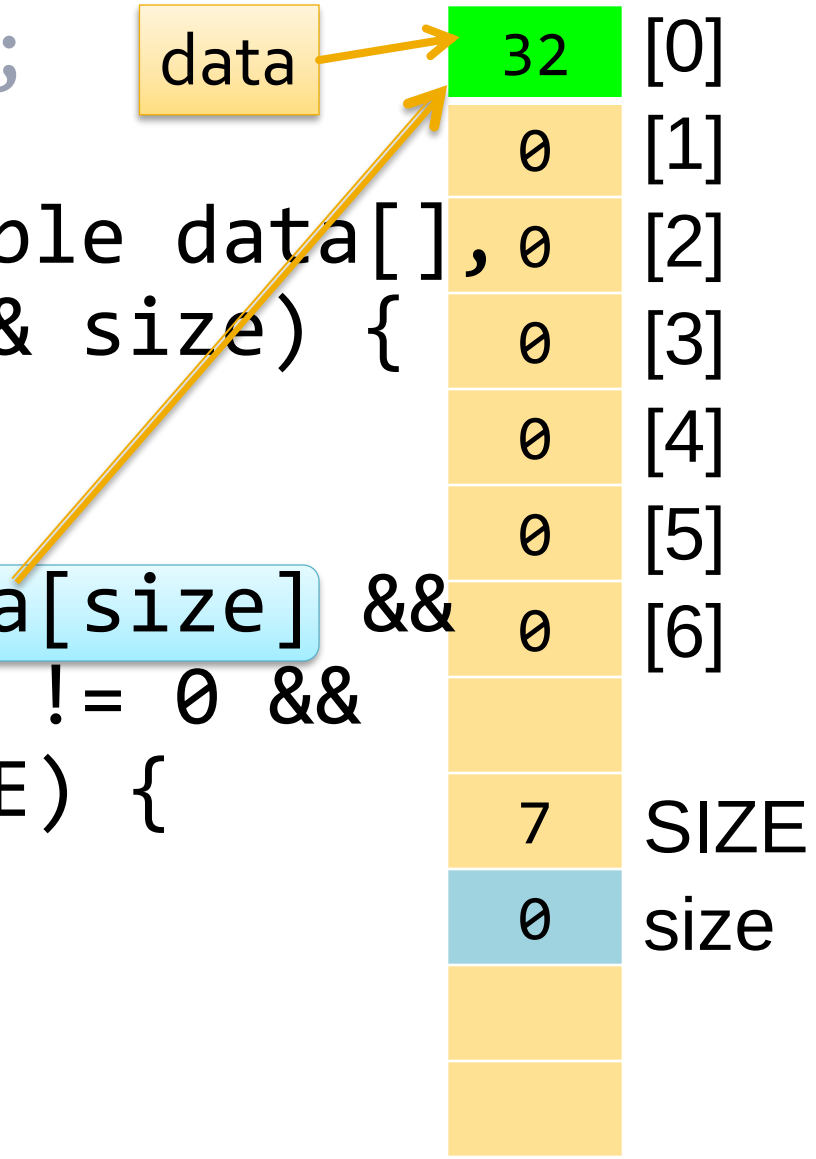


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

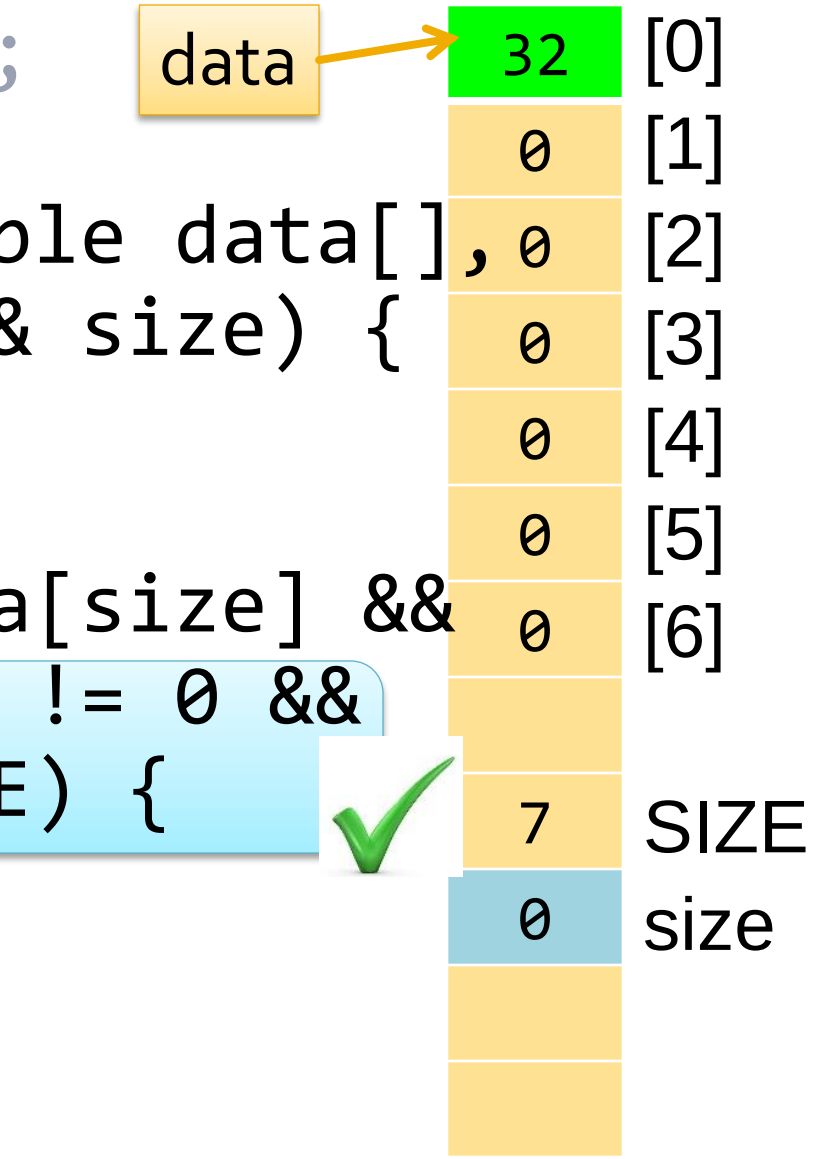


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

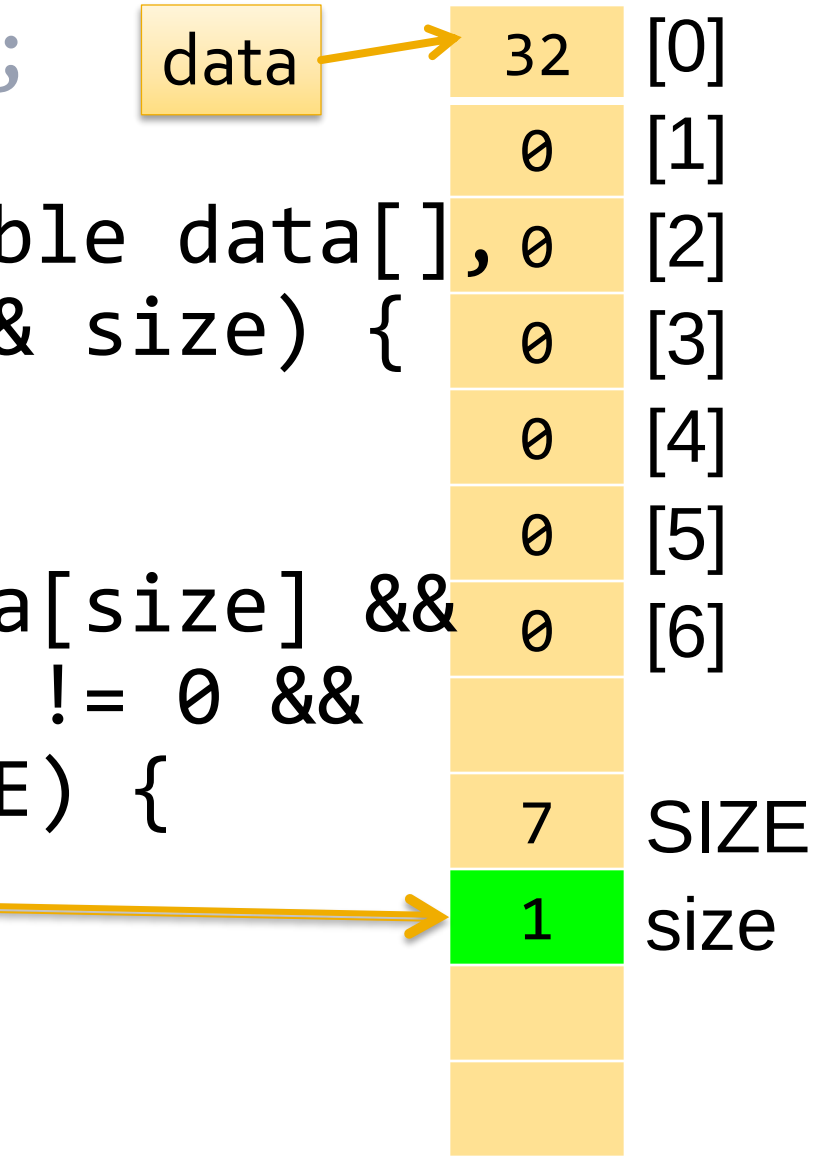


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

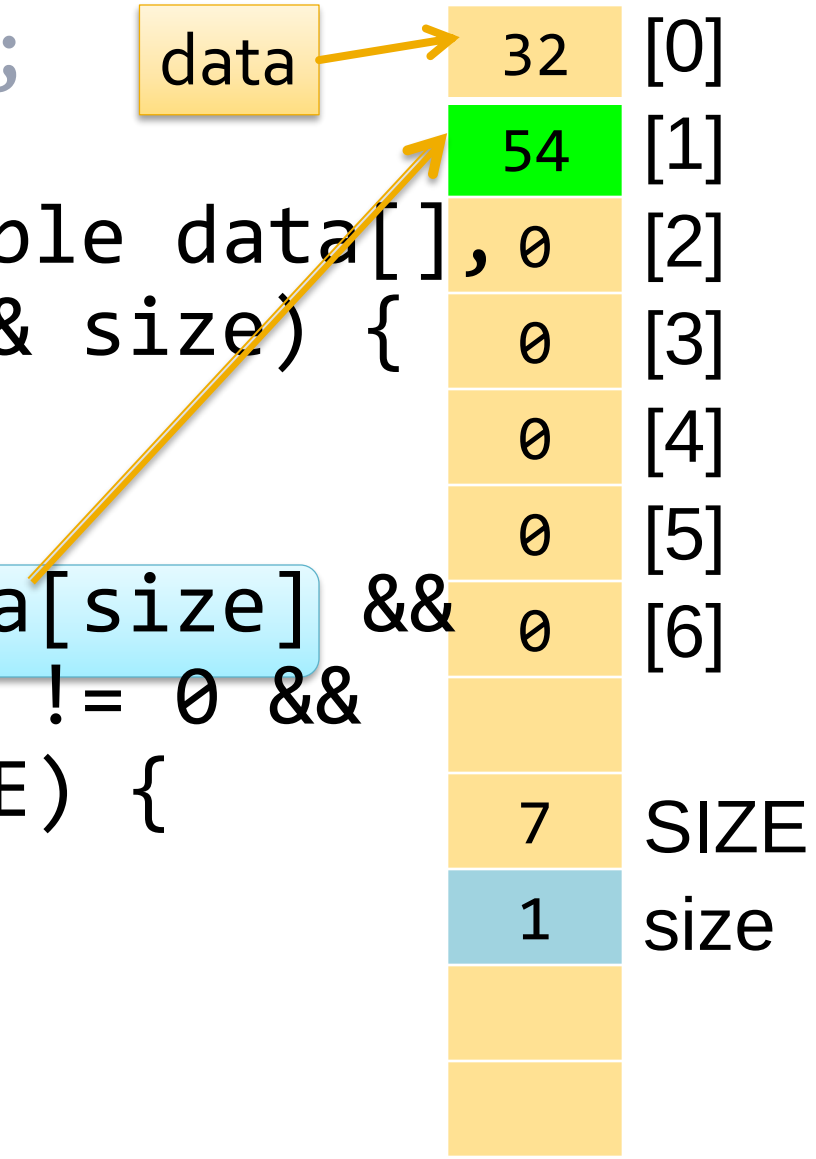


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

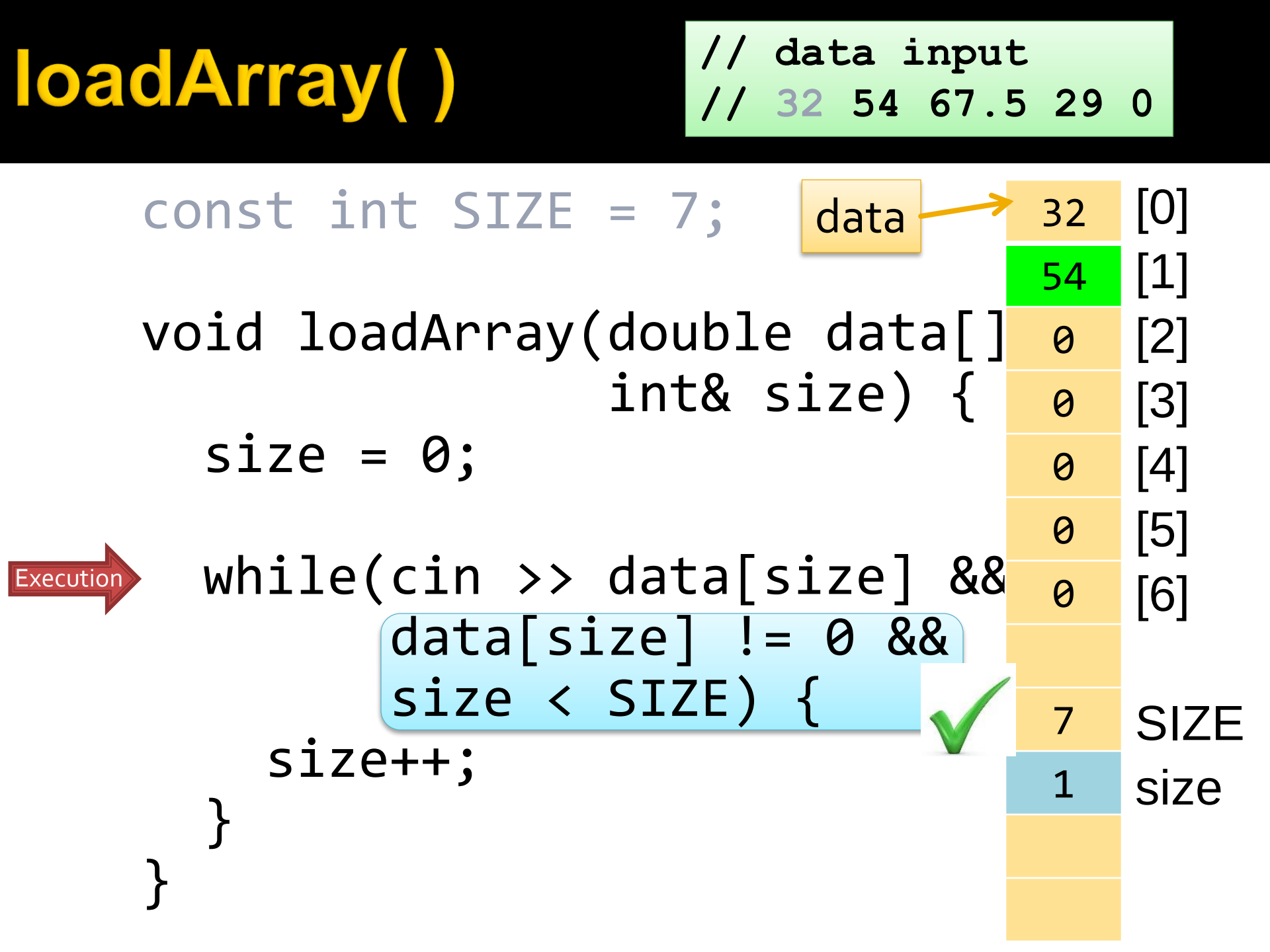


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[]
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

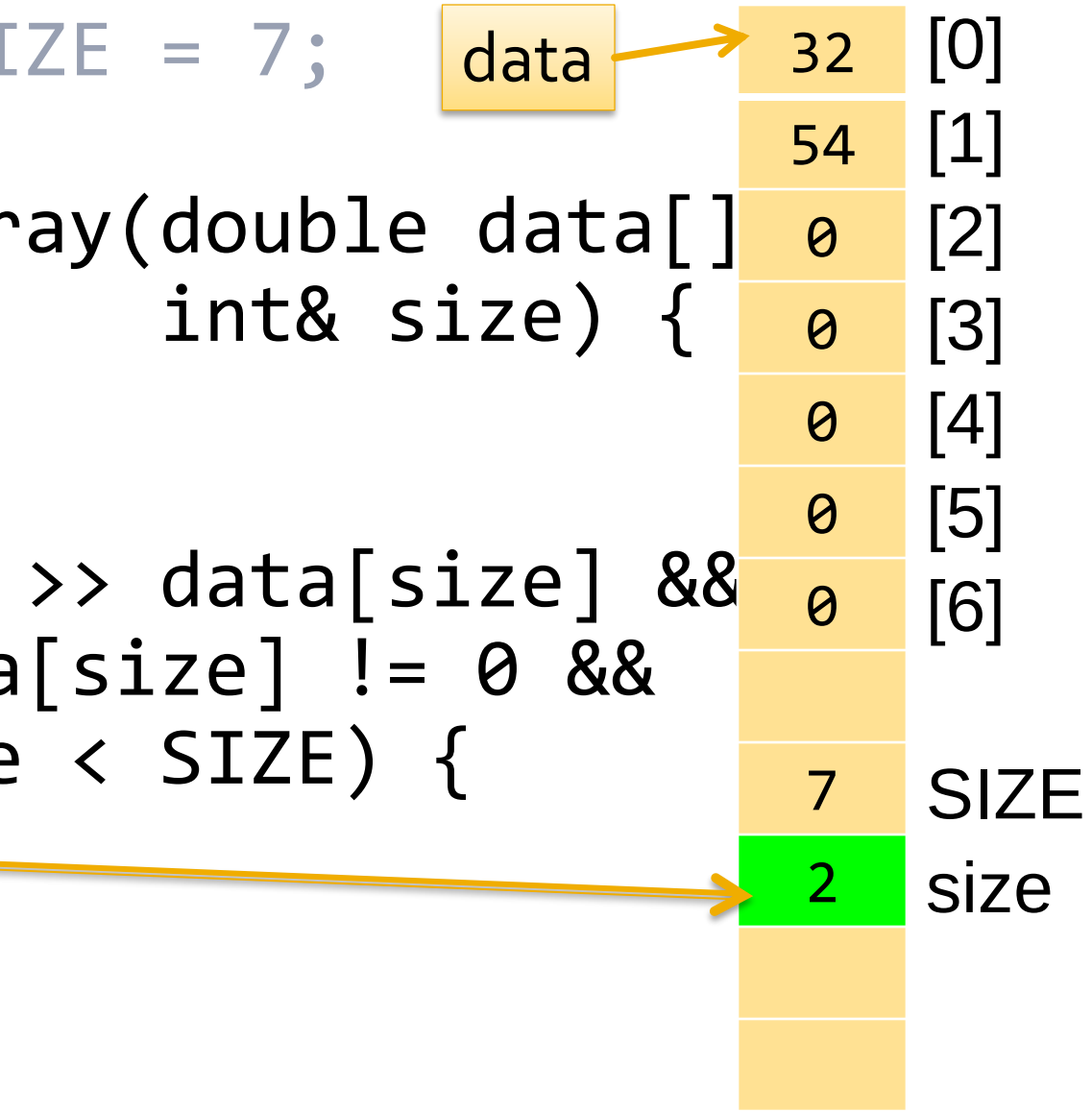


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[]
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

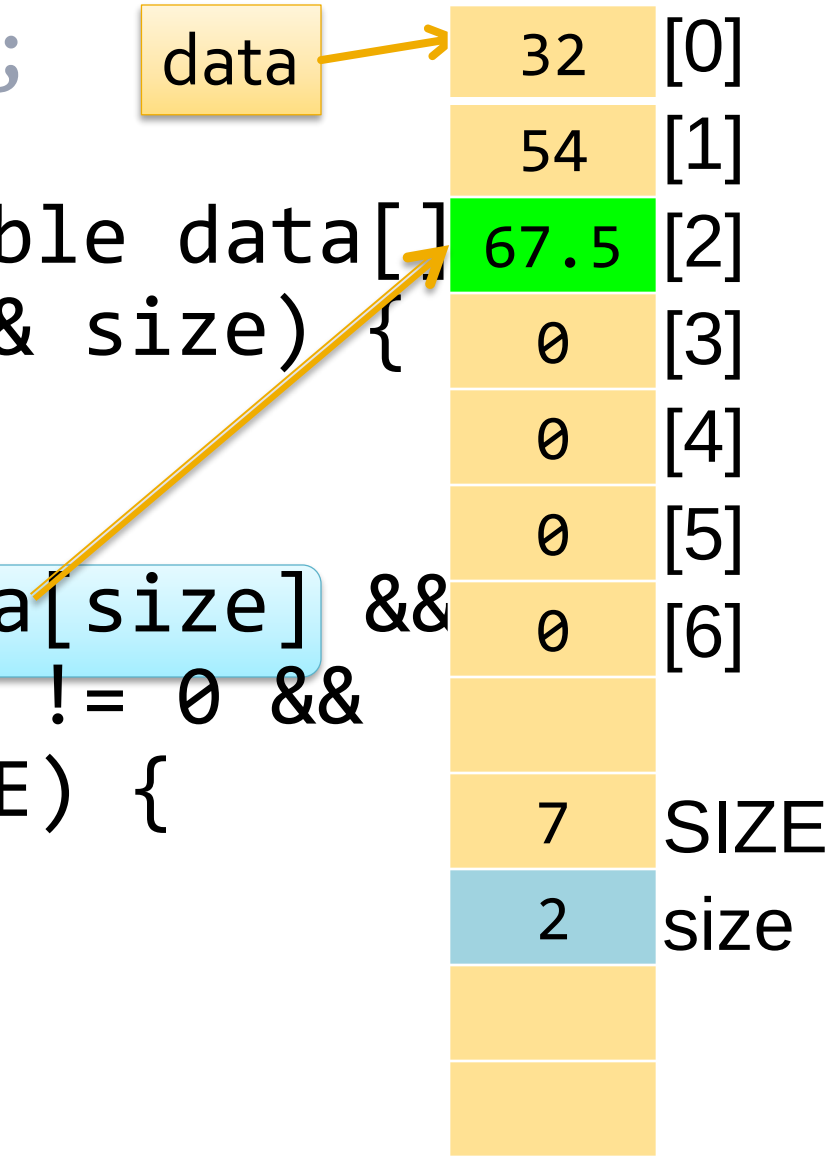


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

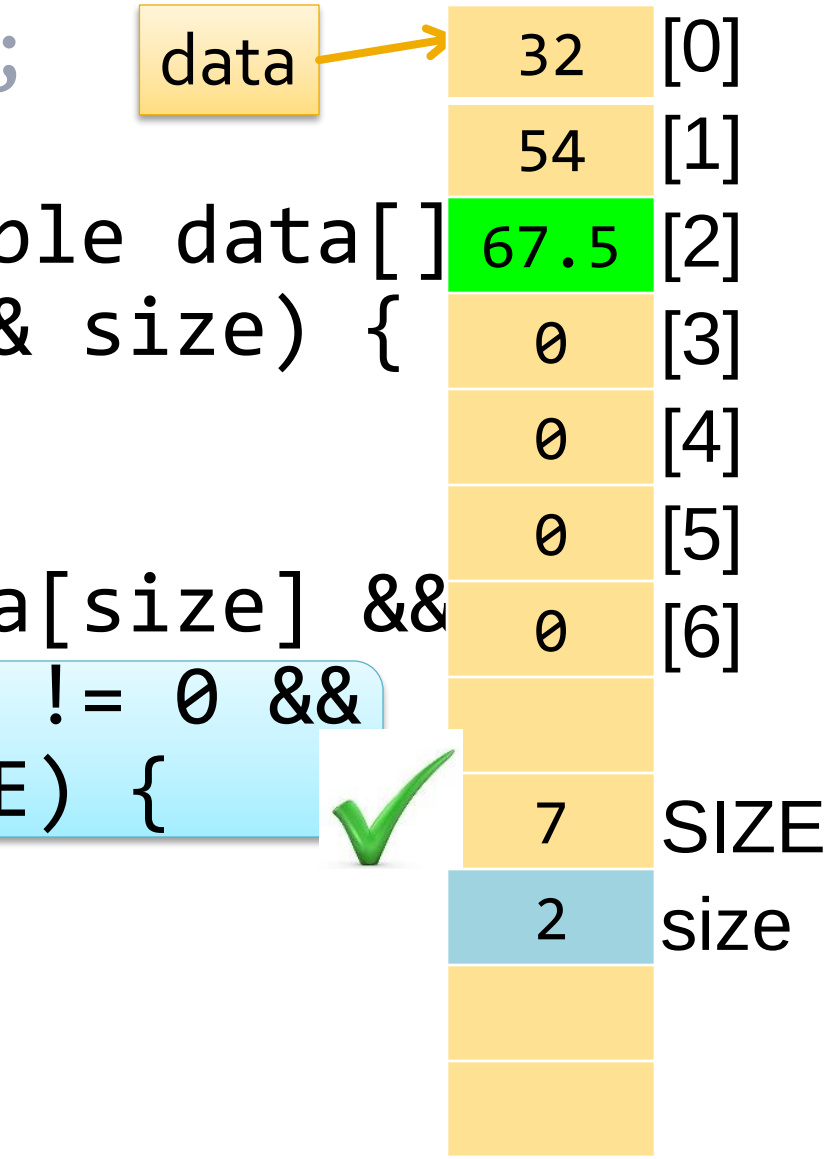


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

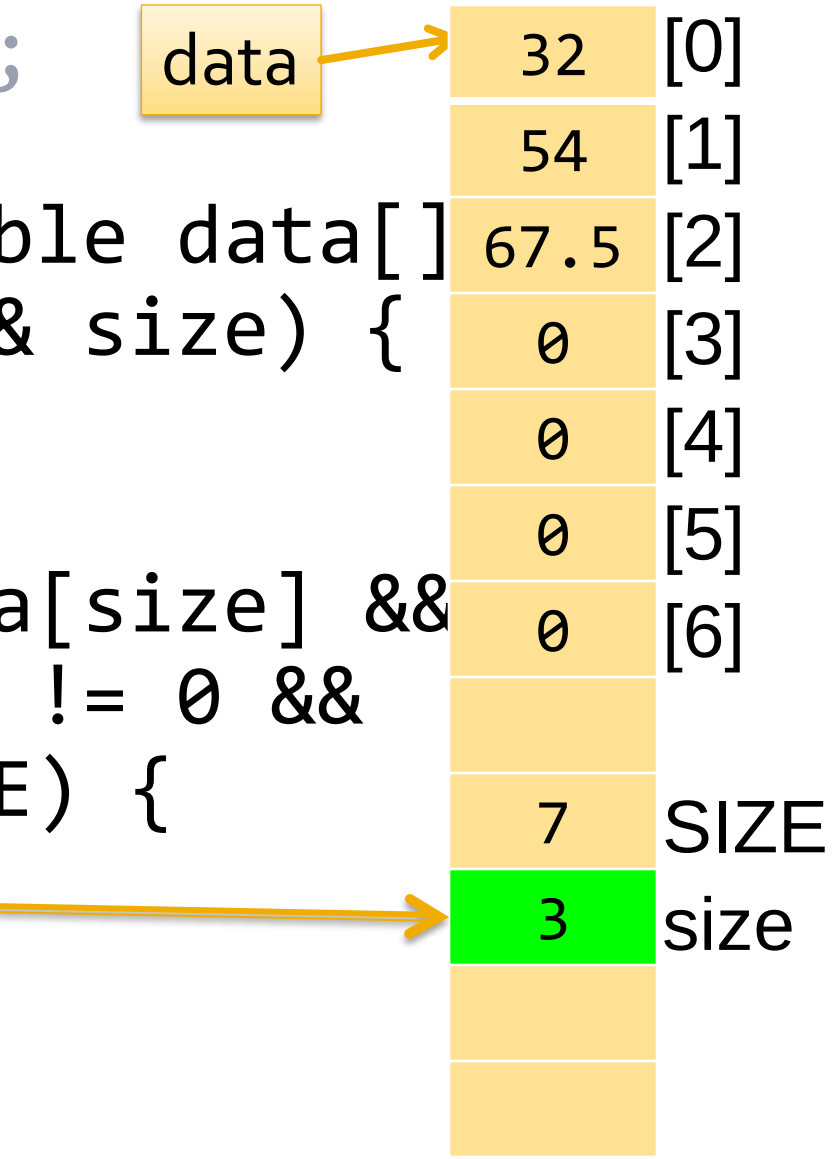


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[]
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

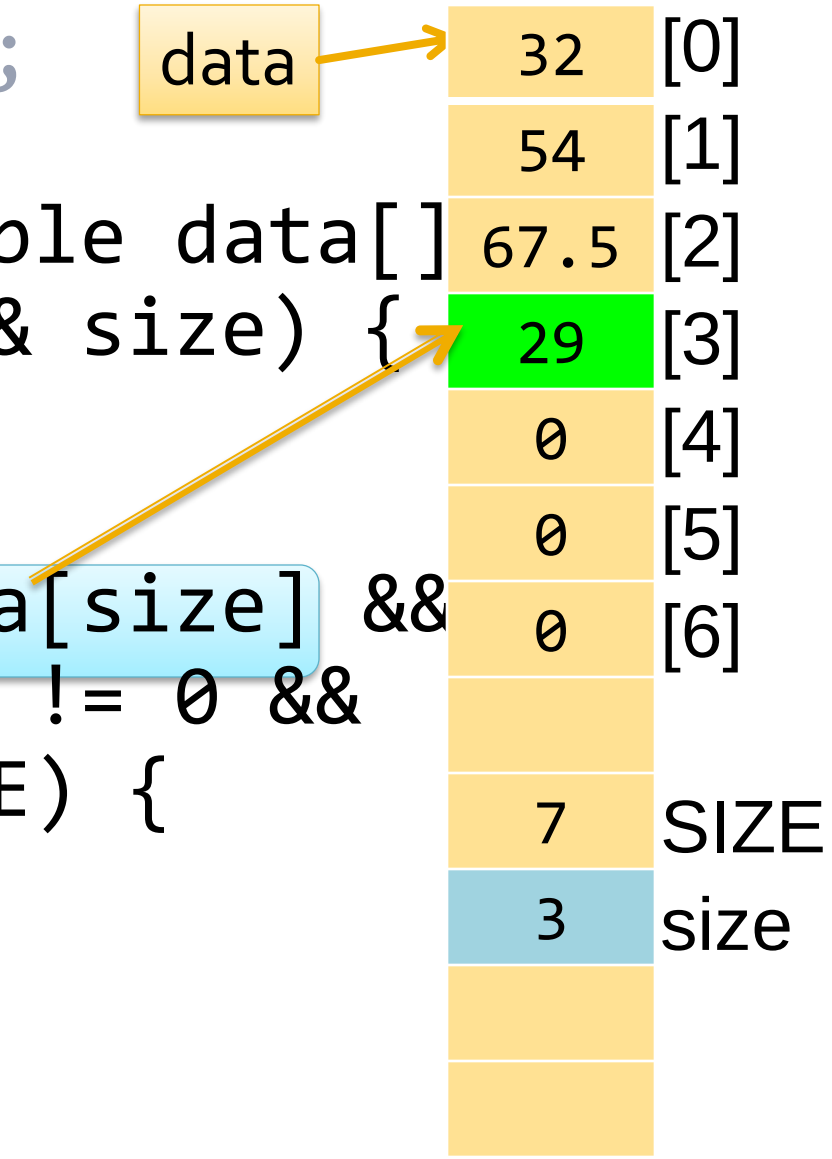


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

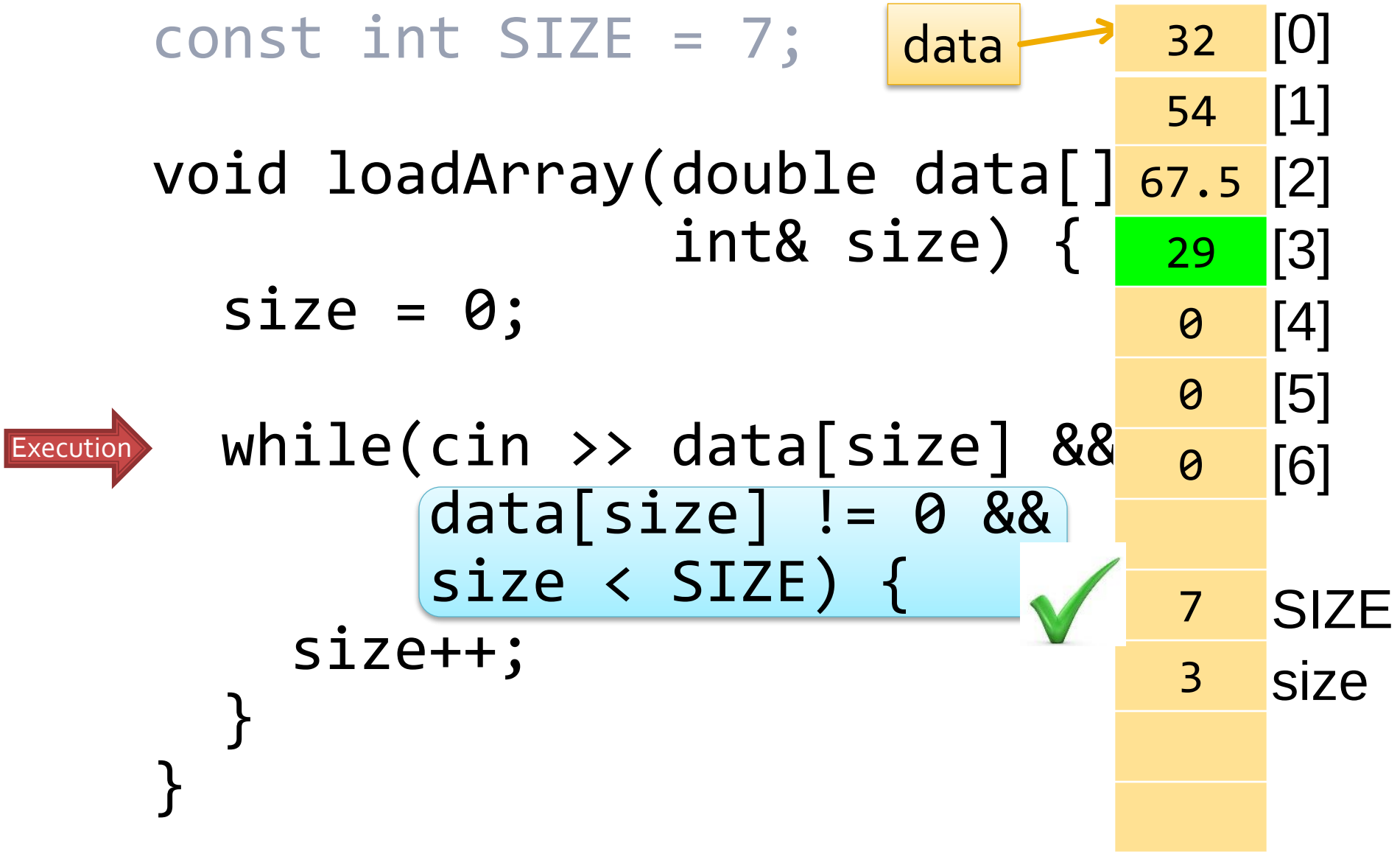


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

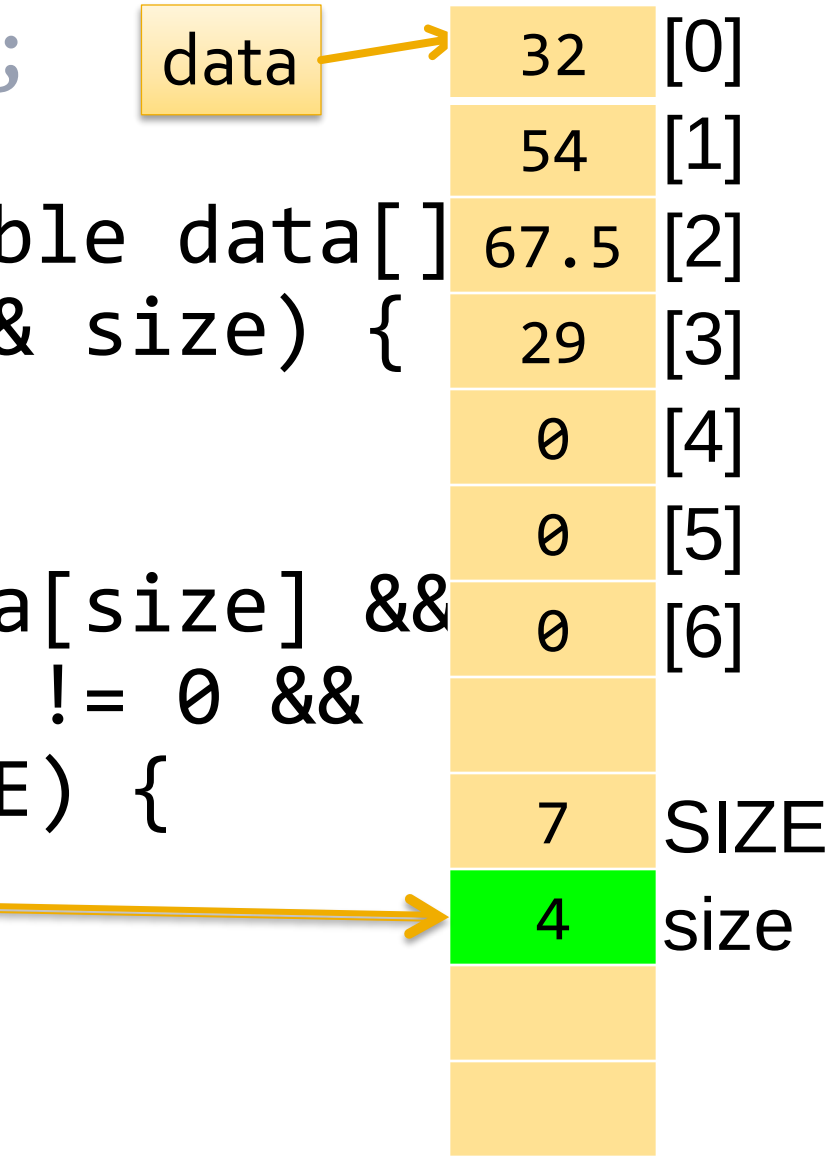


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[]
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

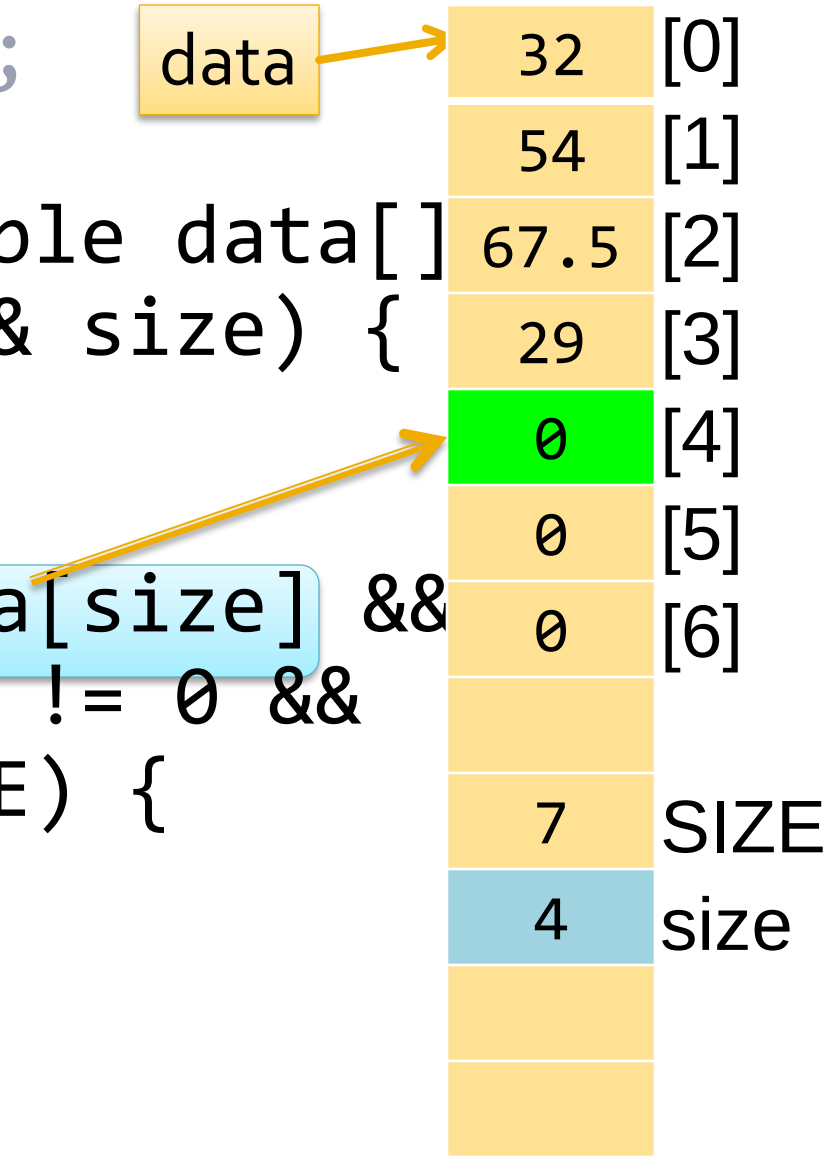


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

Execution

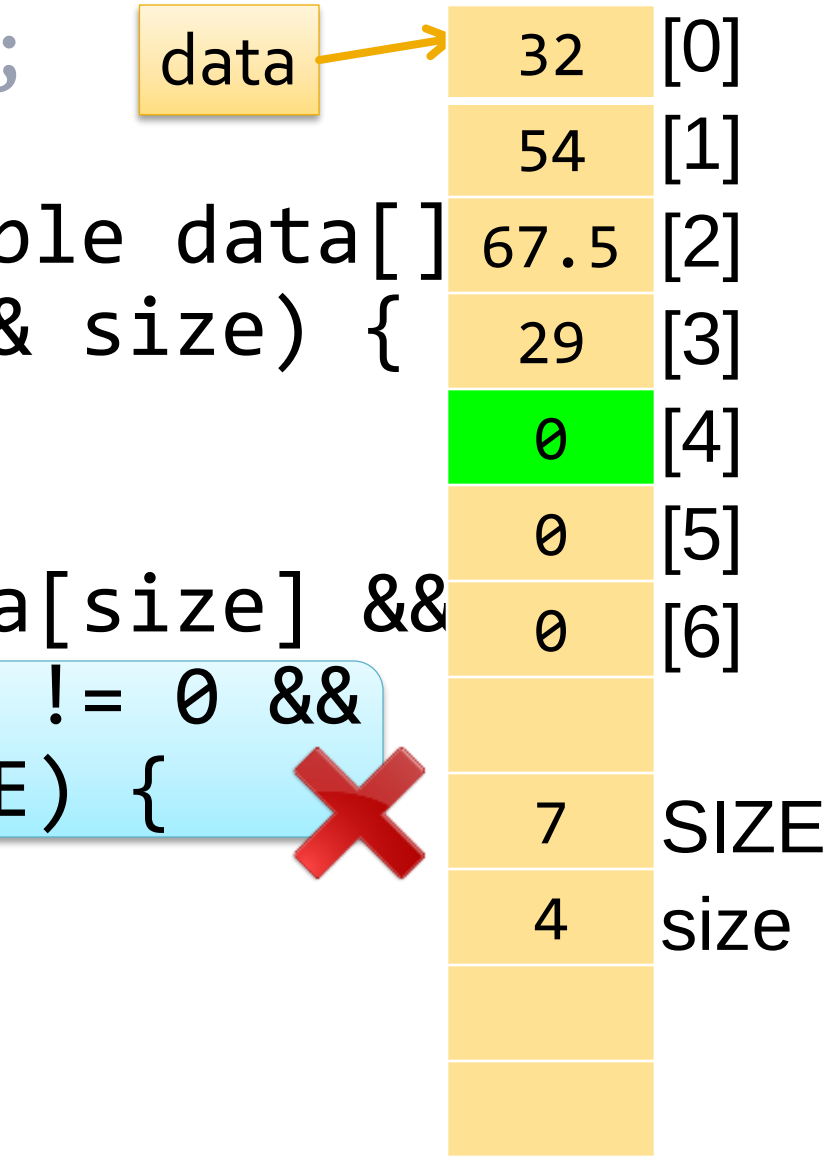


loadArray( )

```
// data input
// 32 54 67.5 29 0
```

```
const int SIZE = 7;

void loadArray(double data[],
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```

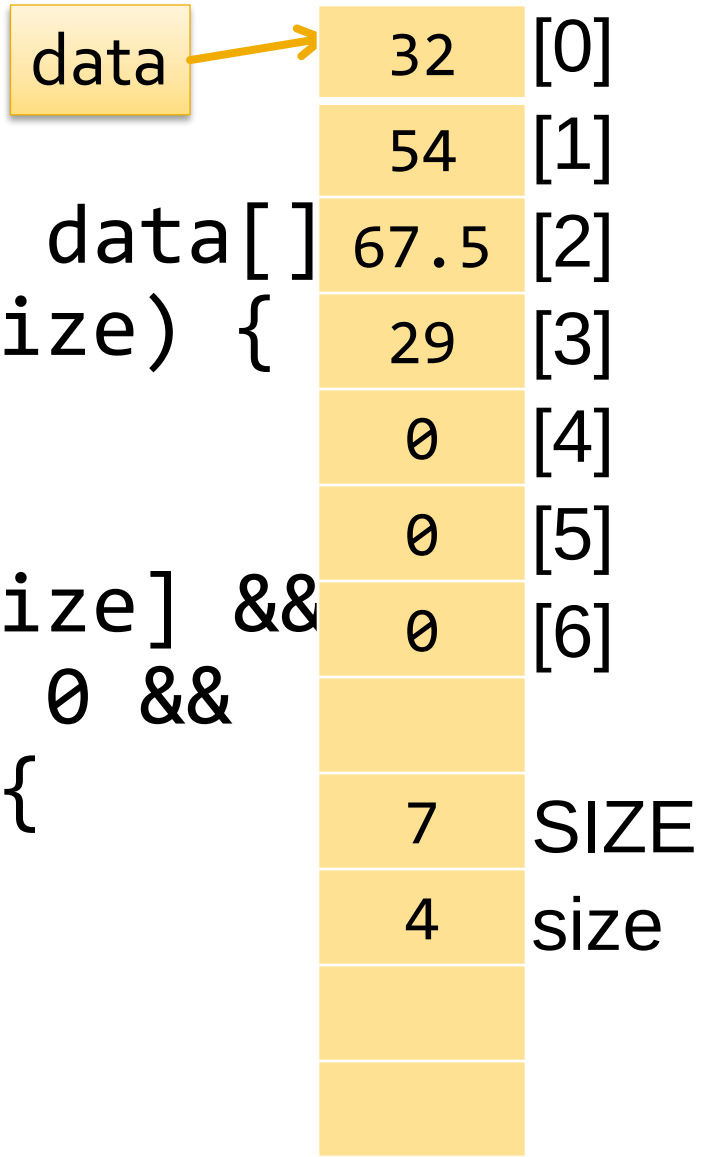
Execution



loadArray( )

```
// data input
// 32 54 67.5 29 0
```

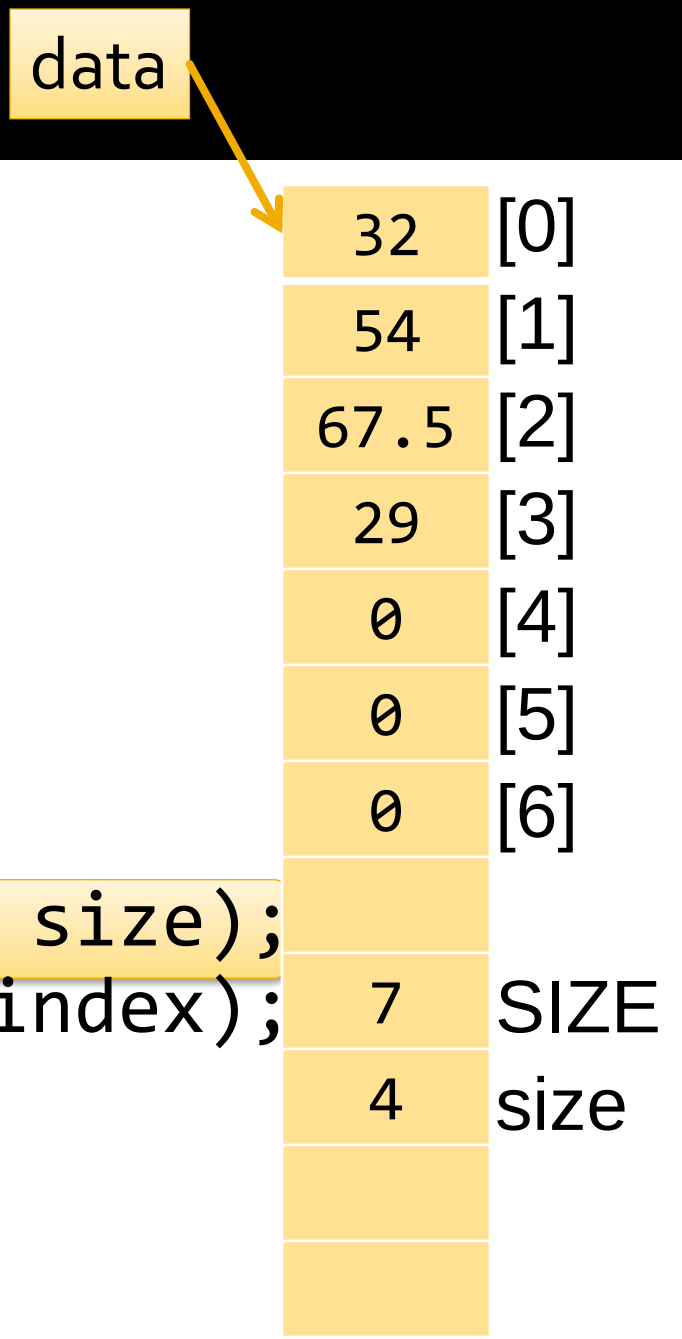
```
const int SIZE = 7;

void loadArray(double data[]
               int& size) {
  size = 0;

  while(cin >> data[size] &&
        data[size] != 0 &&
        size < SIZE) {
    size++;
  }
}
```



Execution

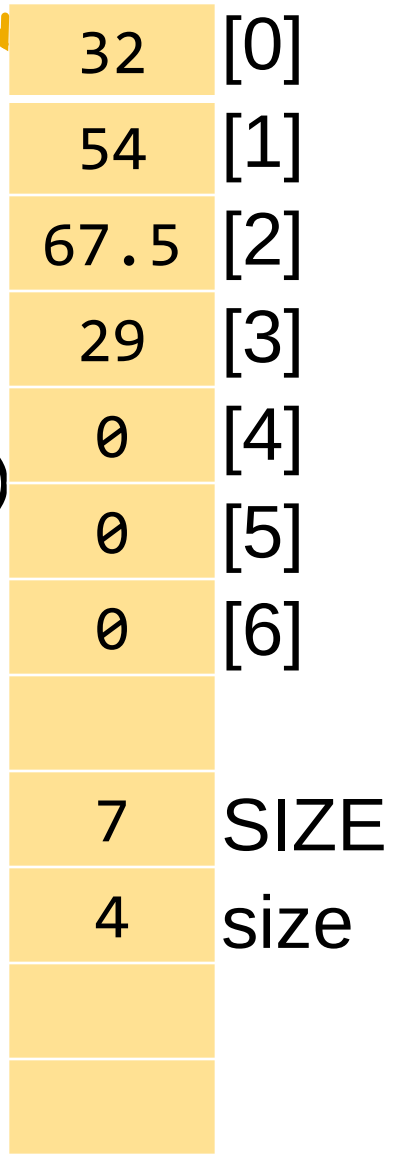
int main( )

```
const int SIZE = 7;

int main(){

   double data[SIZE];
   int size = 0;

   loadArray(data, size);
   int index = findMax(data, size);
   printResults(data, size, index);

   return 0;
}
```



findMax( )

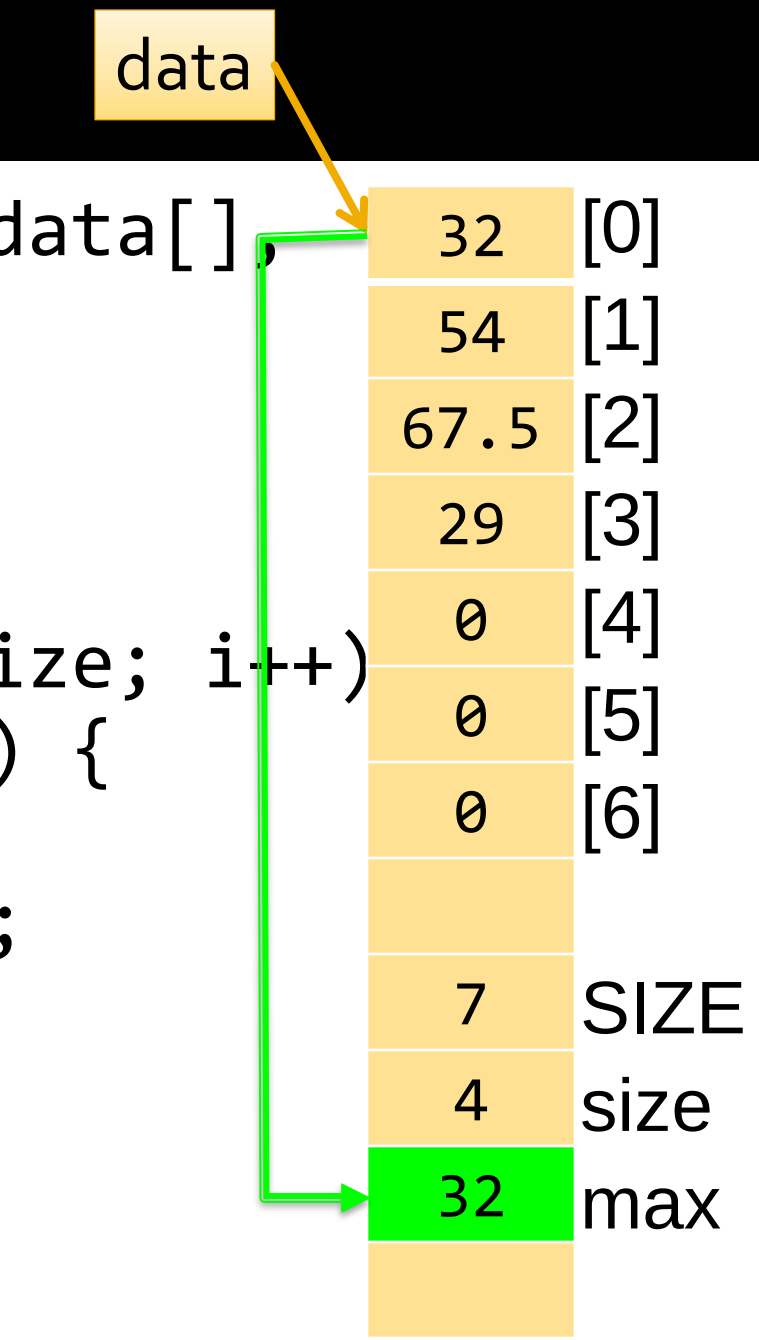
```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

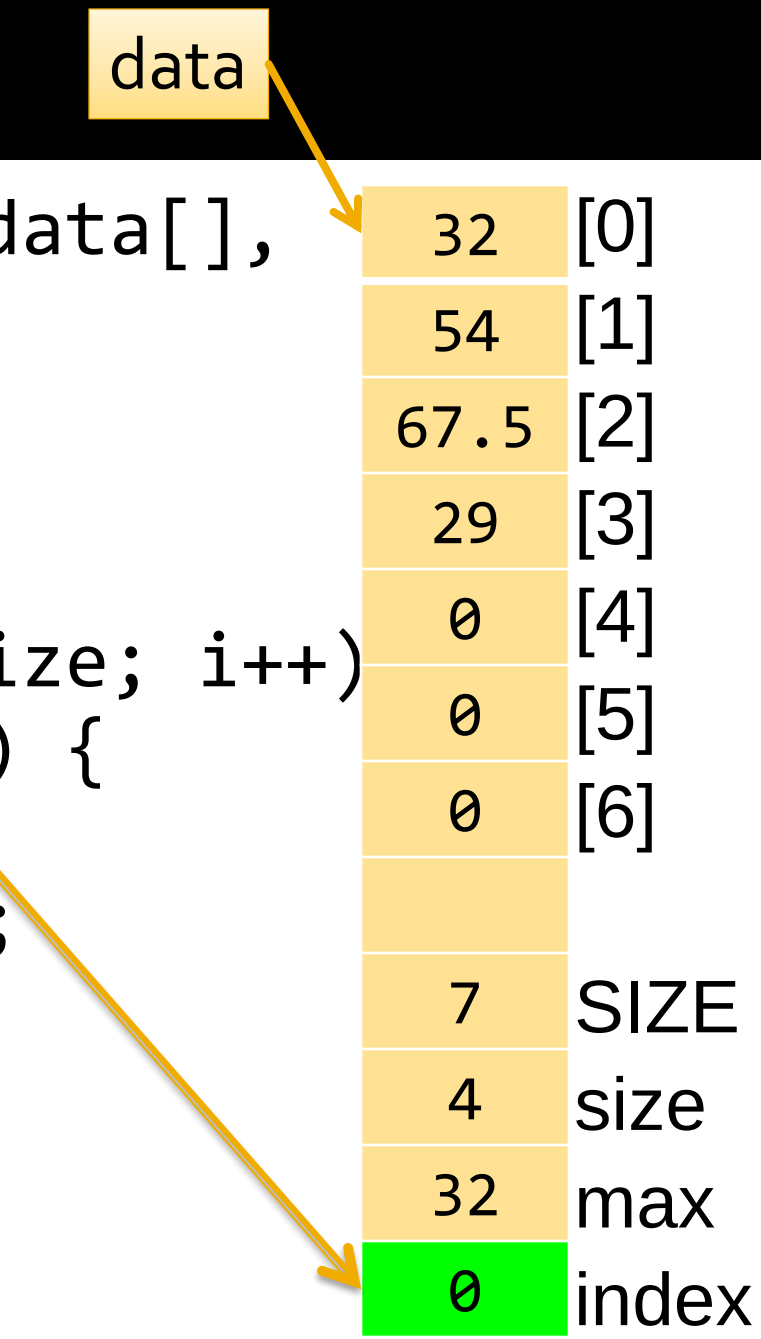
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

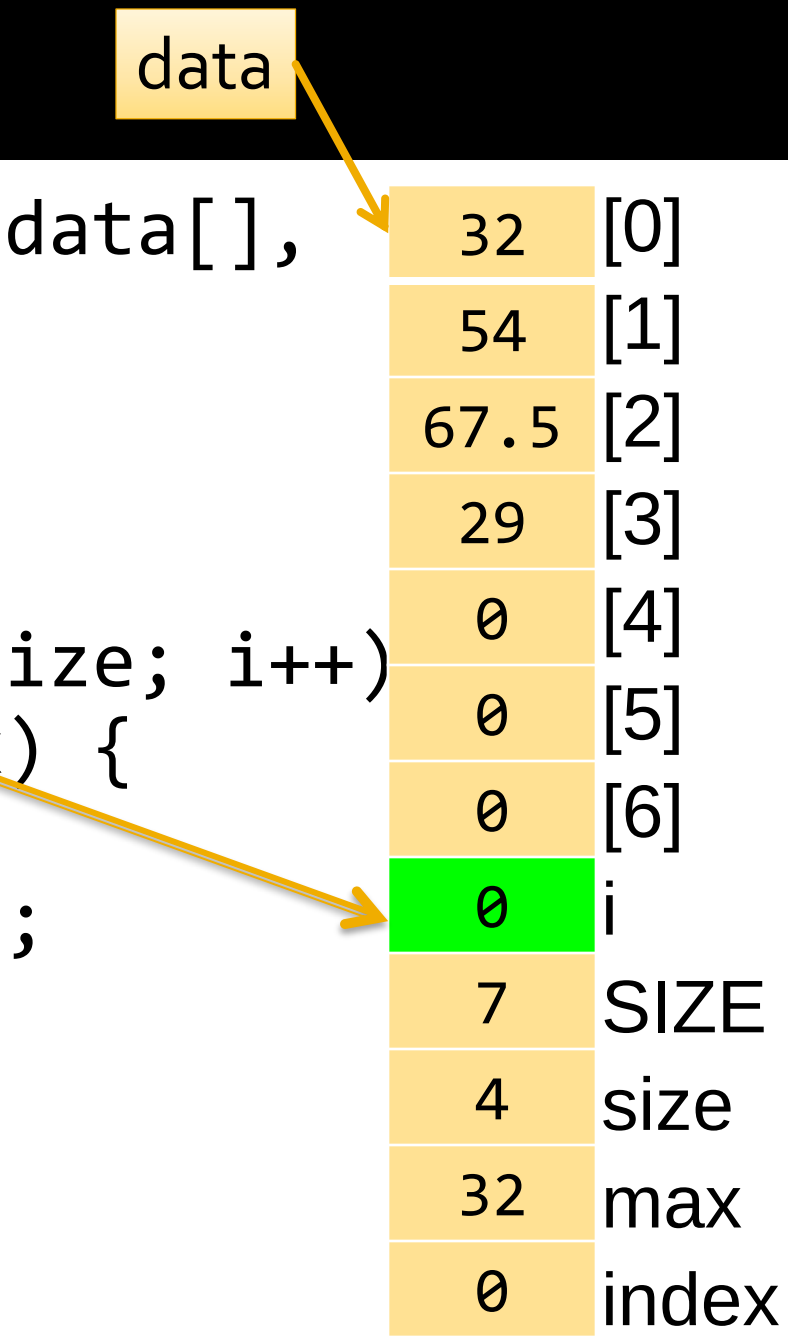
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

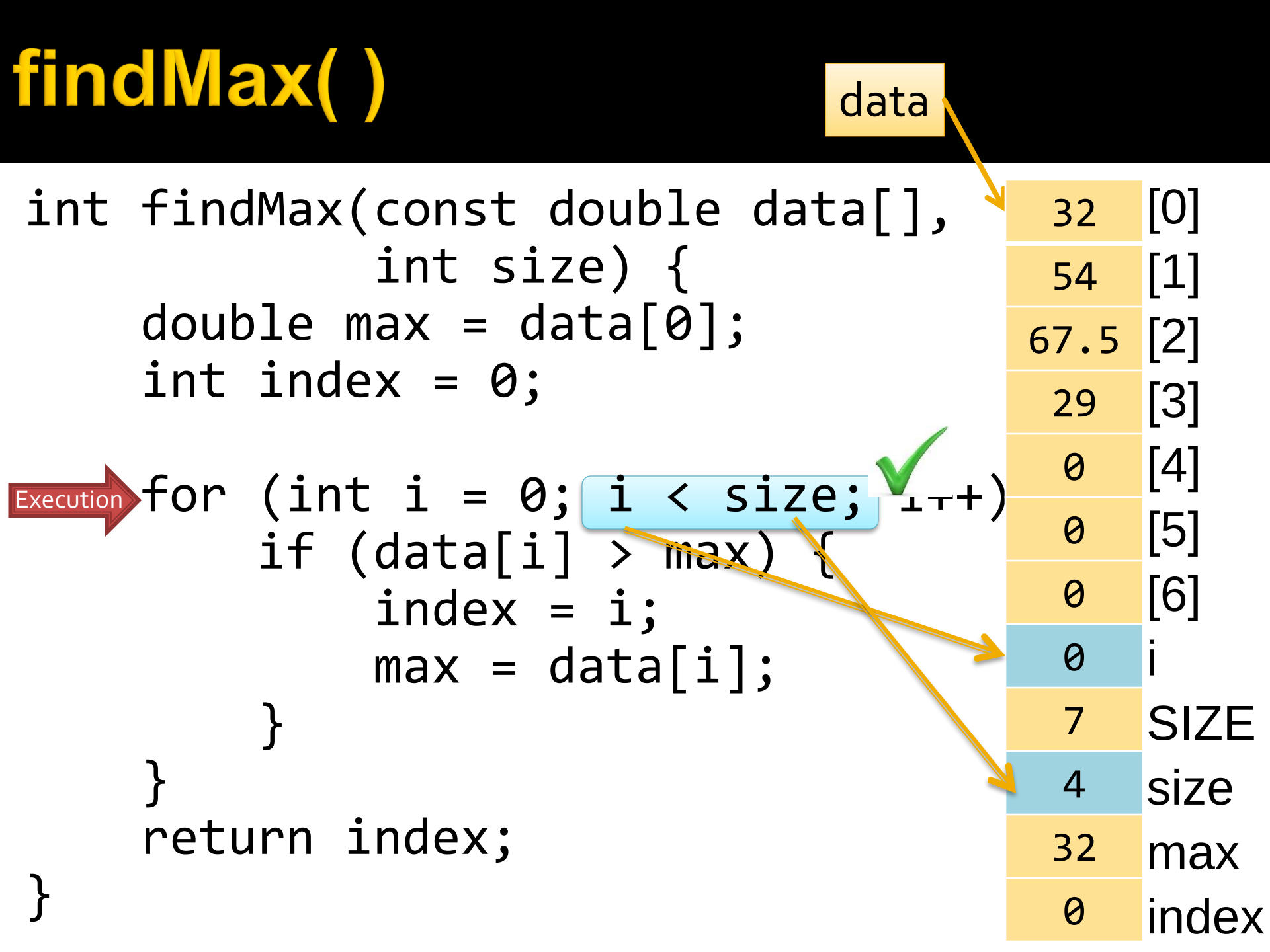
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

Execution

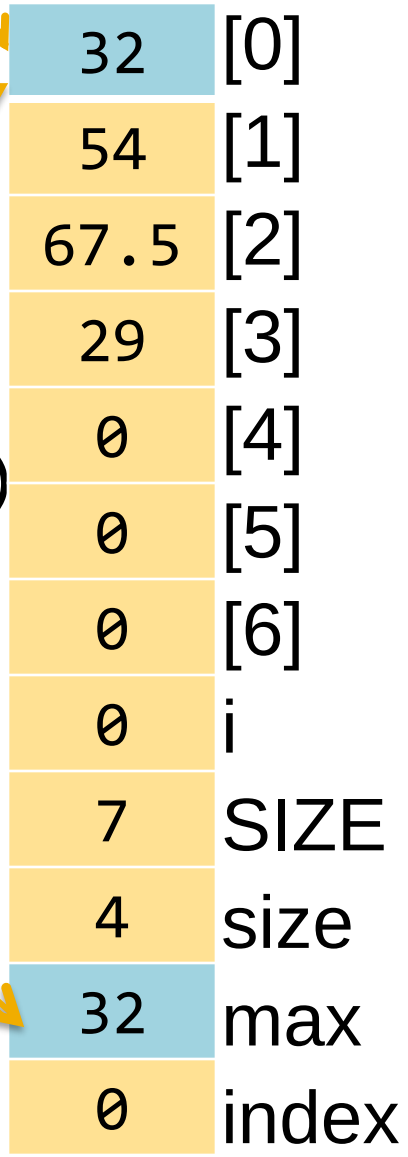


findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

Execution

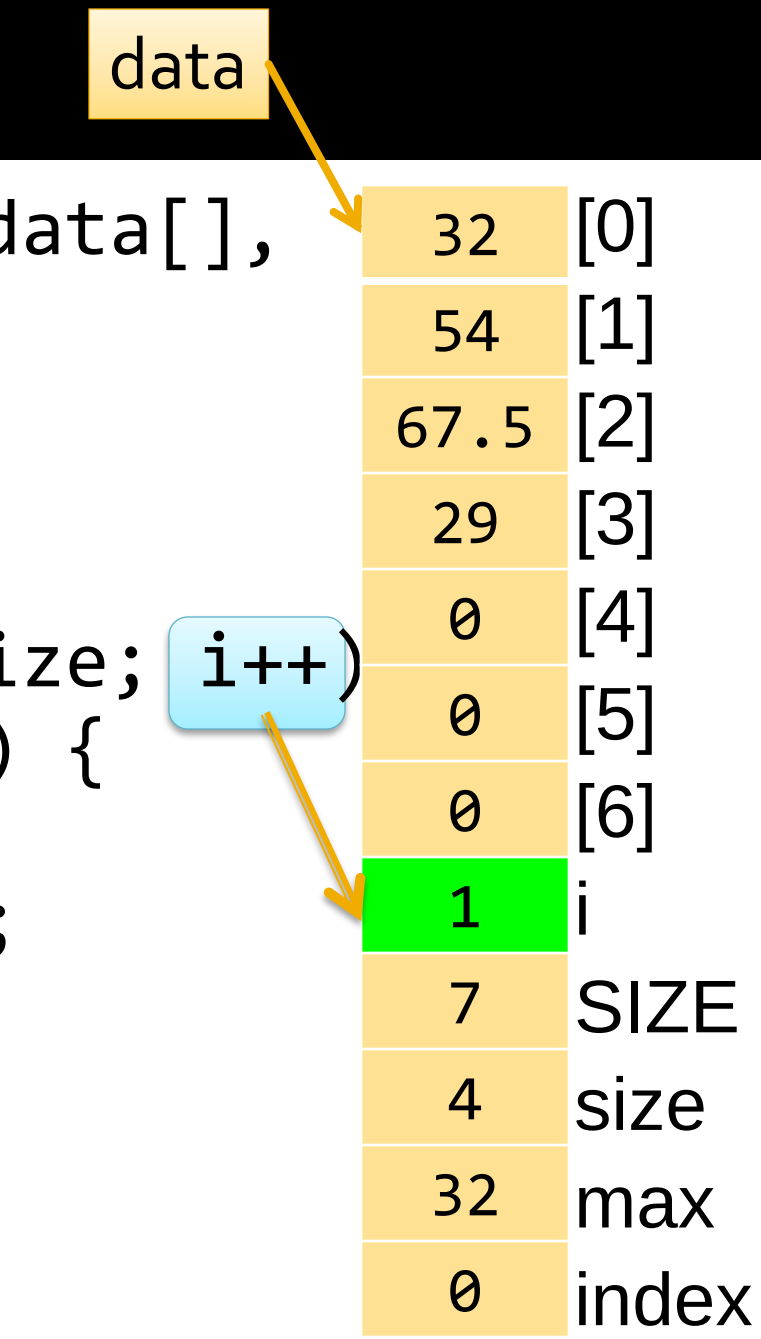
data



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

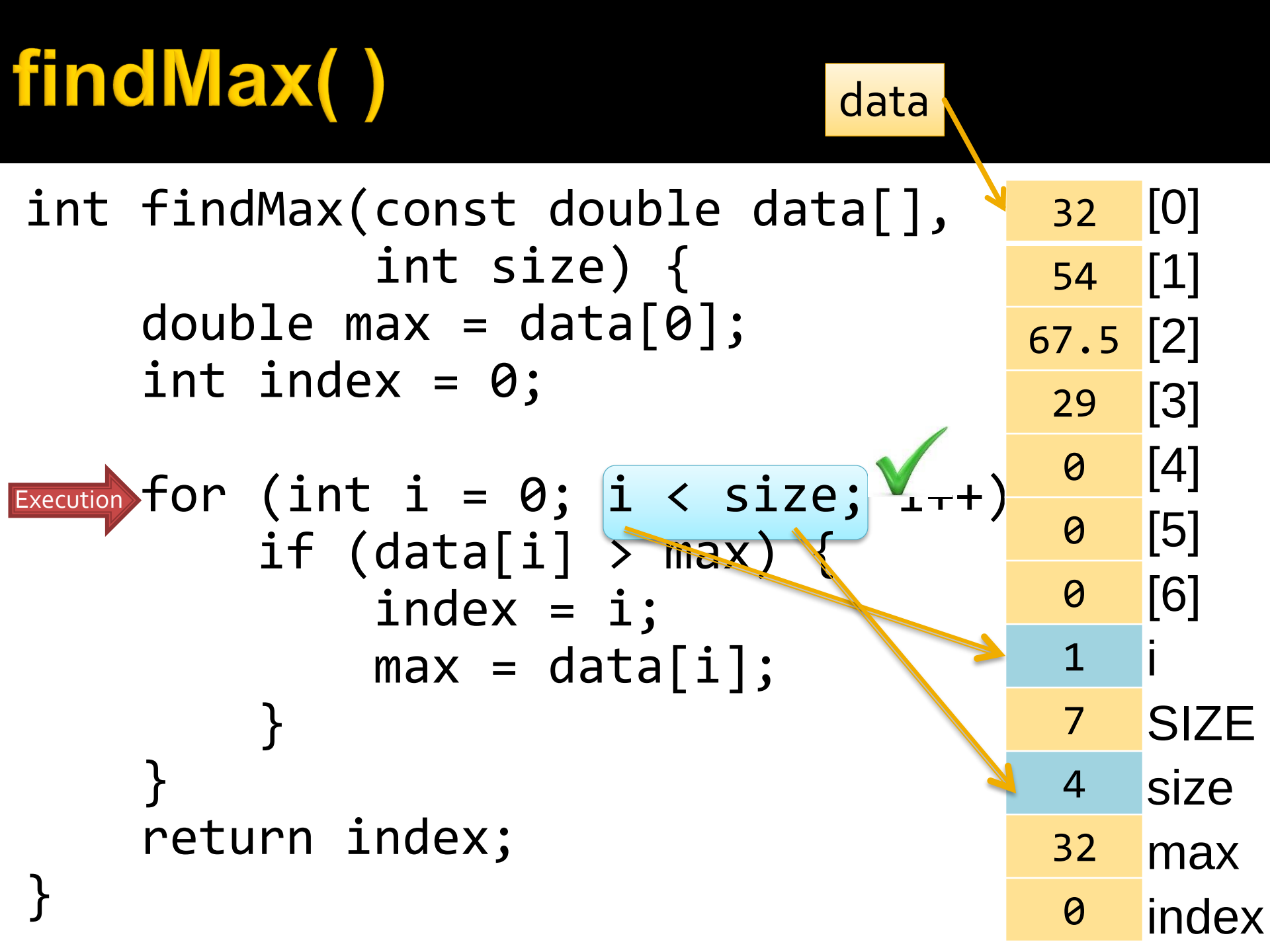
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

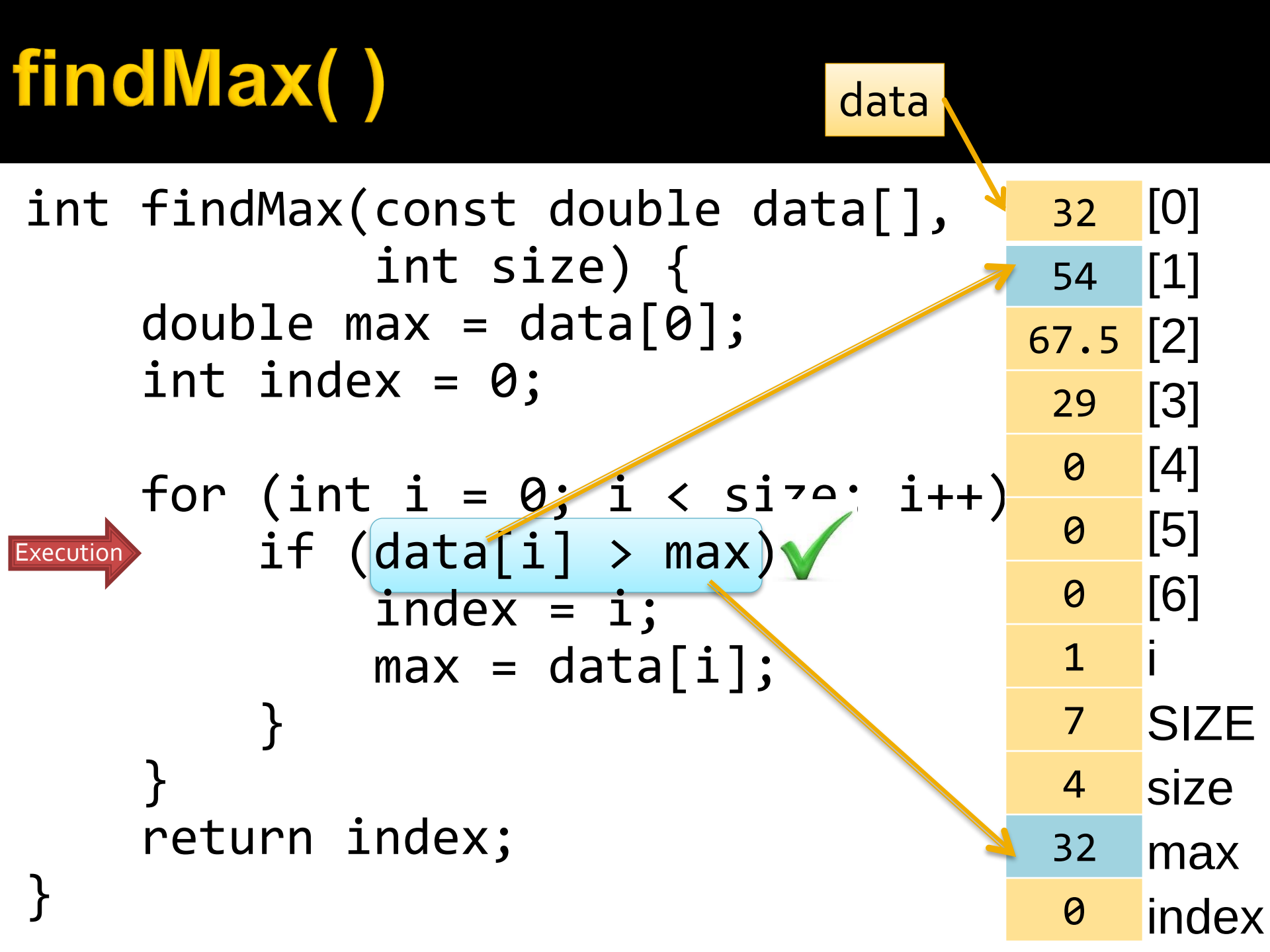
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max)
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

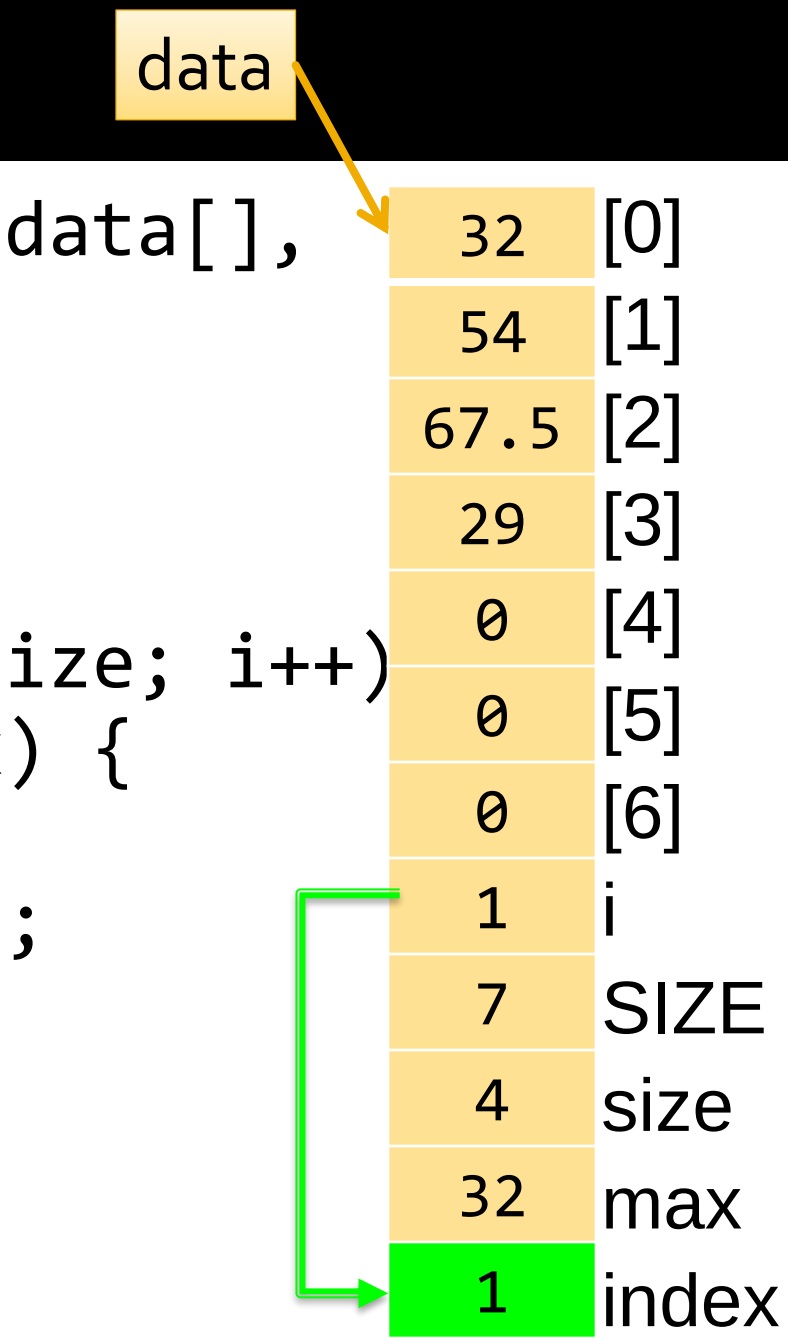
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

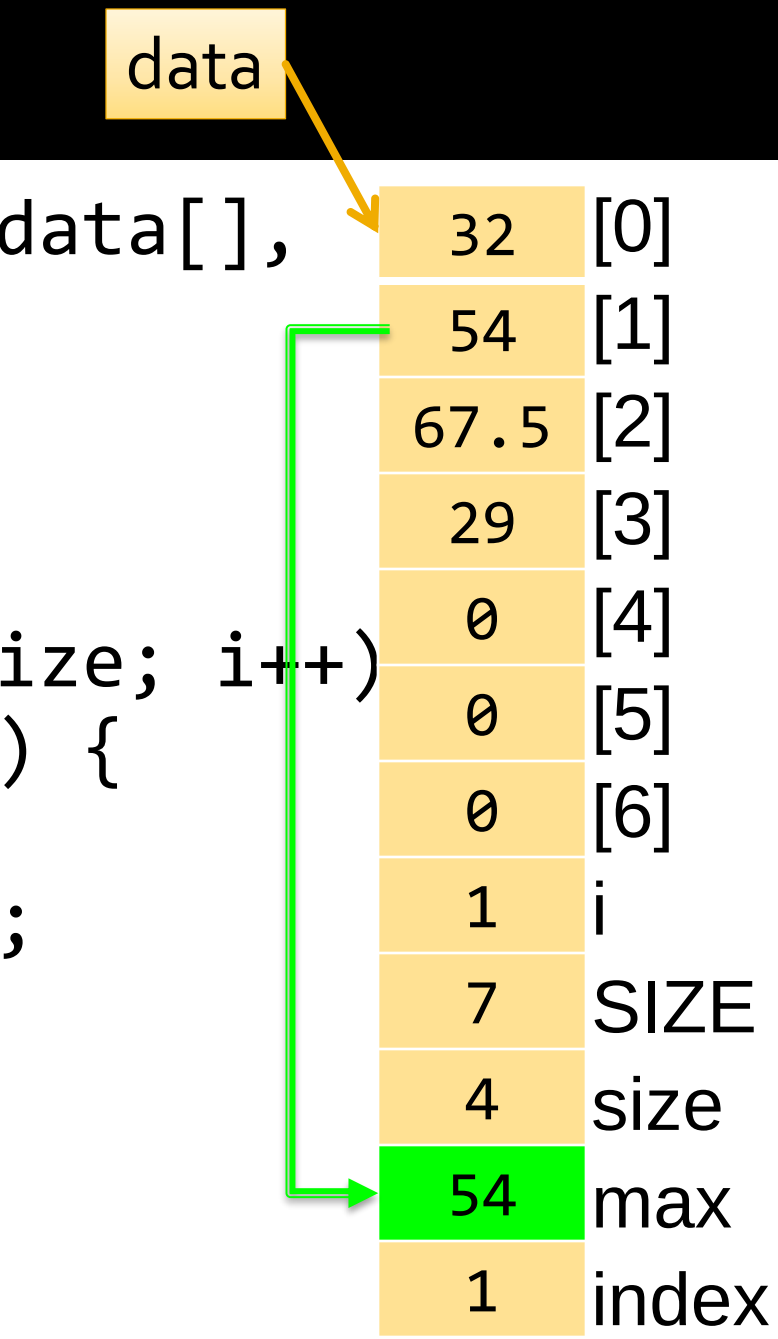
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

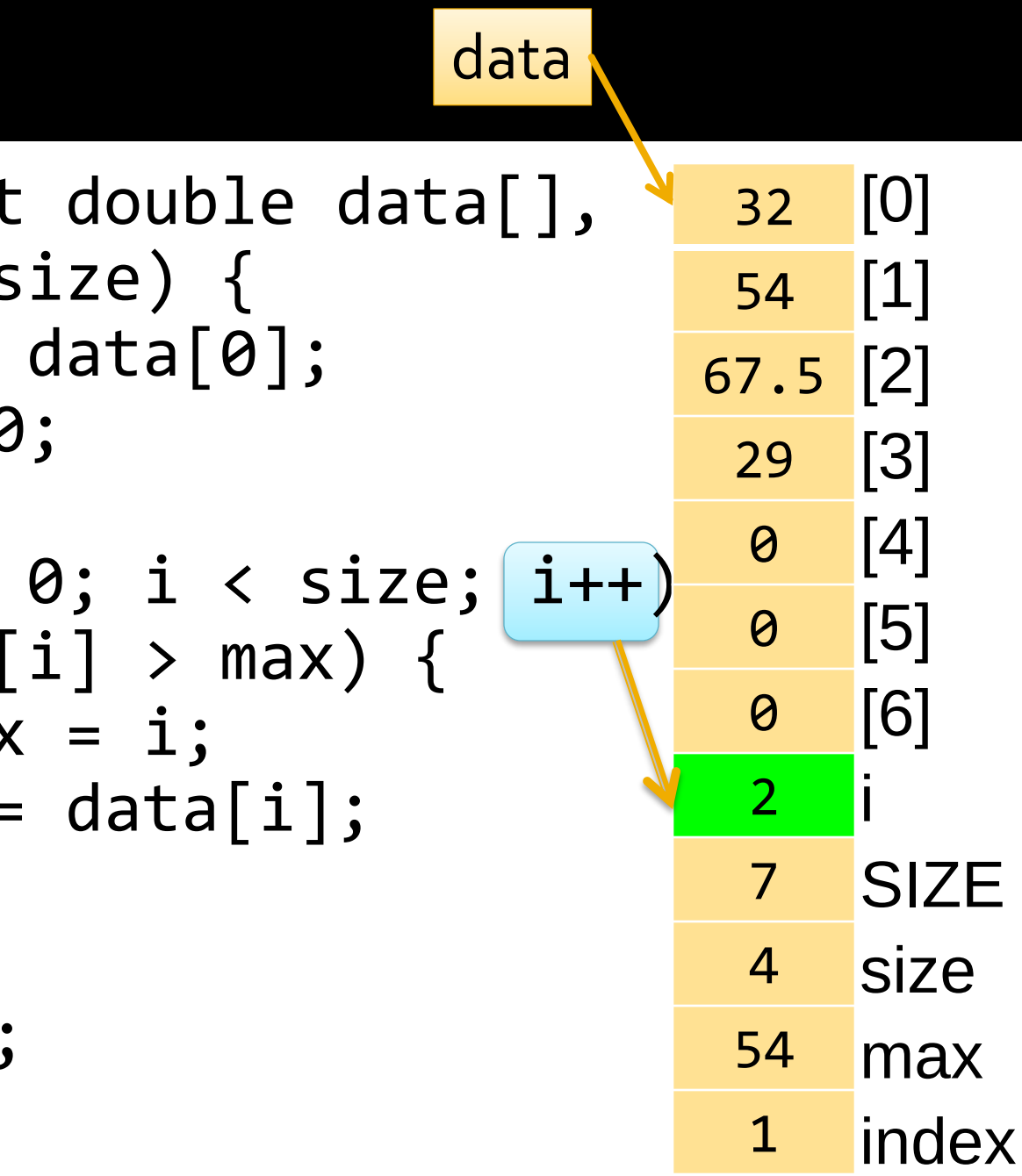
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

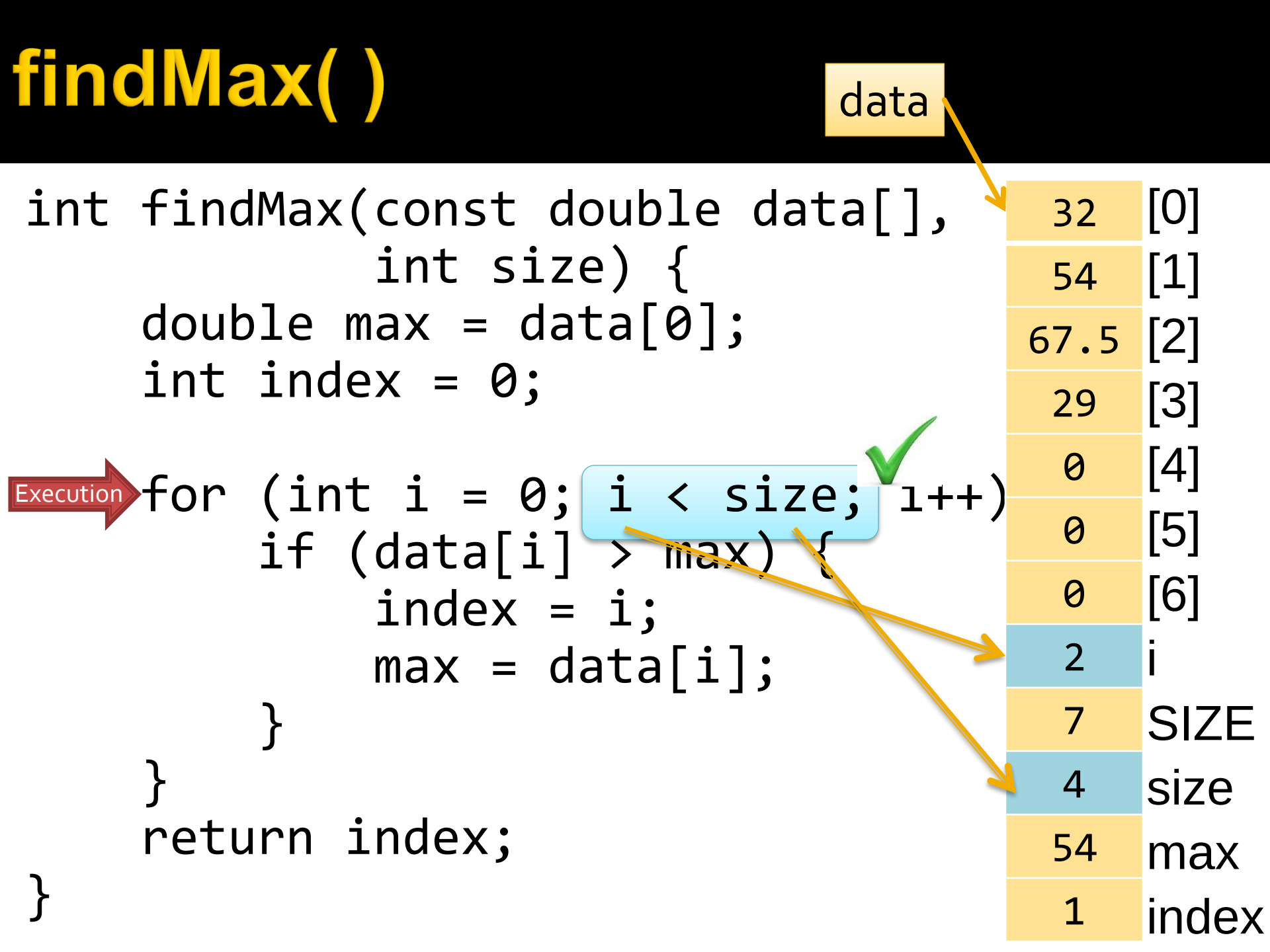
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

Execution



findMax( )

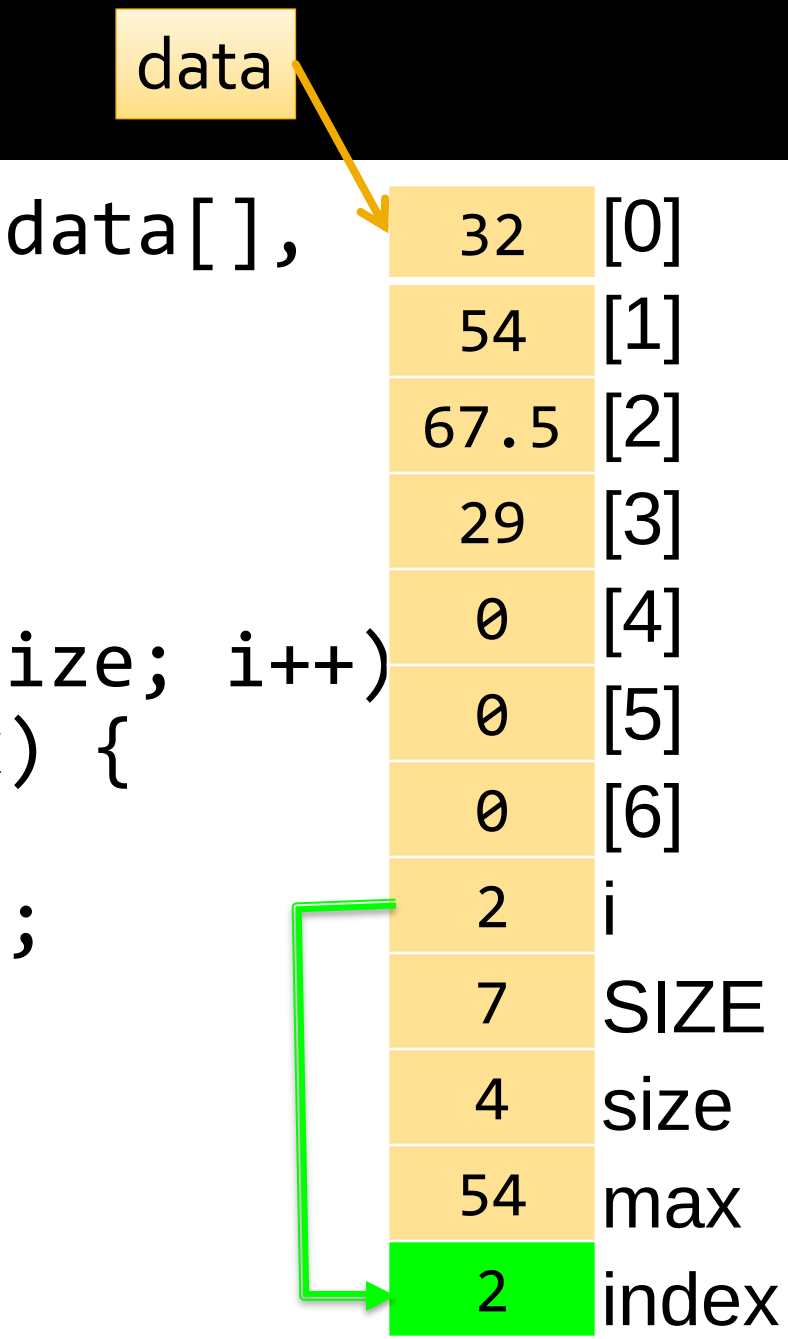
```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < s    ; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

Execution

findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

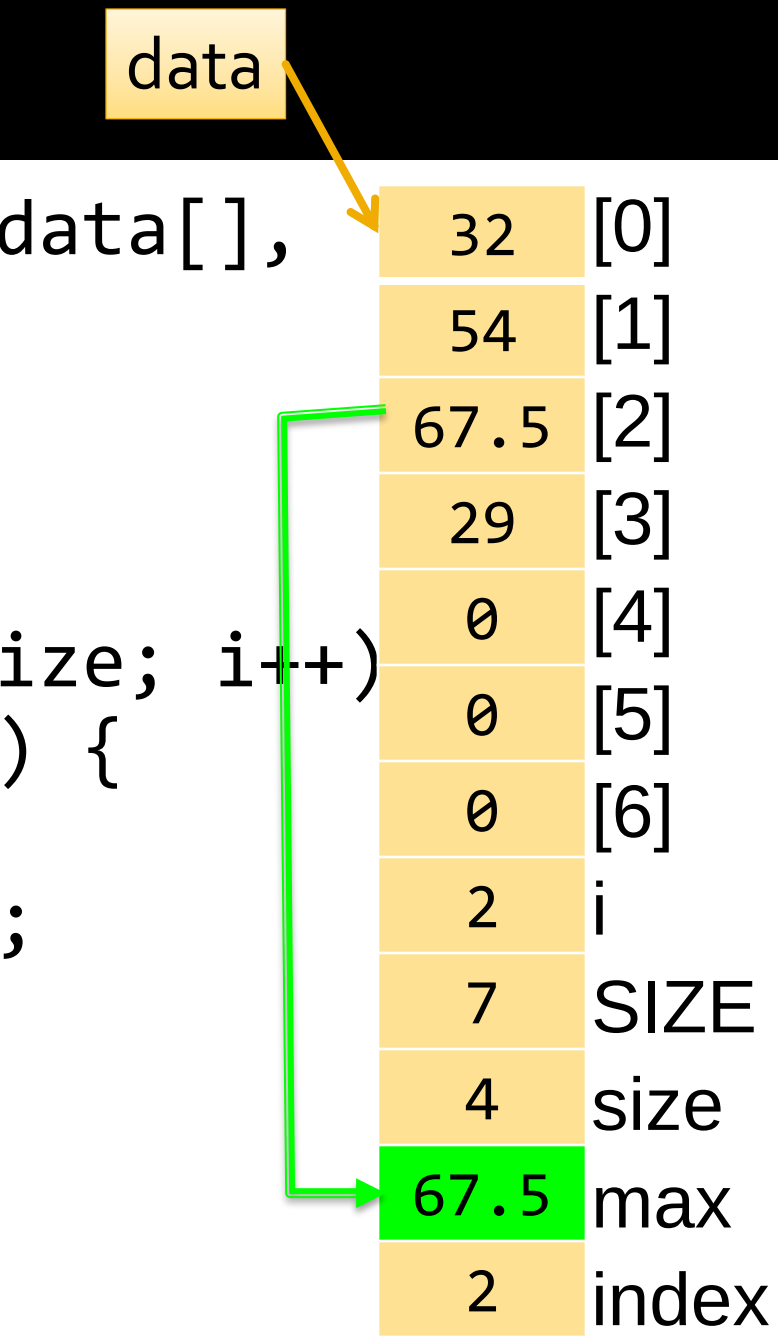
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

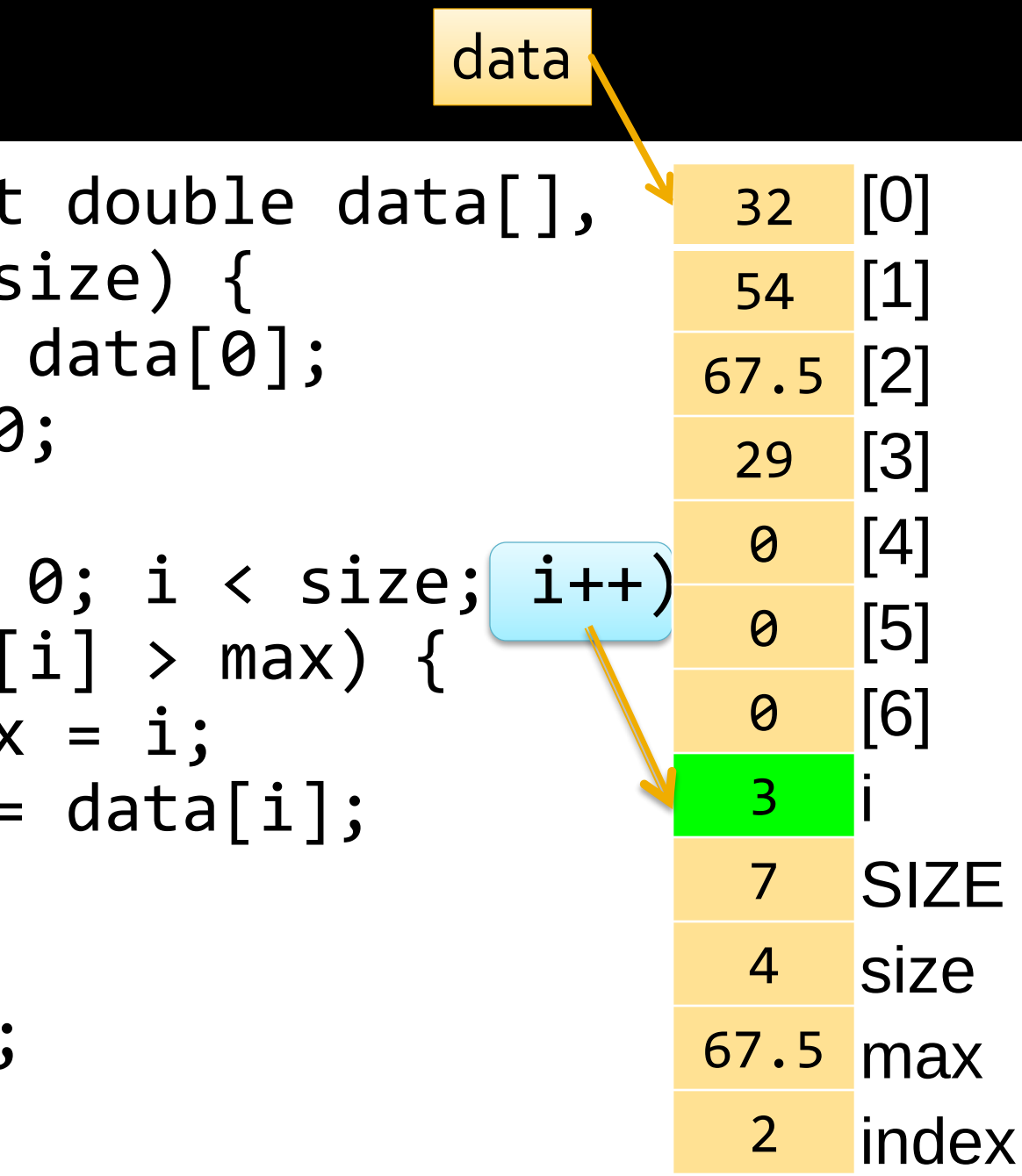
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

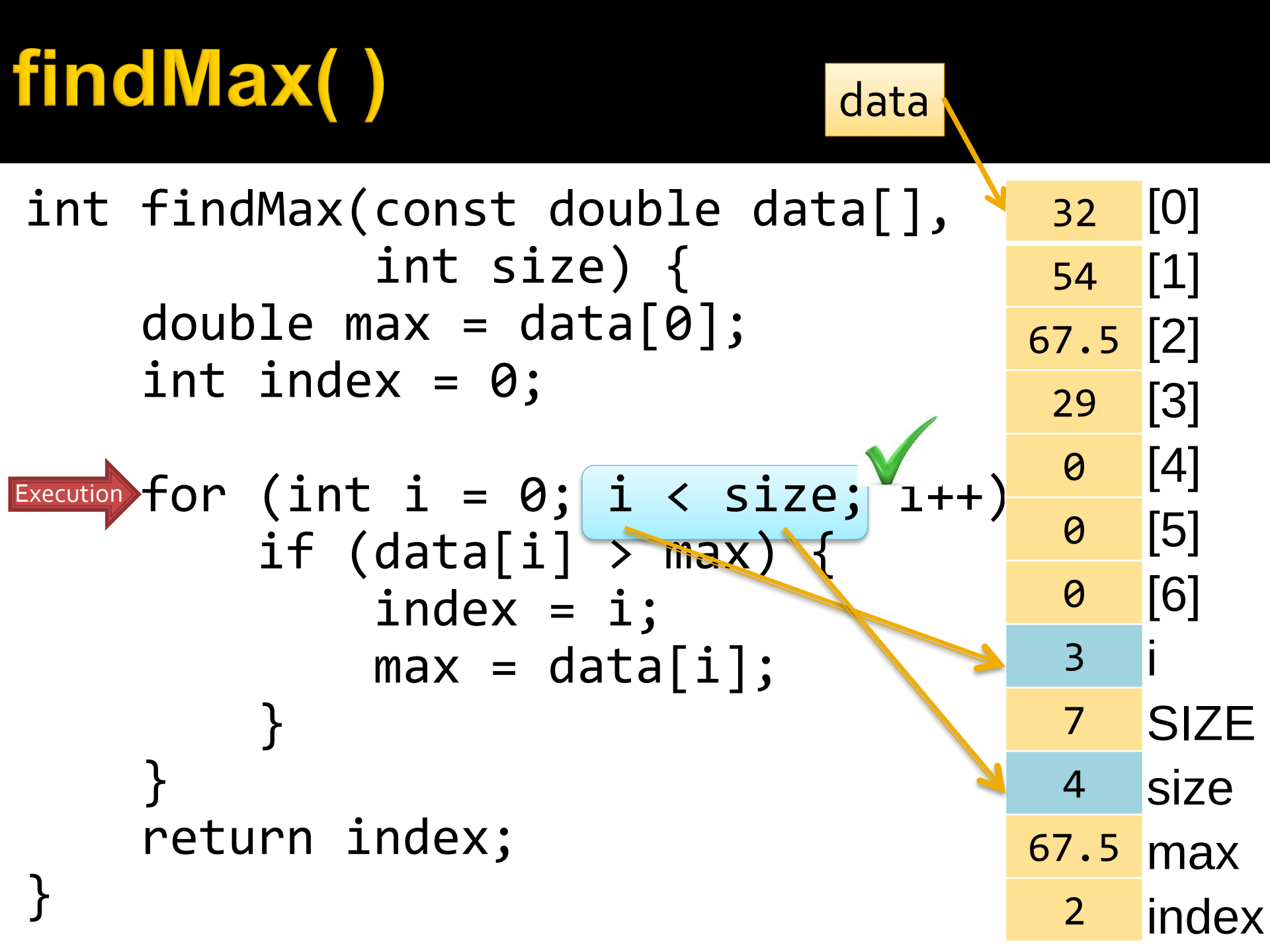
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

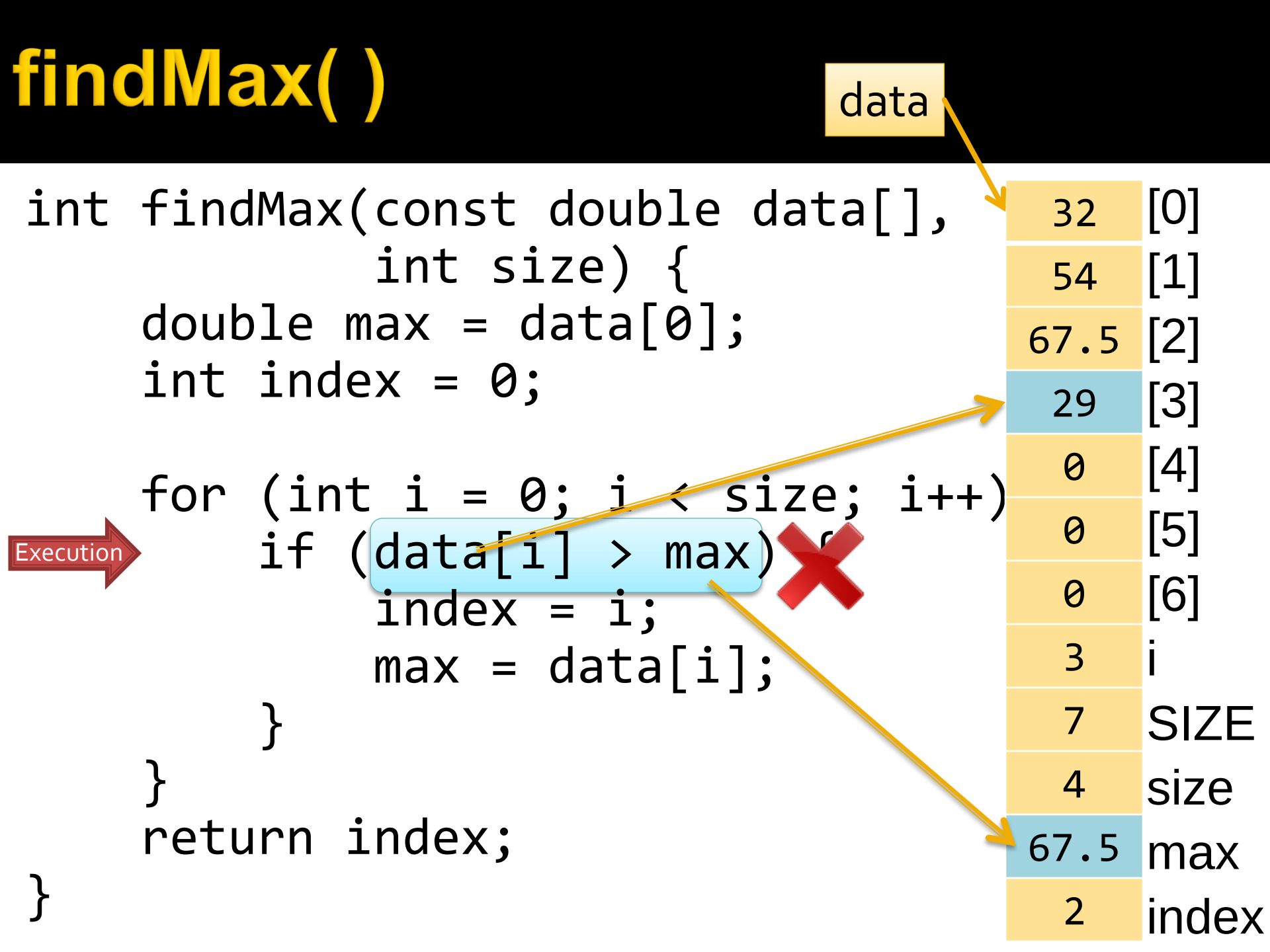
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

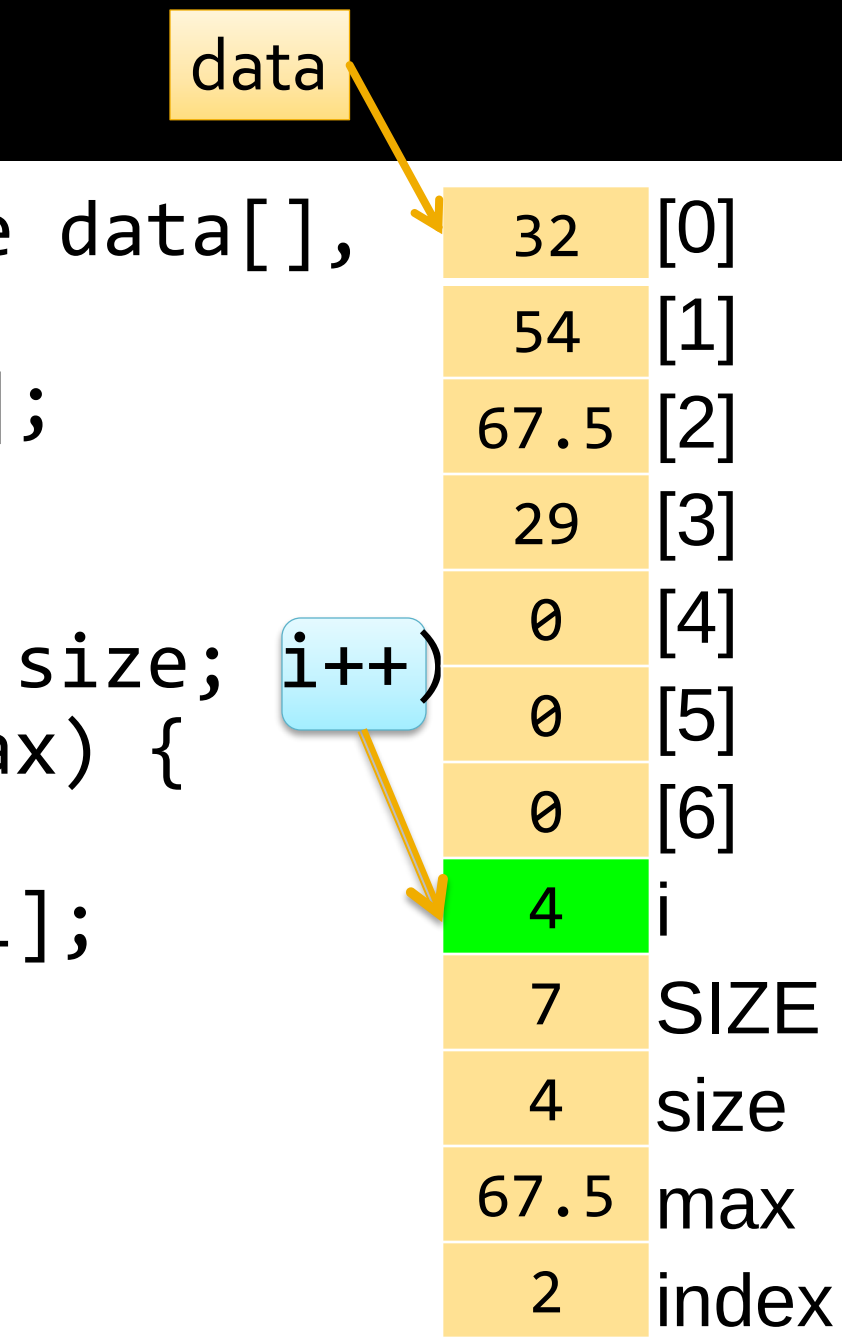
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

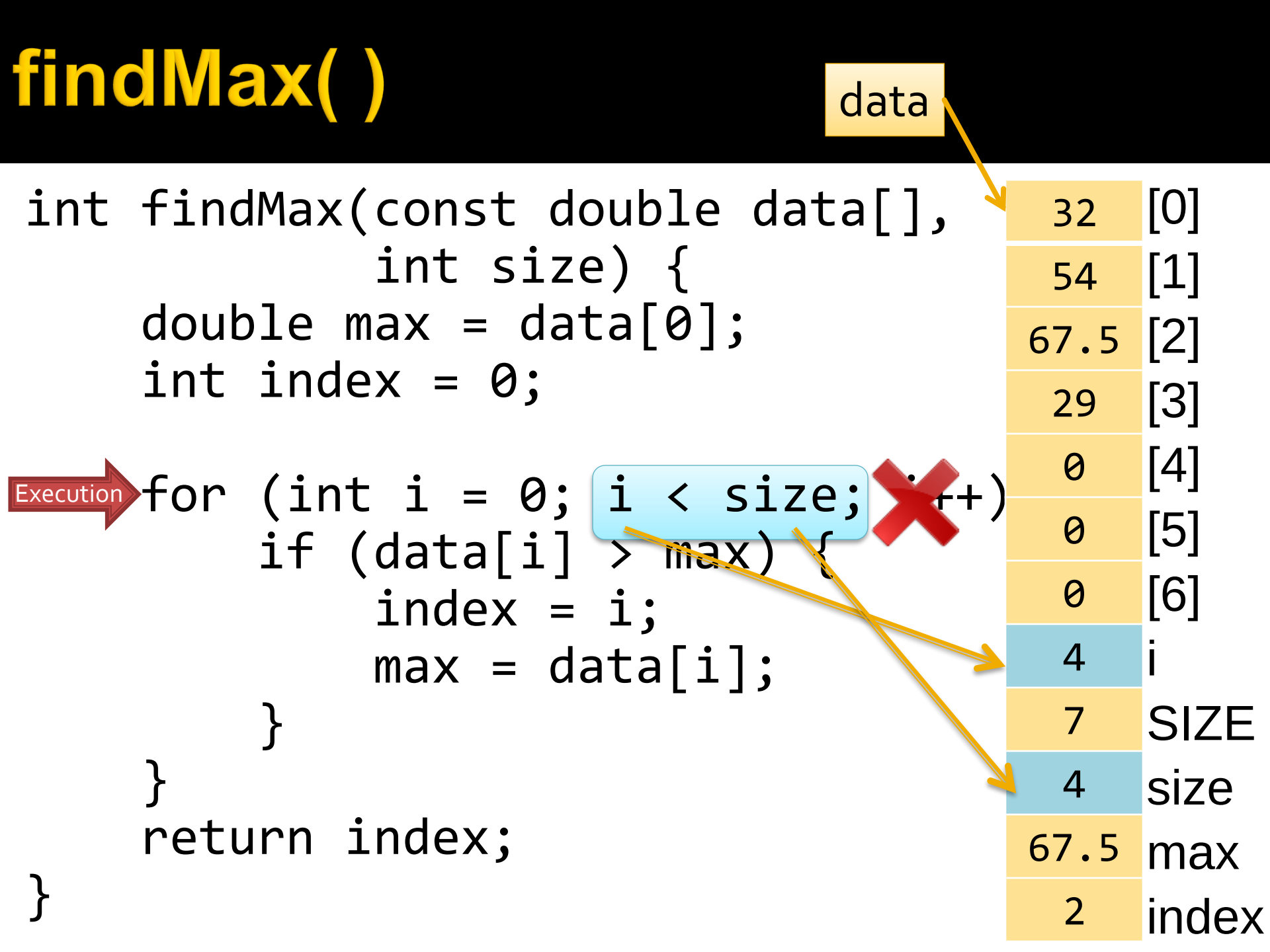
Execution



findMax( )

```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

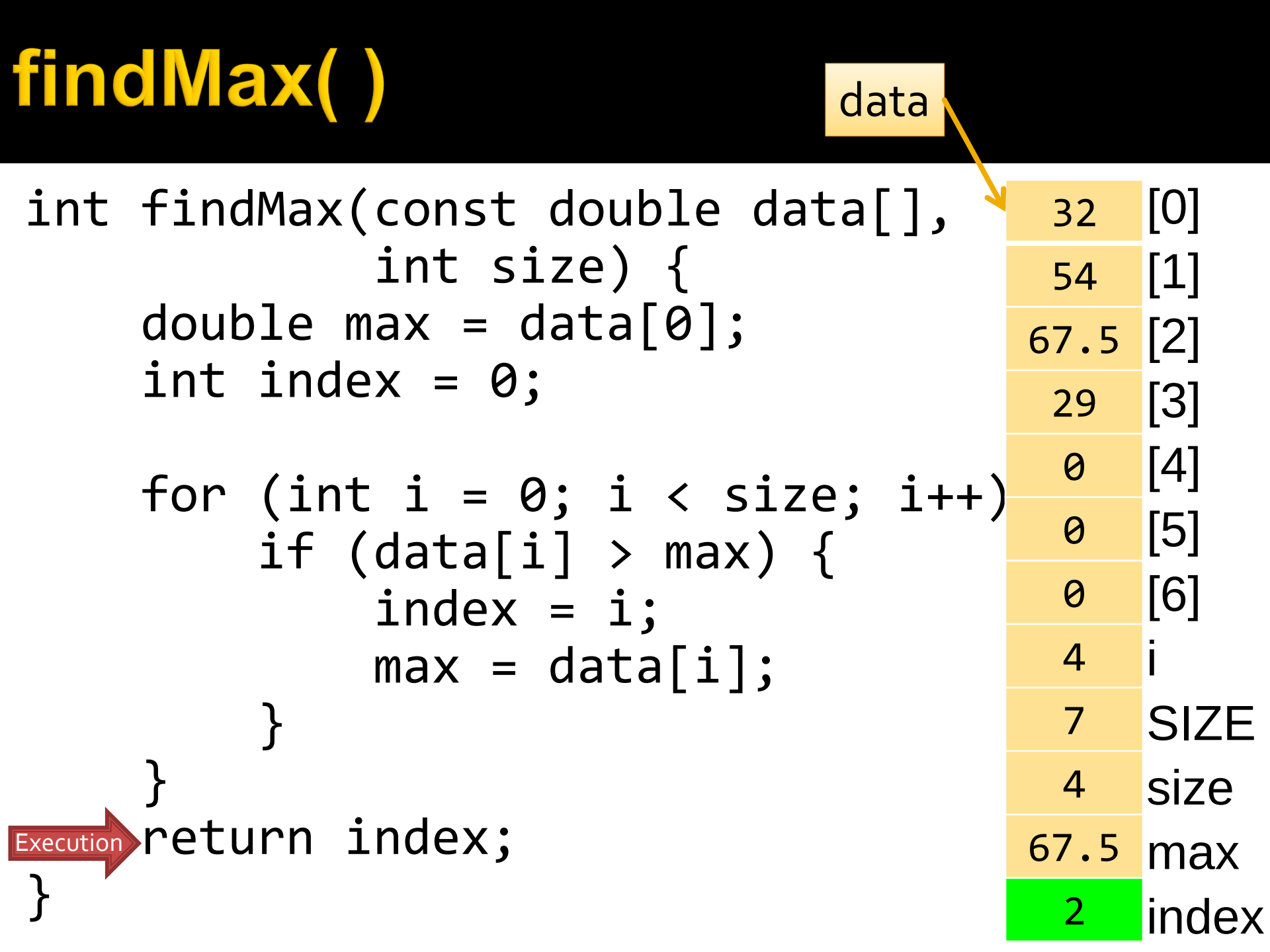
Execution



findMax( )

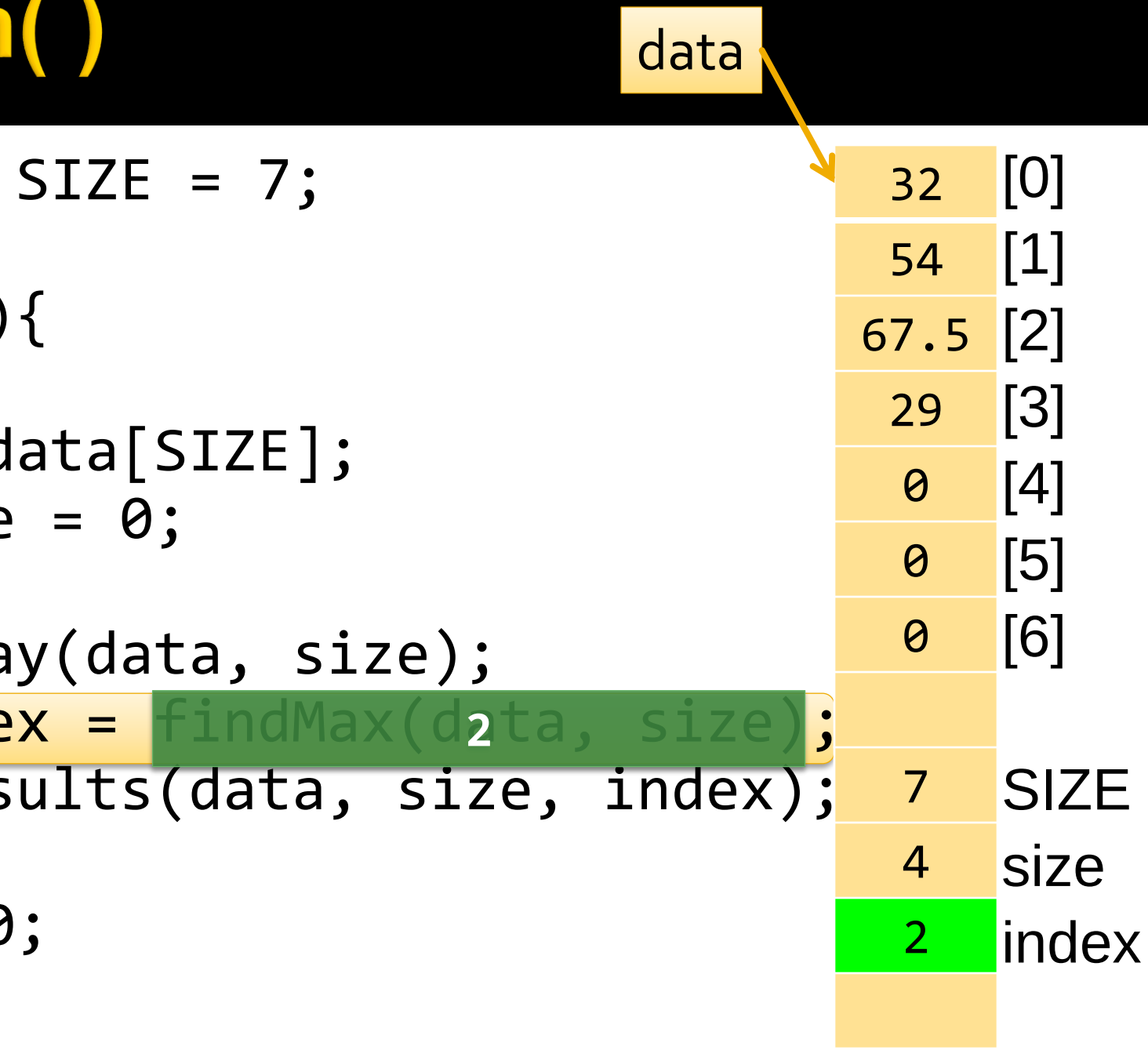
```
int findMax(const double data[],
            int size) {
    double max = data[0];
    int index = 0;

    for (int i = 0; i < size; i++)
        if (data[i] > max) {
            index = i;
            max = data[i];
        }
    }
    return index;
}
```

Execution



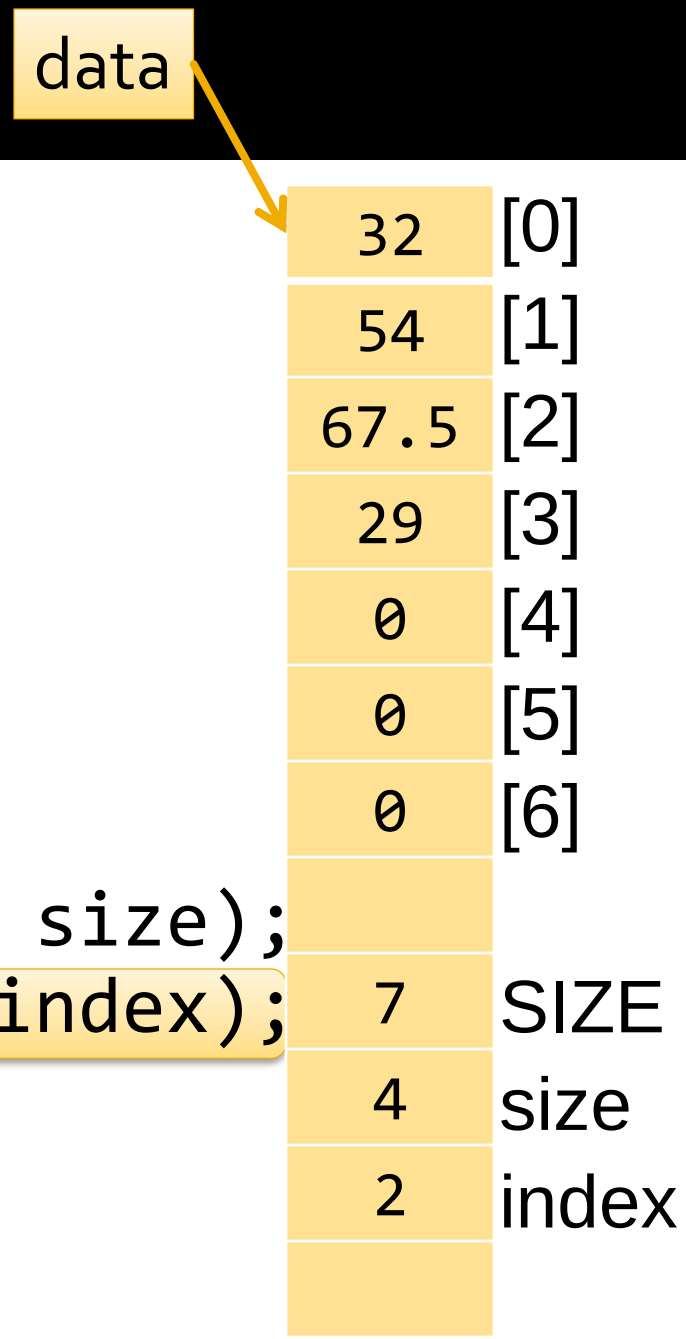
int main( )

```
const int SIZE = 7;

int main(){

   double data[SIZE];
   int size = 0;

   loadArray(data, size);
   int index = findMax(data, size);
   printResults(data, size, index);

   return 0;
}
```



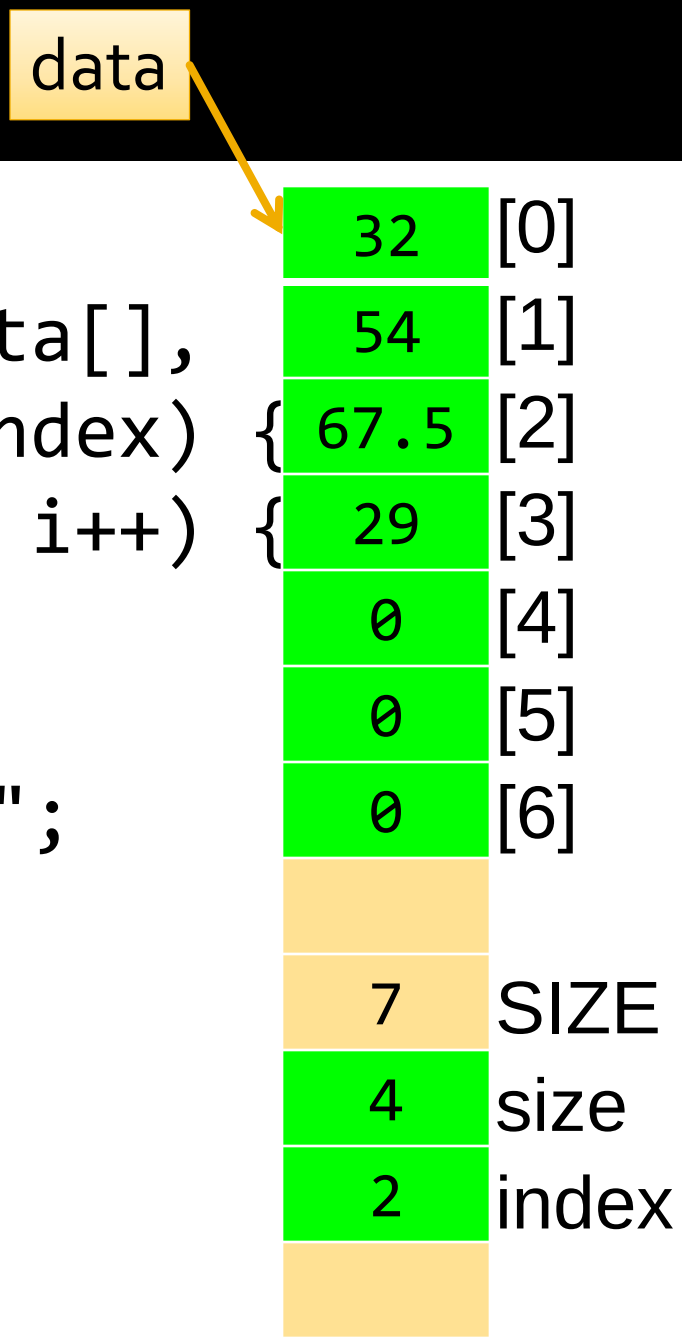
int main( )

```
const int SIZE = 7;

int main(){

   double data[SIZE];
   int size = 0;

   loadArray(data, size);
   int index = findMax(data, size);
   printResults(data, size, index);

   return 0;
}
```

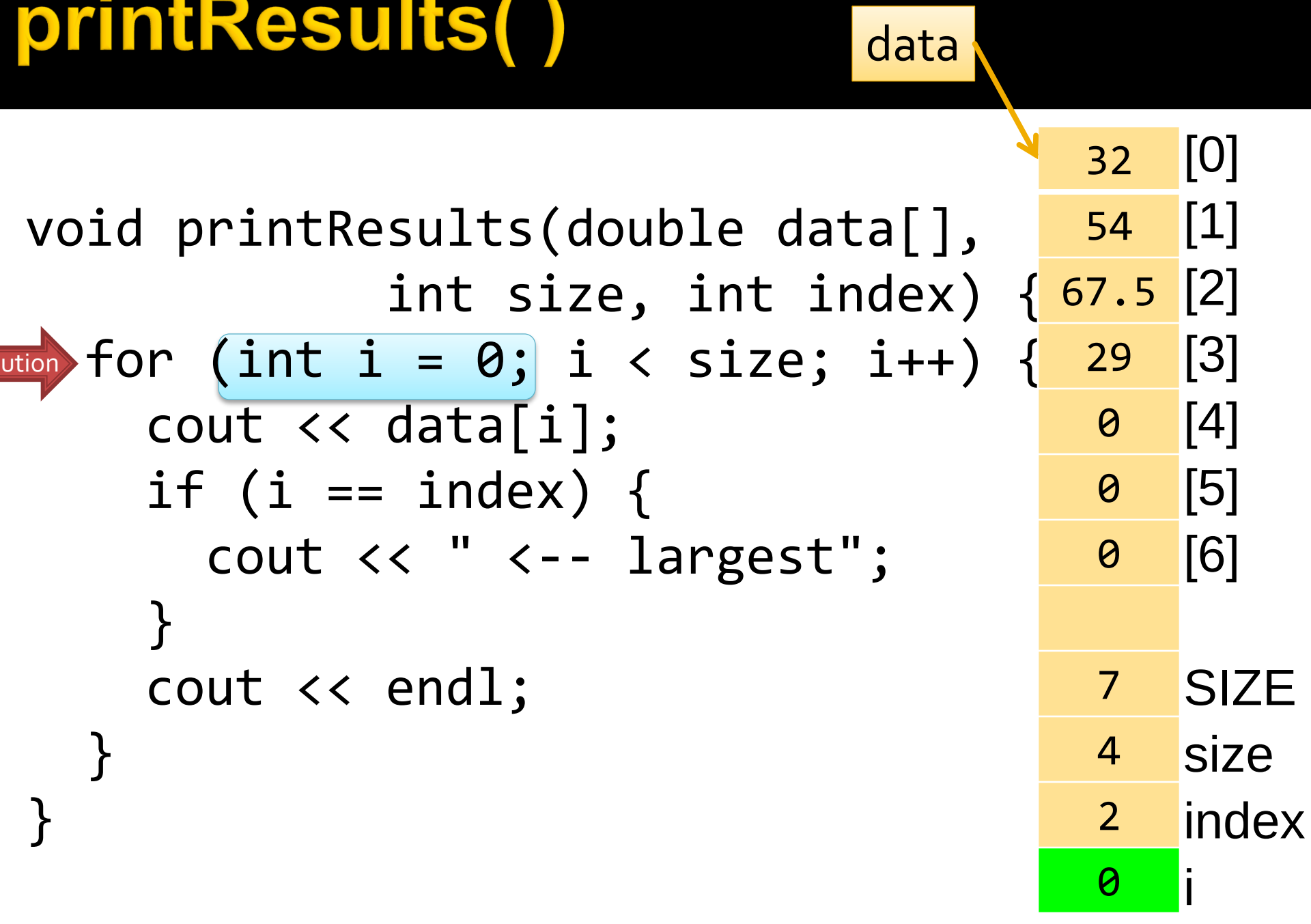


printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

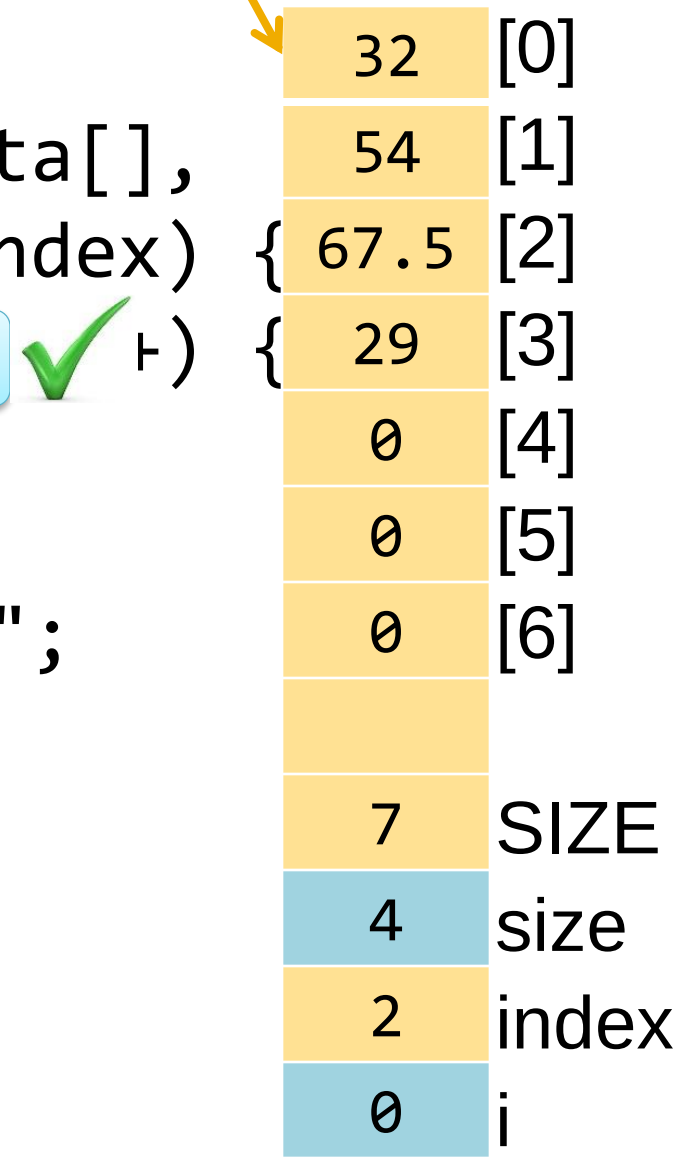


printResults( )



printResults( )

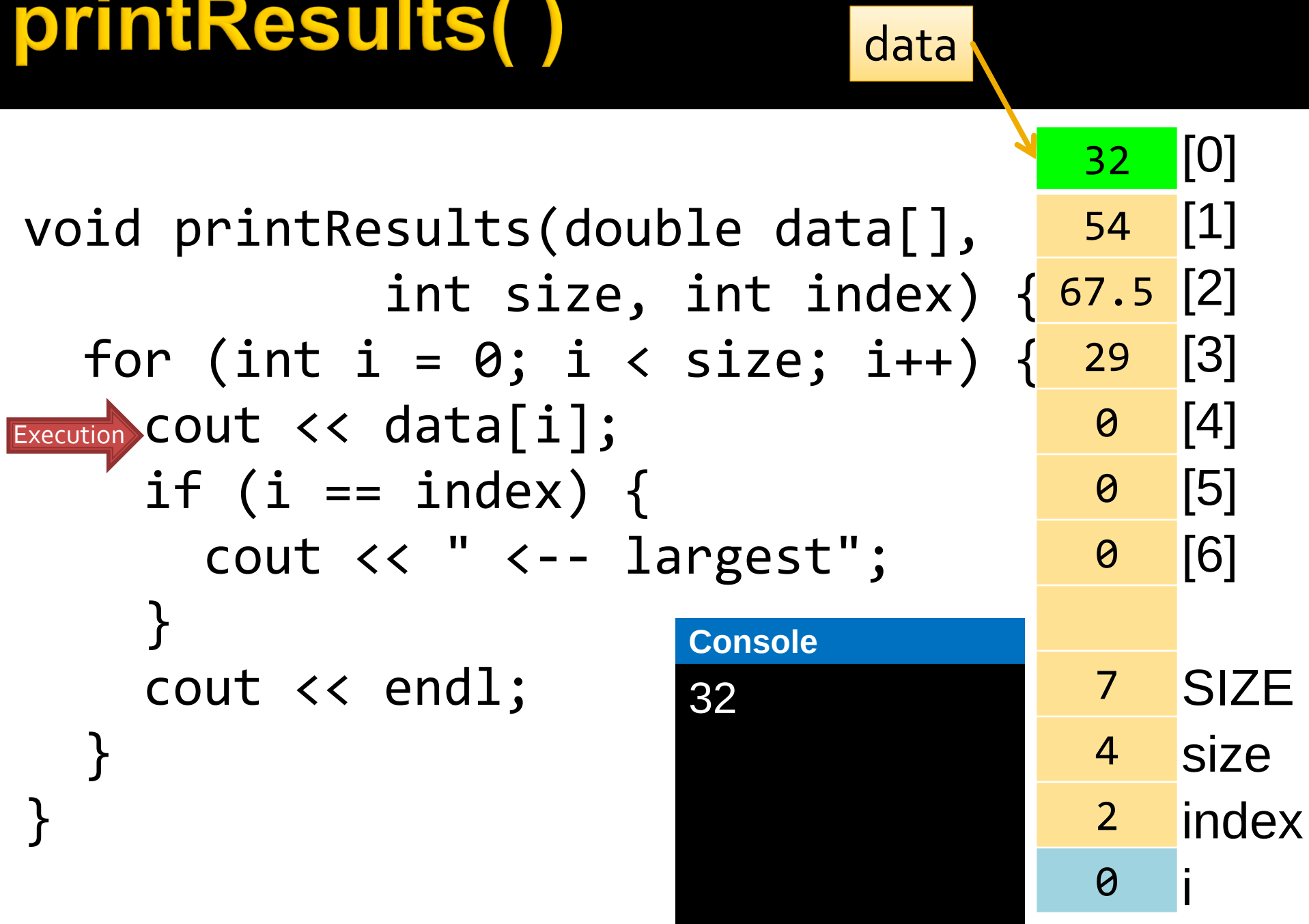
```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```



printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution



printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution

data

32	[0]
54	[1]
67.5	[2]
29	[3]
0	[4]
0	[5]
0	[6]
7	SIZE
4	size
2	index
0	i

Console

```
32
```

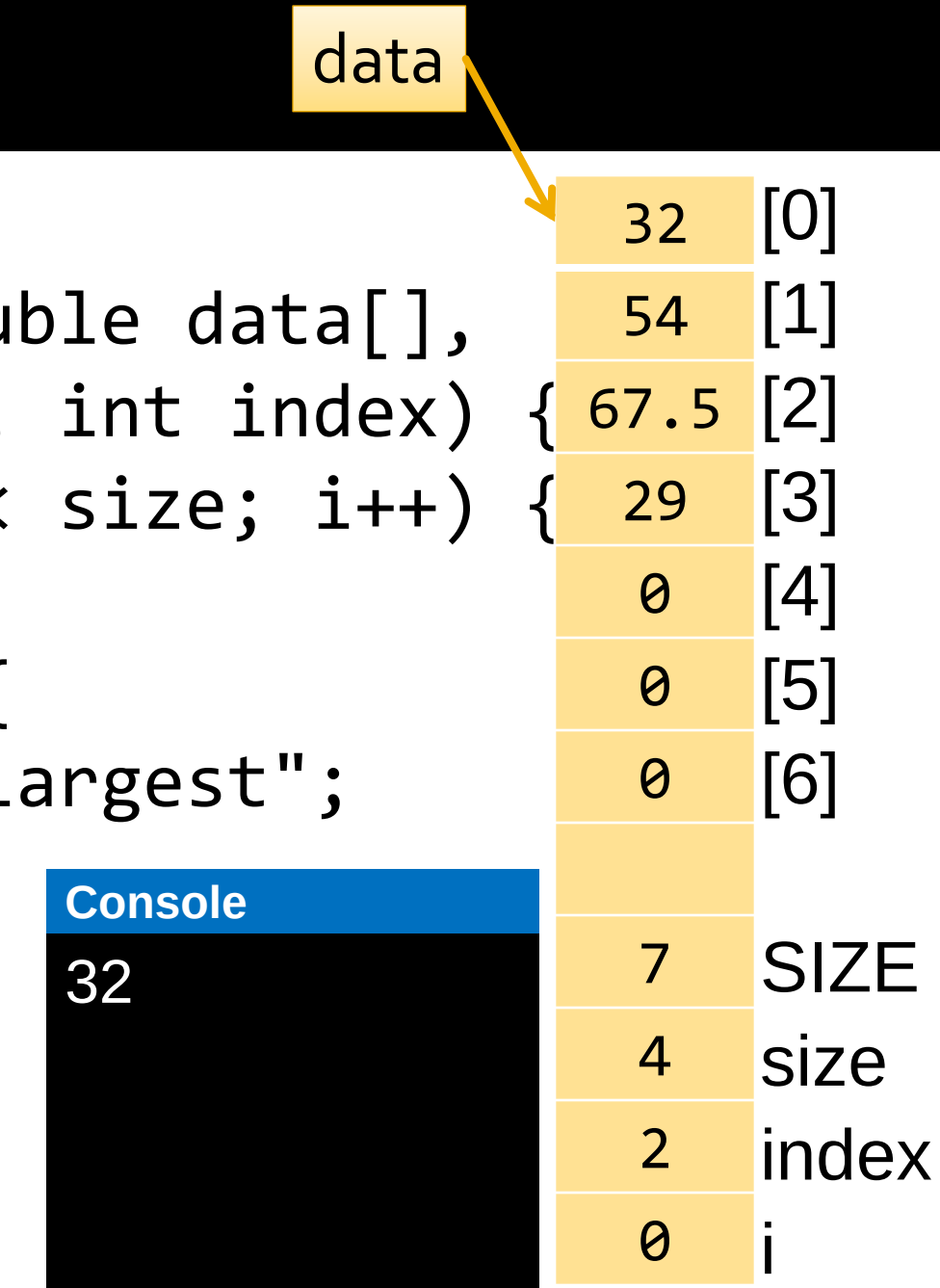
printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution

Console

```
32
```

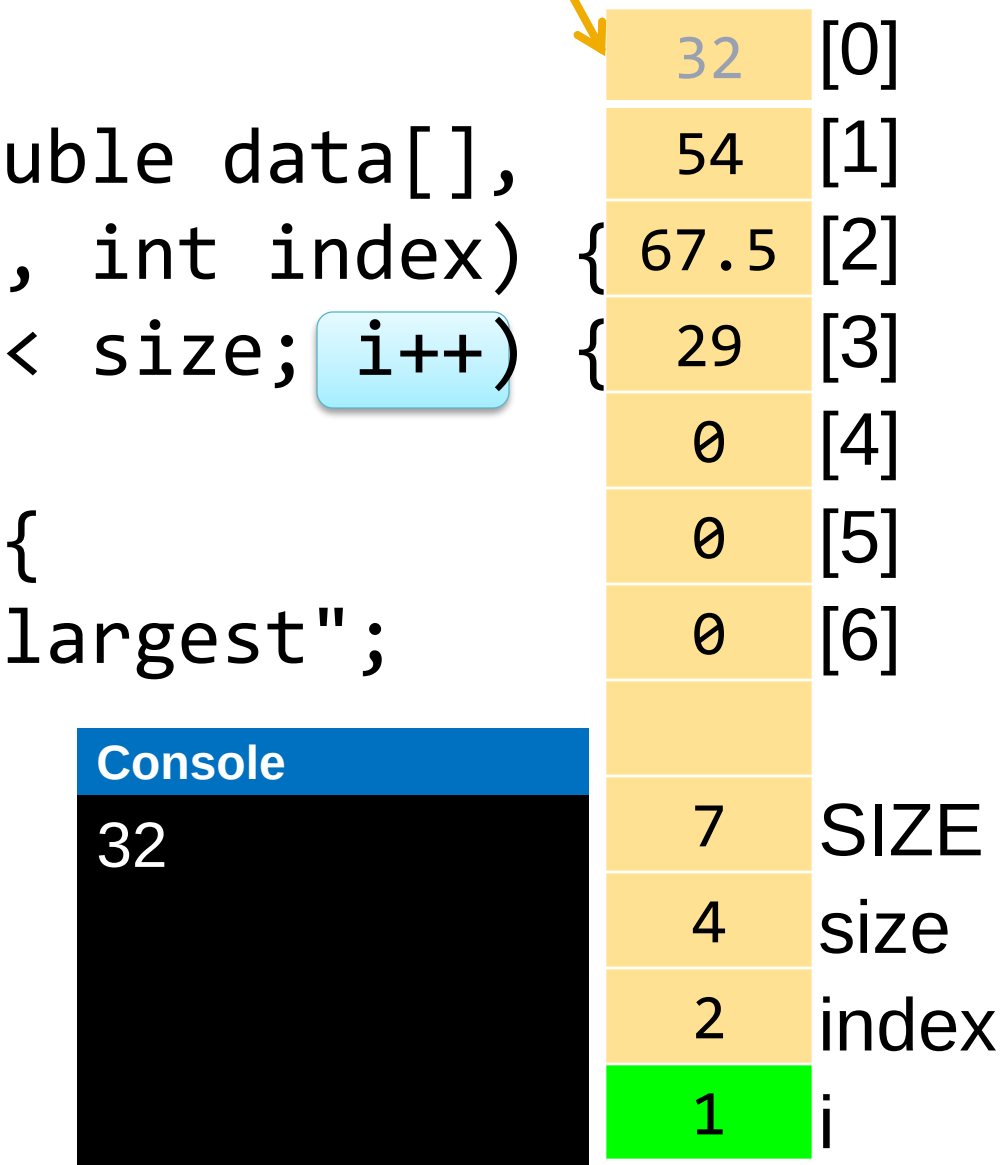


printResults( )

```
void printResults(double data[],
                  int size, int index) {
ution for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Console

```
32
```

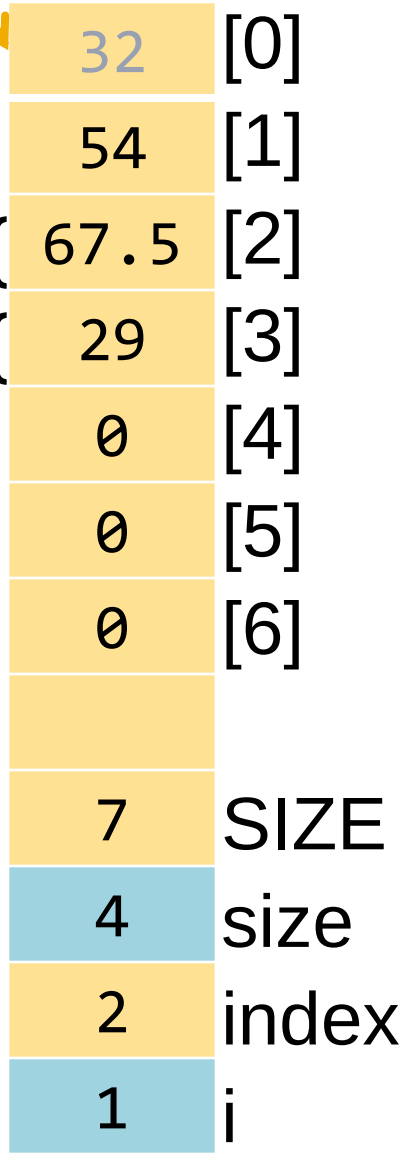


printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Console

```
32
```



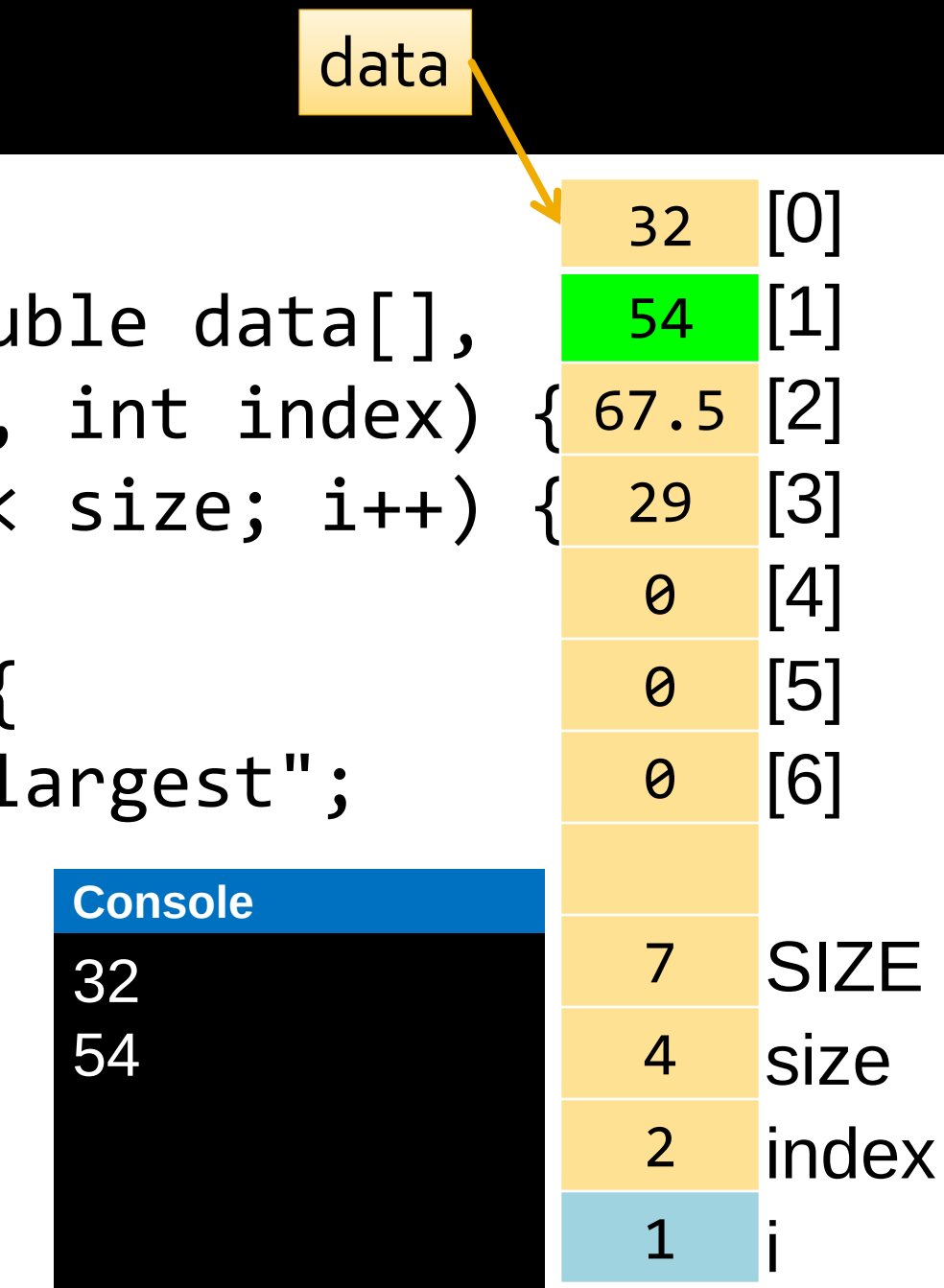
printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution

Console

```
32
54
```



printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution

data

32	[0]
54	[1]
67.5	[2]
29	[3]
0	[4]
0	[5]
0	[6]
7	SIZE
4	size
2	index
1	i

Console

```
32
54
```

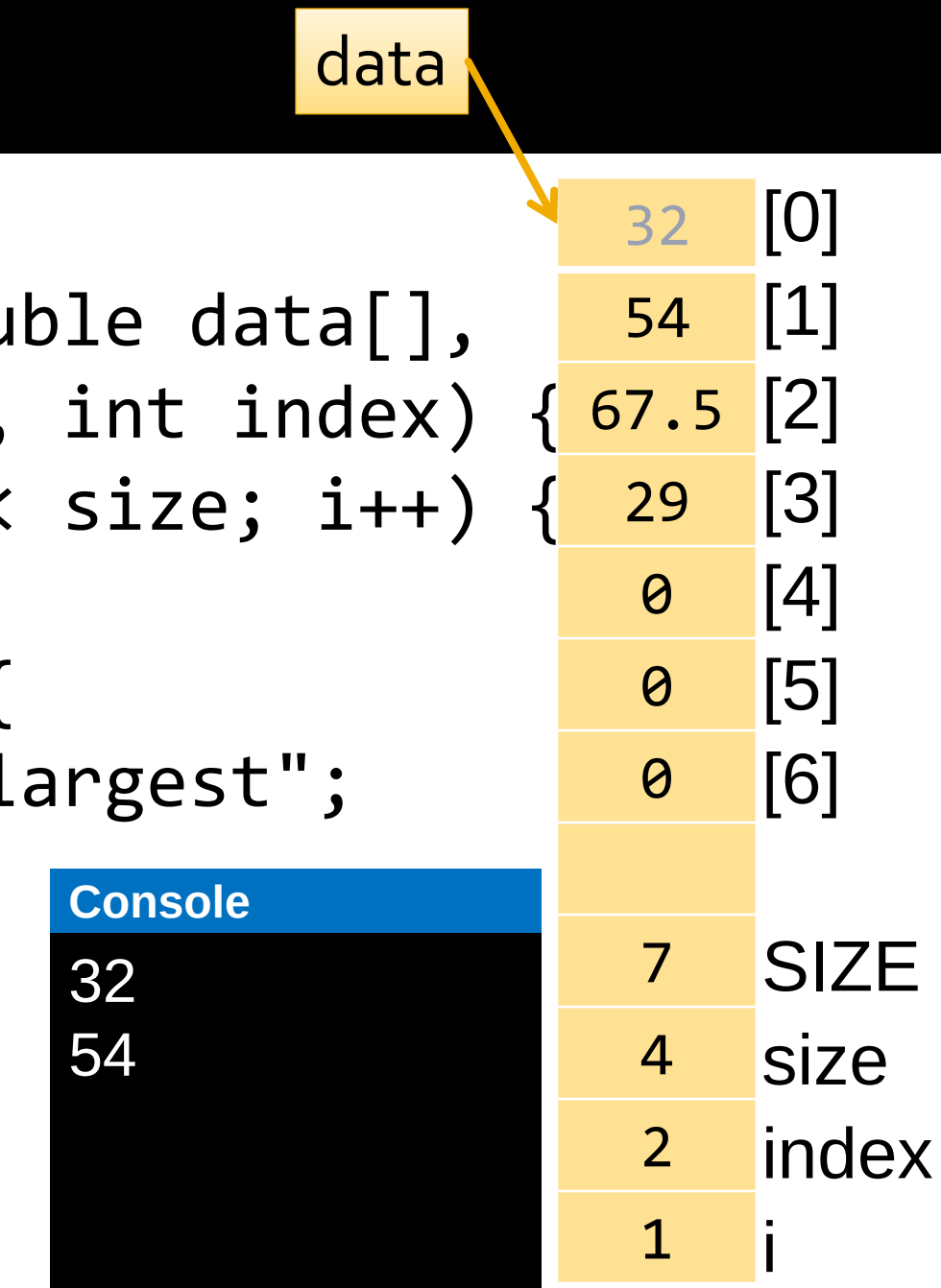
printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution

Console

```
32
54
```

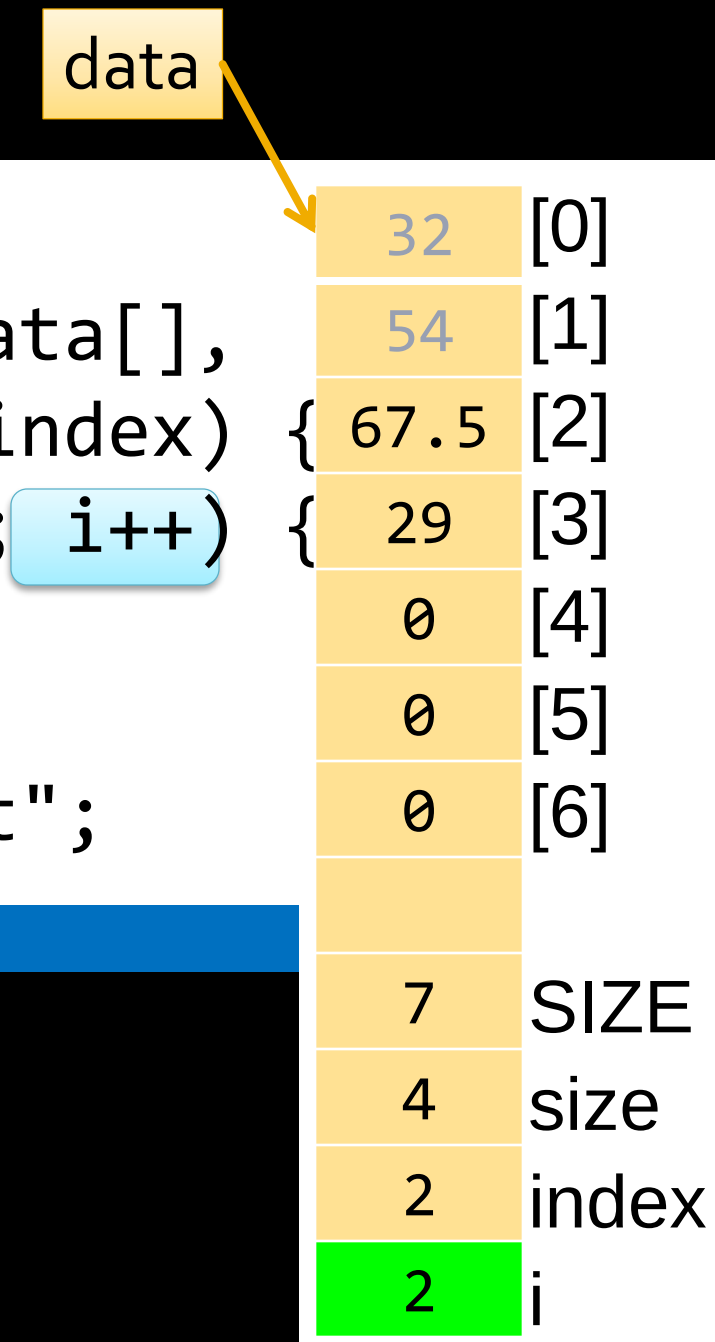


printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Console

```
32
54
```



printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Console

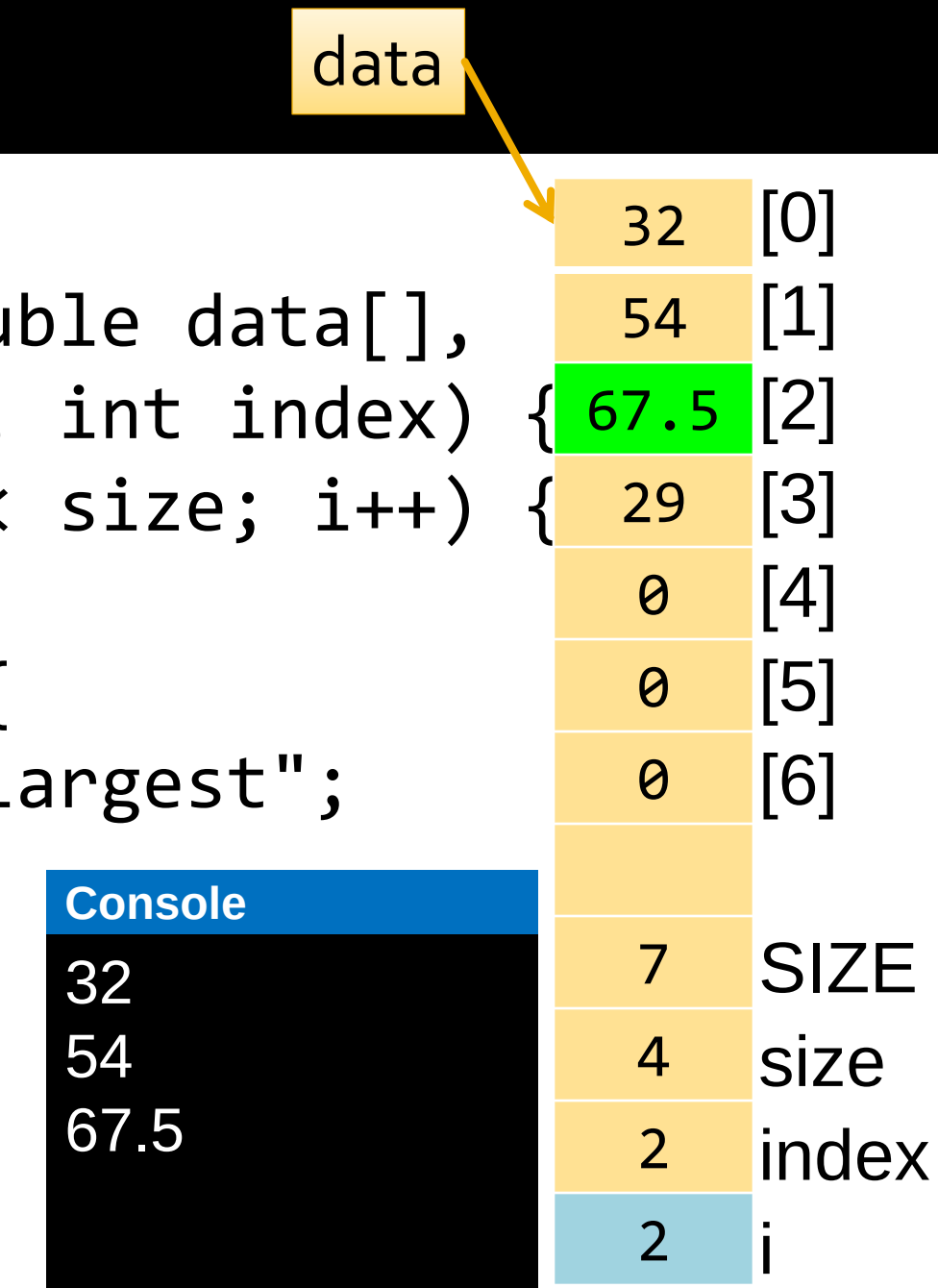
```
32
54
```

data	
32	[0]
54	[1]
67.5	[2]
29	[3]
0	[4]
0	[5]
0	[6]
7	SIZE
4	size
2	index
2	i

printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution



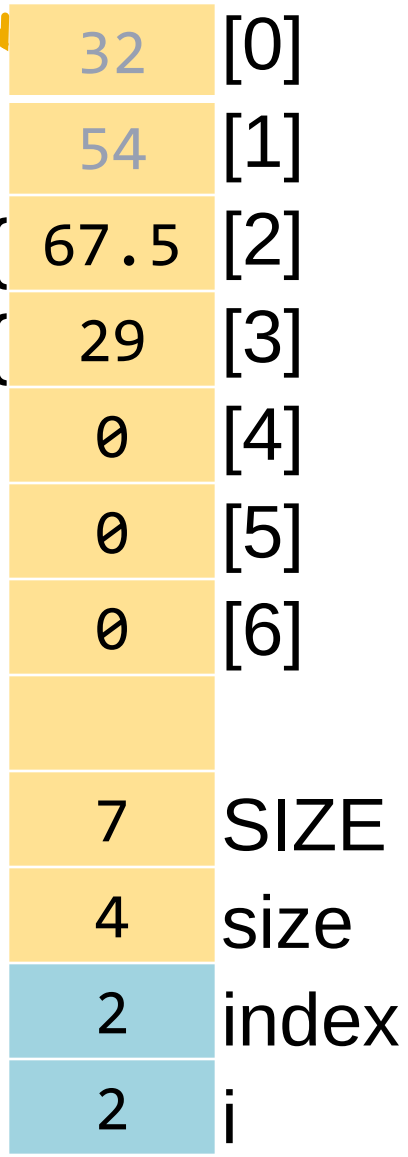
printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index)
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution

Console

```
32
54
67.5
```



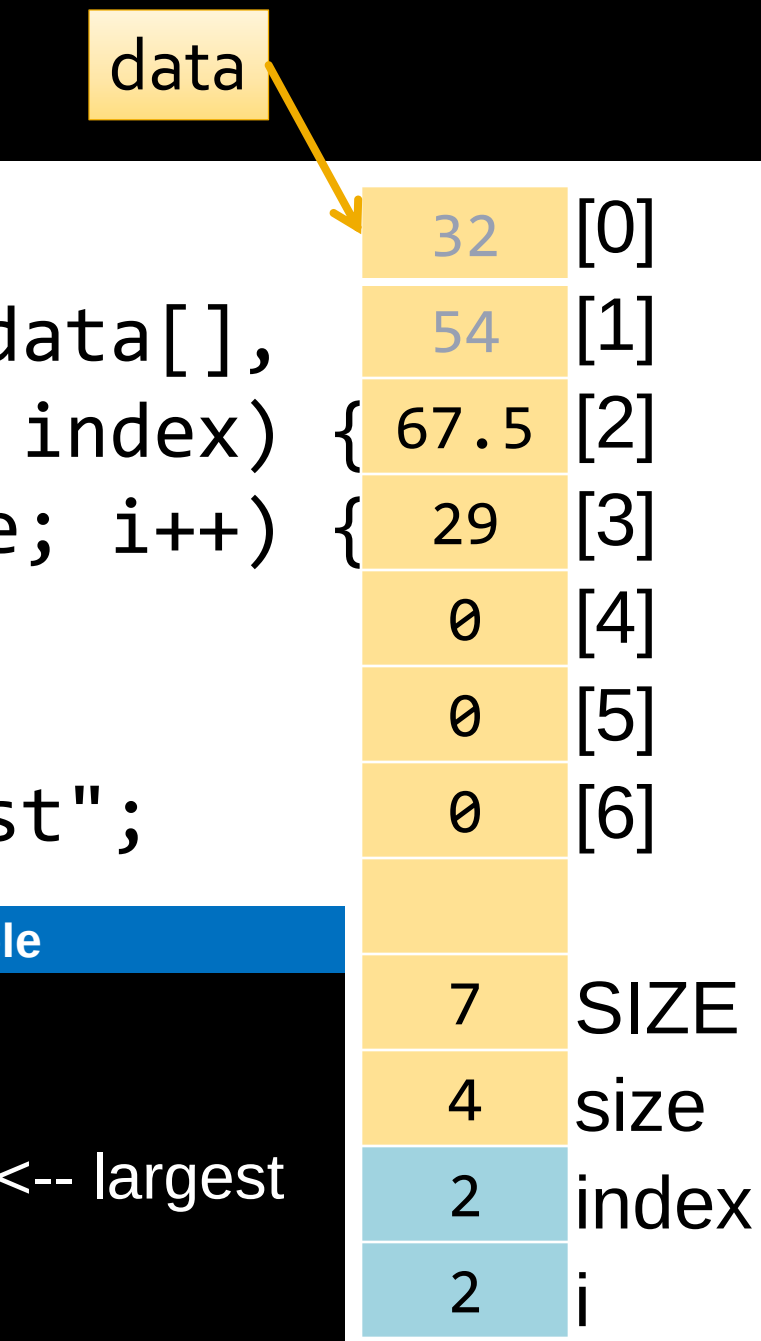
printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution

Console

```
32
54
67.5 <-- largest
```



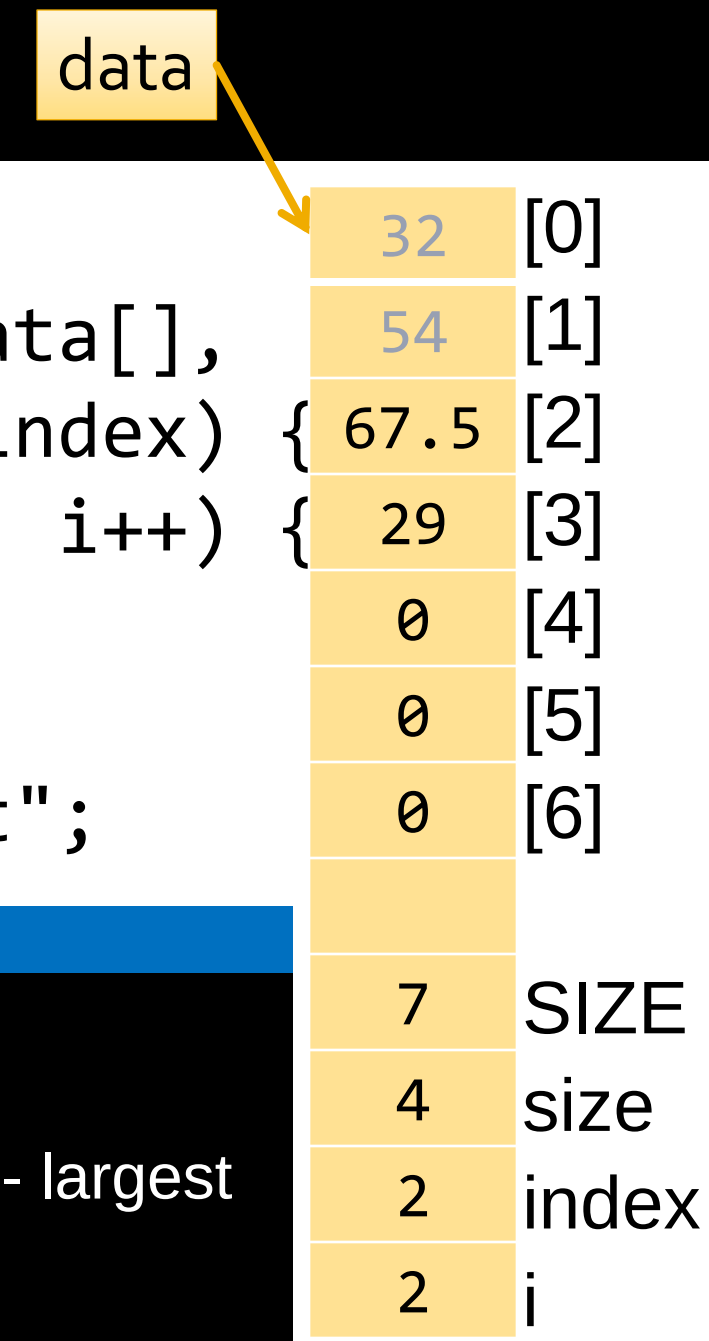
printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution

Console

```
32
54
67.5 <-- largest
```

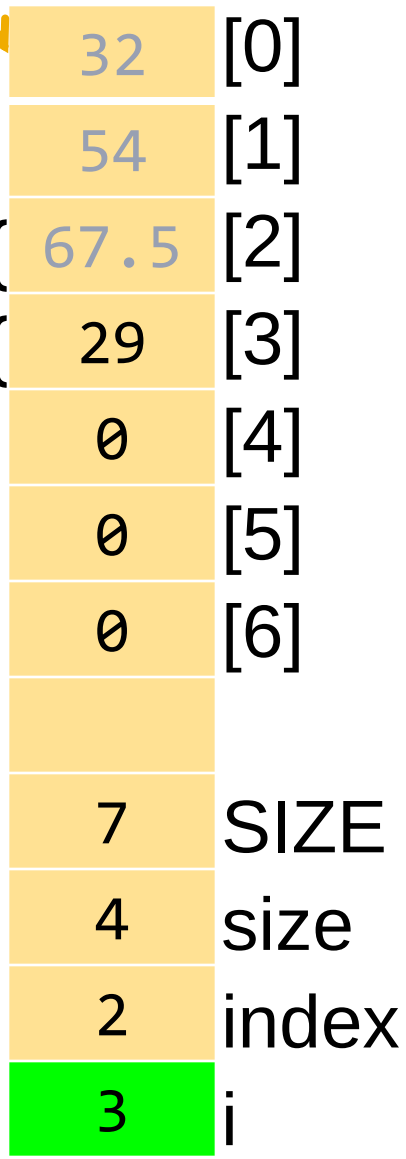


printResults( )

```
void printResults(double data[],
                  int size, int index) {
ution for (int i = 0; i < size;  i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Console

```
32
54
67.5 <-- largest
```

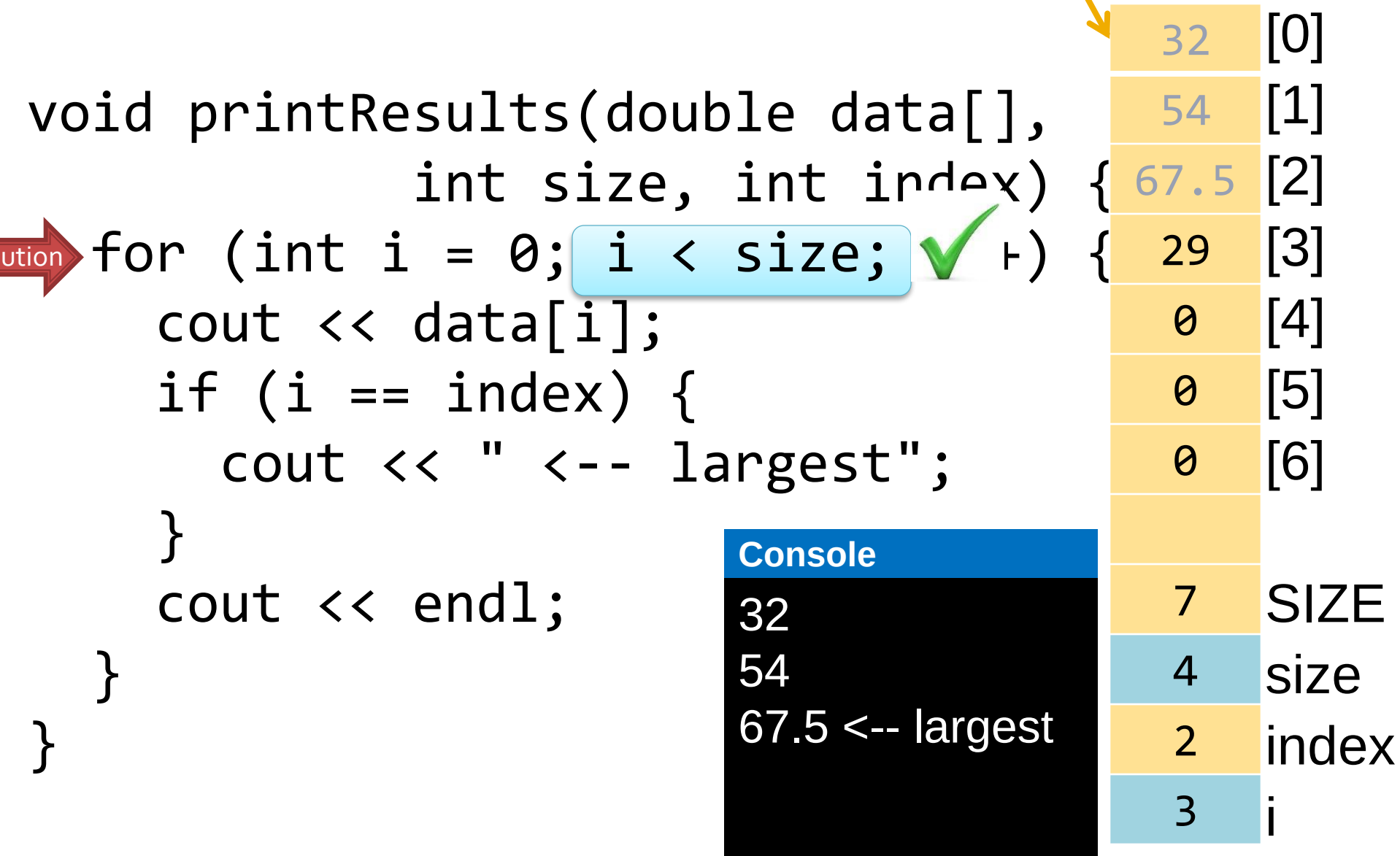


printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Console

```
32
54
67.5 <-- largest
```



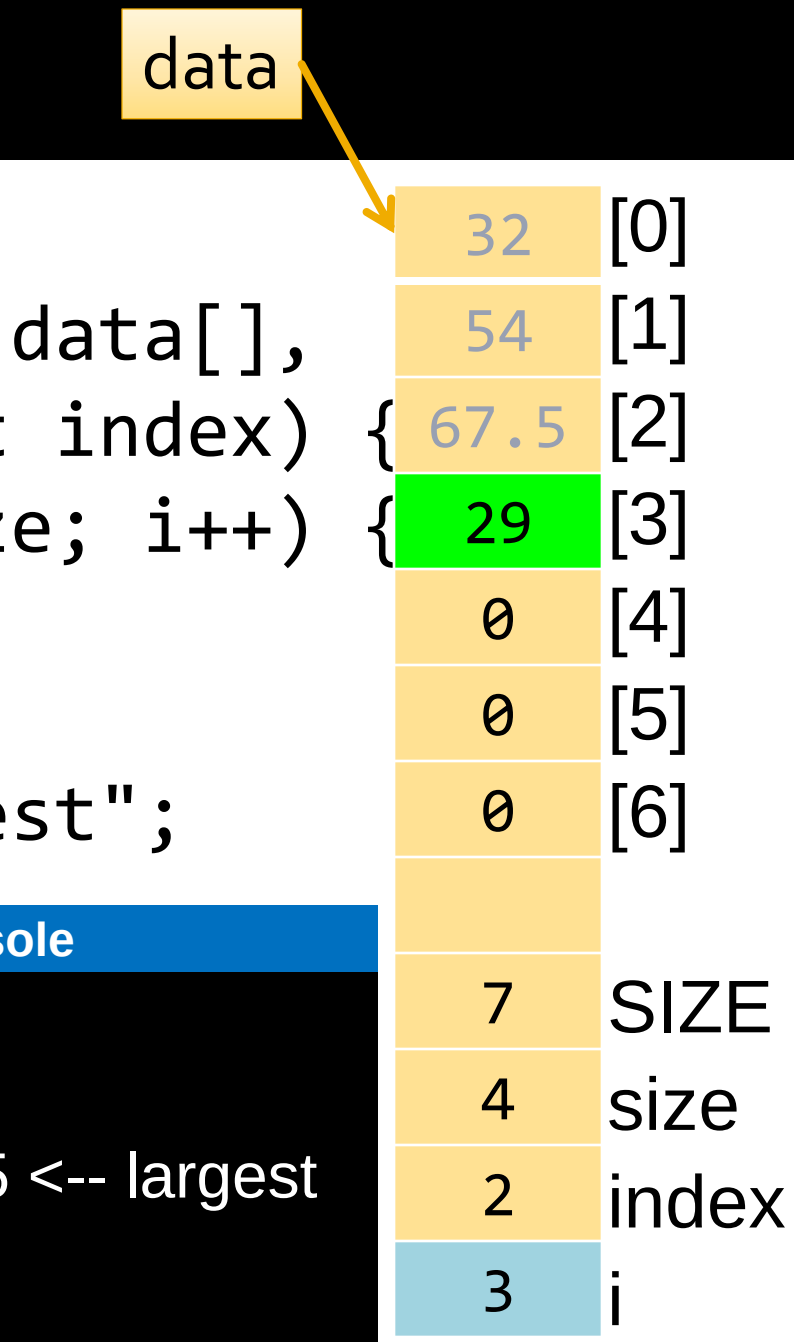
printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution

Console

```
32
54
67.5 <-- largest
29
```



printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution

data

32	[0]
54	[1]
67.5	[2]
29	[3]
0	[4]
0	[5]
0	[6]
7	SIZE
4	size
2	index
3	i

Console

```
32
54
67.5 <-- largest
29
```

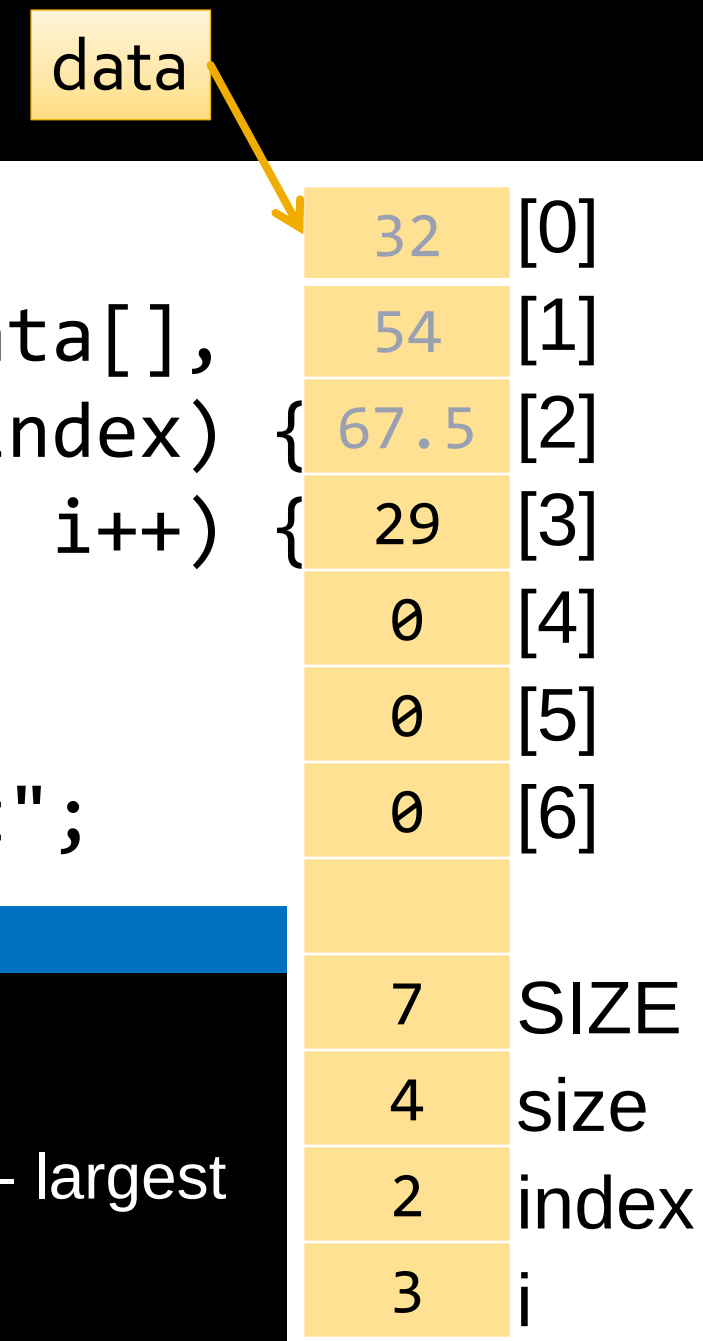
printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

Execution

Console

```
32
54
67.5 <-- largest
29
```

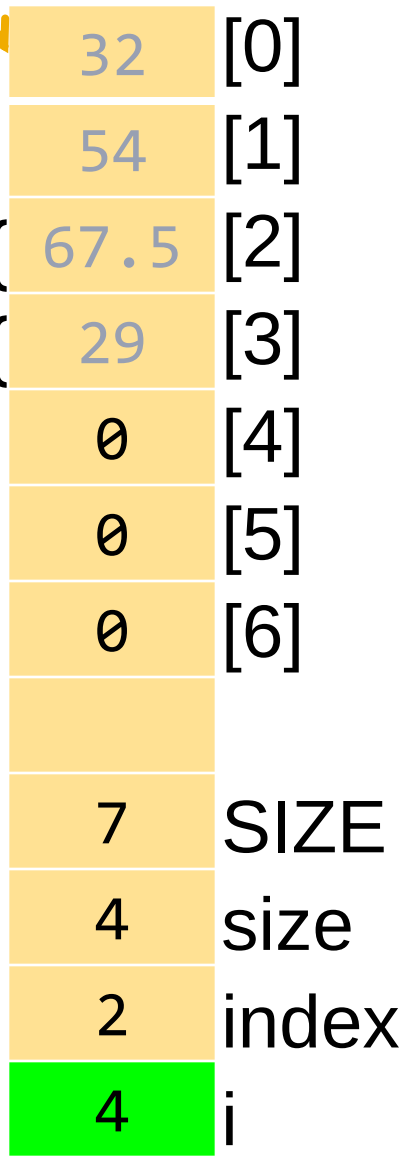


printResults( )

```
void printResults(double data[],
                  int size, int index) {
ution for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```

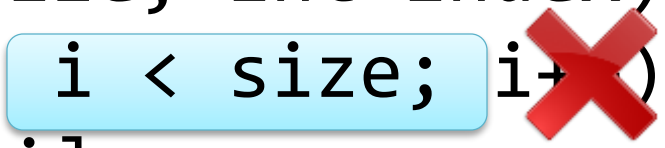
Console

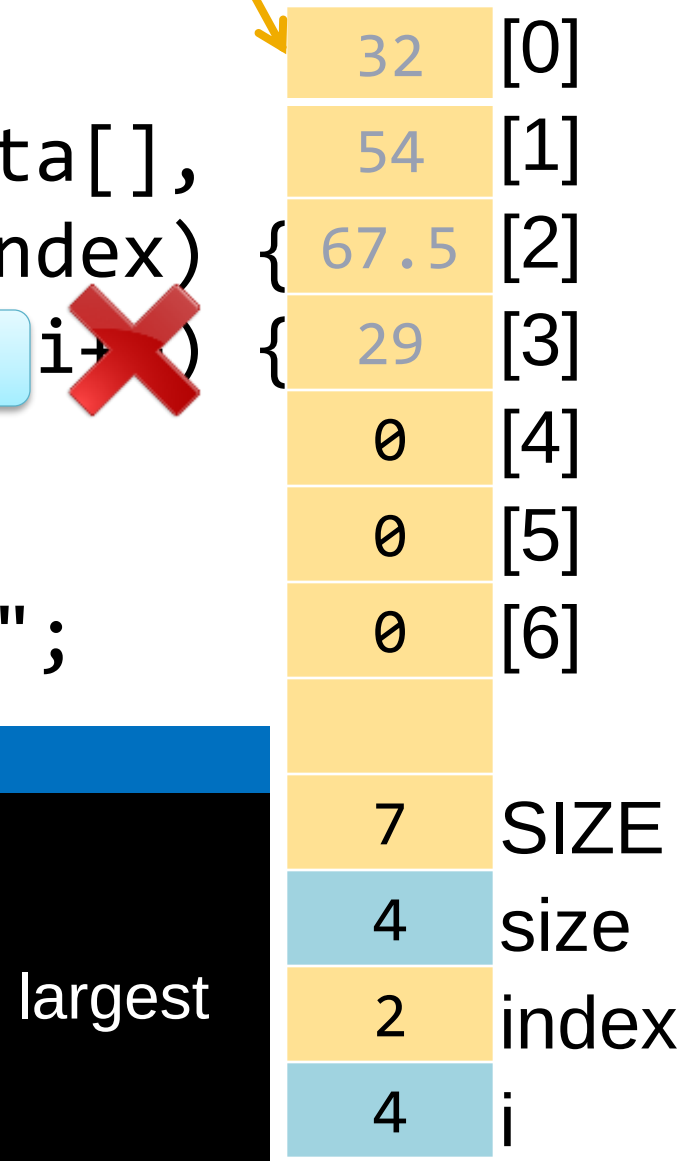
```
32
54
67.5 <-- largest
29
```



printResults( )

```
void printResults(double data[],
                  int size, int index) {
  for (int i = 0; i < size; i++) {
    cout << data[i];
    if (i == index) {
      cout << " <-- largest";
    }
    cout << endl;
  }
}
```





Console

```
32
54
67.5 <-- largest
29
```

int main( )

```
const int SIZE = 7;

int main(){

   double data[SIZE];
   int size = 0;

   loadArray(data, size);
   int index = findMax(data, size);
   printResults(data, size, index);

   return 0;
}
```

Console

```
32
54
67.5 <-- largest
29
```