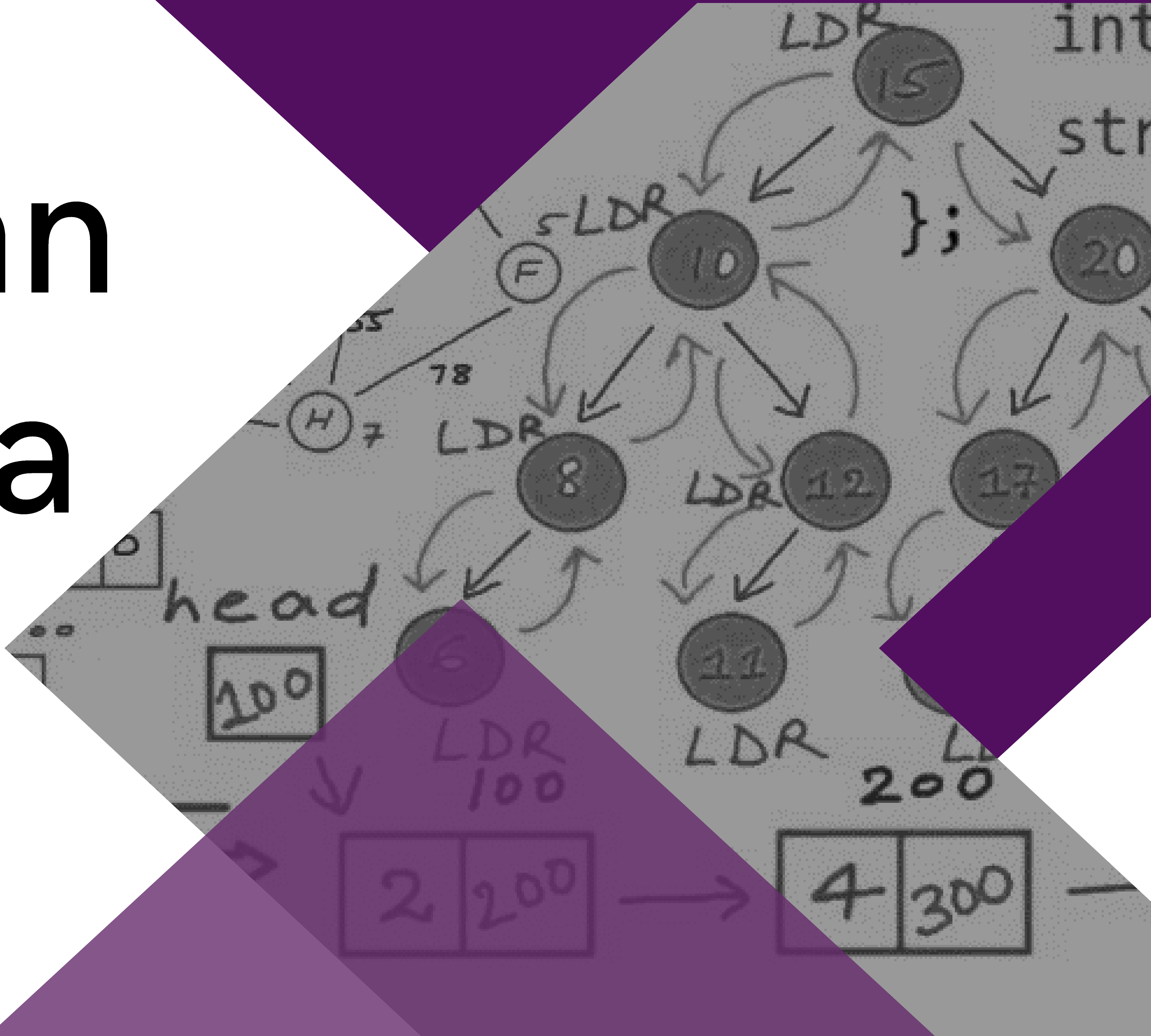
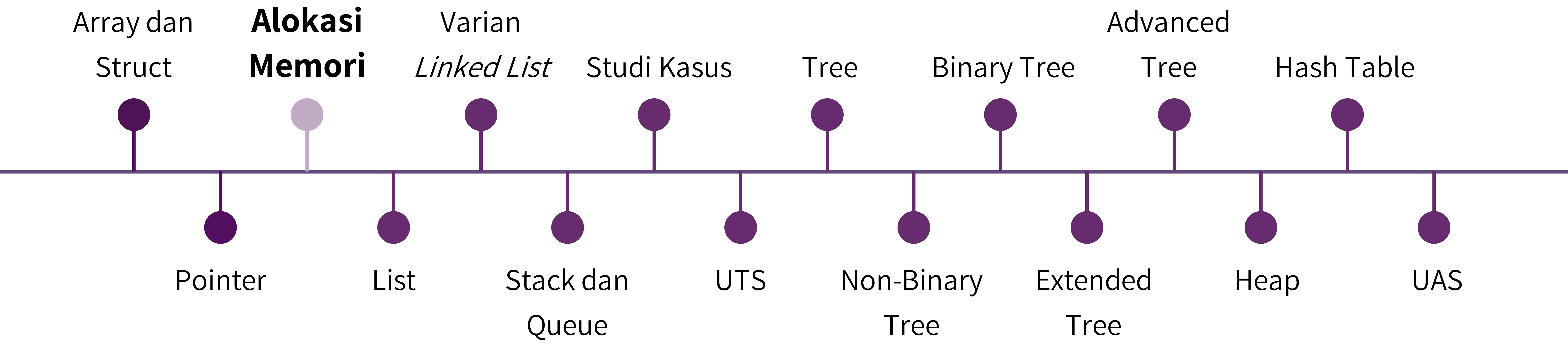
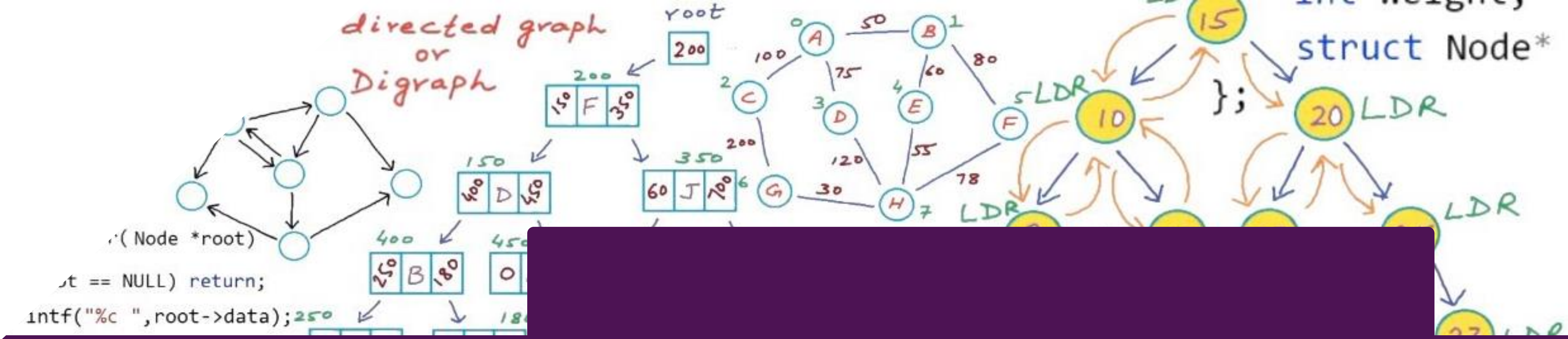
Algoritma dan Struktur Data



Pekan 3


Array dan Struct
Pointer
Alokasi Memori
List
Varian *Linked List*
Stack dan Queue
Studi Kasus
UTS
Tree
Non-Binary Tree
Binary Tree
Extended Tree
Advanced Tree
Heap
Hash Table
UAS

Tujuan



1	Mahasiswa memahami alokasi memori statis dan dinamis
2	Mahasiswa memahami representasi data yang mencakup definisi fungsional, representasi logis dan representasi fisik.

Alokasi Memori

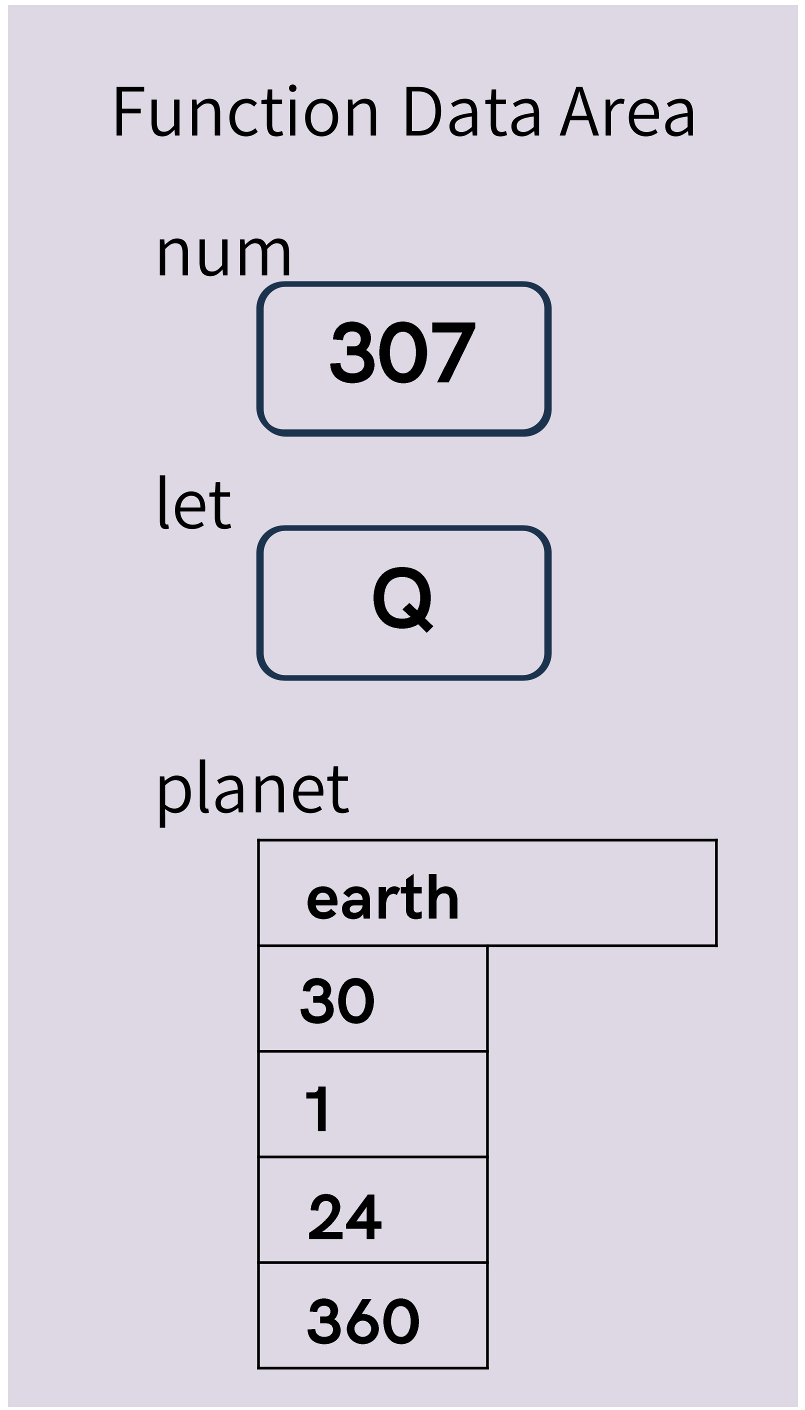
Dynamically Allocated Array

Dynamically Allocated Array:

1. A dynamically allocated array refers to an array that is explicitly allocated from the heap memory (using functions like malloc in C or new in C++) rather than being created as part of the program's stack frame.
2. Dynamically allocated arrays are used when you need to allocate memory for an array at runtime, and you have explicit control over memory allocation and deallocation.
3. They require manual memory management, meaning you must release the allocated memory when it's no longer needed (using free in C or delete in C++).
4. Dynamically allocated arrays can have a fixed size or a size determined at runtime, but they don't automatically resize like dynamic arrays.

Alokasi Memori

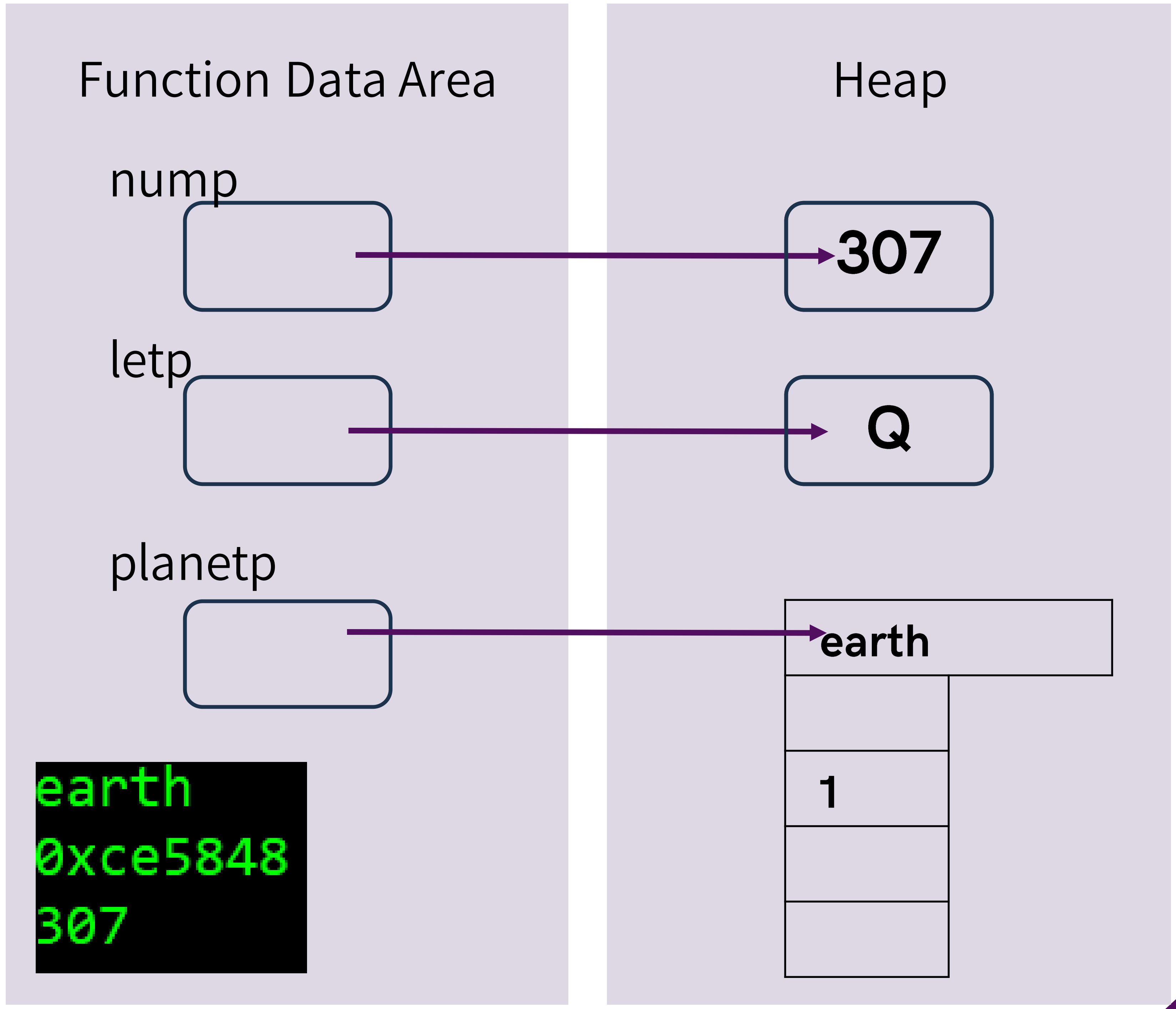
```cpp
#include <iostream>
#include <cstring>
using namespace std;
#define STRSZ 10
typedef struct {
    char name[STRSZ];
    float diameter;
    int moons;
    float orbit_time, rotation_time;
} planet_t;

int main(){
    int num;
    char let;
    planet_t planet;

    num = 307;
    let = 'Q';
    strcpy(planet.name,"earth");
    planet.diameter = 30;
    planet.moons = 1;
    planet.orbit_time = 24;
    planet.rotation_time = 360;

    cout << planet.name;

    return(0);
}
```



A specific region of memory or data storage dedicated to storing data used by the system's functions or tasks

Alokasi Memori Dinamis

```
#include <iostream>
#include <cstring>
using namespace std;
#define STRSZ 10

typedef struct {
    char name[STRSZ];
    float diameter;
    int moons;
    float orbit_time, rotation_time;
}planet_t;

int main(){
    int *nump;
    char *letp;
    planet_t *planetp;

    nump = new int;
    letp = new char;
    planetp = new planet_t;

    *nump = 307;
    *letp = 'Q';
    strcpy(planetp->name,"earth");
    planetp->moons = 1;

    cout << planetp->name << endl;
    cout << nump << endl;
    cout << *nump << endl;

    return (0);
}
```



```
earth
0xce5848
307
```

region of memory in which function `new` dynamically allocates blocks of storage

Mengakses Komponen pada Struktur dengan Alokasi Dinamis

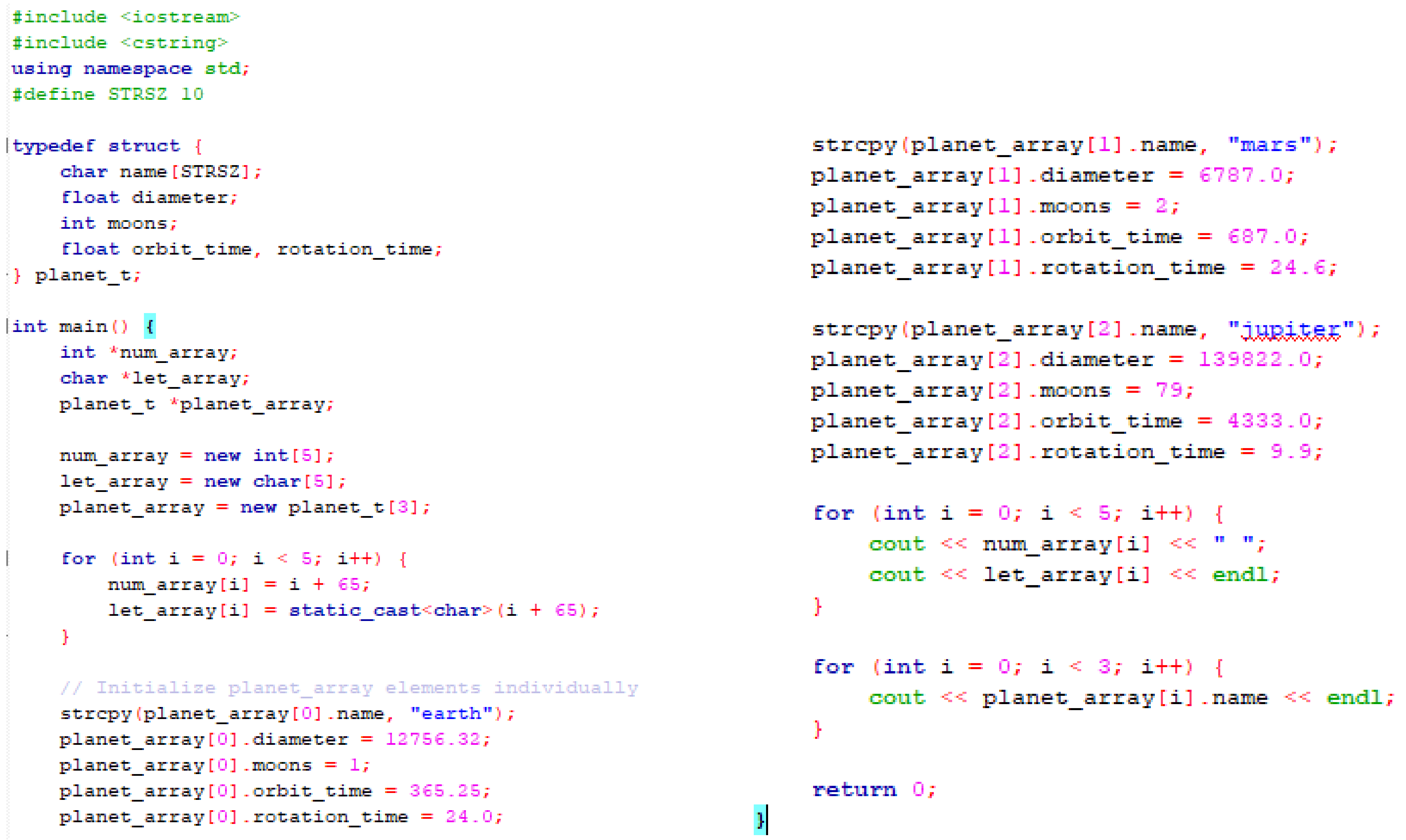
```
(*nama_var_pointer).nama_komponen
```

```
nama_var_pointer->nama_komponen
```

```
cout << planetp->name << endl;
cout << (*planetp).name << endl;
```

```
earth
earth
```

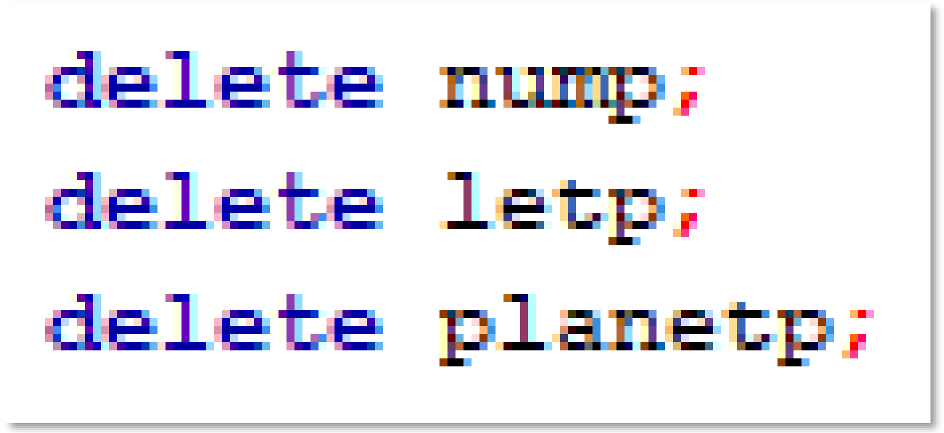
Alokasi Array Dinamis (Dynamic Allocated Array)



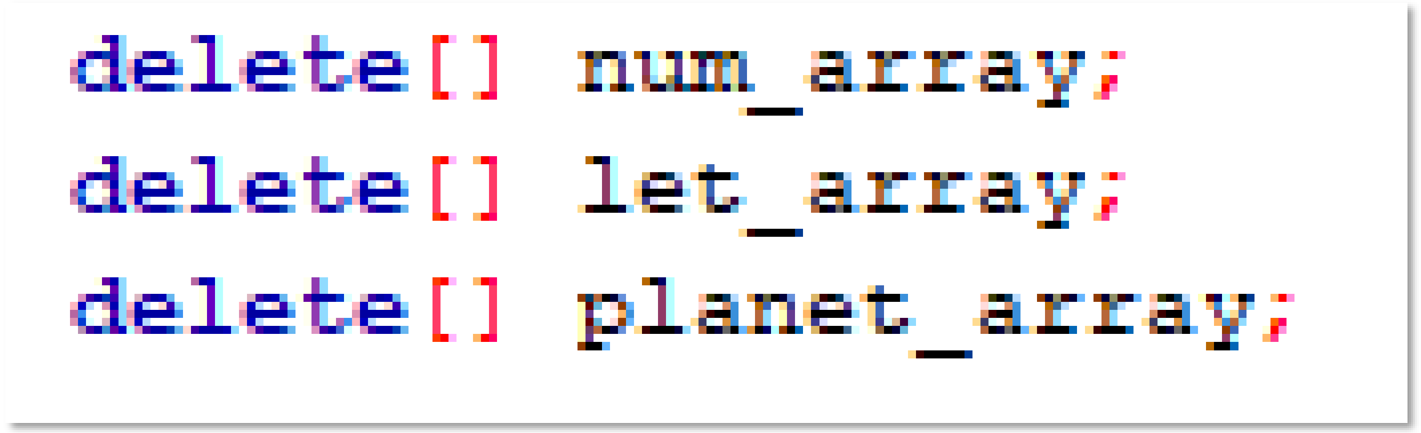
```cpp
#include <iostream>
#include <cstring>
using namespace std;
#define STRSZ 10

typedef struct {
    char name[STRSZ];
    float diameter;
    int moons;
    float orbit_time, rotation_time;
} planet_t;

int main() {
    int *num_array;
    char *let_array;
    planet_t *planet_array;

    num_array = new int[5];
    let_array = new char[5];
    planet_array = new planet_t[3];

    for (int i = 0; i < 5; i++) {
        num_array[i] = i + 65;
        let_array[i] = static_cast<char>(i + 65);
    }

    // Initialize planet_array elements individually
    strcpy(planet_array[0].name, "earth");
    planet_array[0].diameter = 12756.32;
    planet_array[0].moons = 1;
    planet_array[0].orbit_time = 365.25;
    planet_array[0].rotation_time = 24.0;

    strcpy(planet_array[1].name, "mars");
    planet_array[1].diameter = 6787.0;
    planet_array[1].moons = 2;
    planet_array[1].orbit_time = 687.0;
    planet_array[1].rotation_time = 24.6;

    strcpy(planet_array[2].name, "jupiter");
    planet_array[2].diameter = 139822.0;
    planet_array[2].moons = 79;
    planet_array[2].orbit_time = 4333.0;
    planet_array[2].rotation_time = 9.9;

    for (int i = 0; i < 5; i++) {
        cout << num_array[i] << " ";
        cout << let_array[i] << endl;
    }

    for (int i = 0; i < 3; i++) {
        cout << planet_array[i].name << endl;
    }

    return 0;
}
```

Menghapus Memori Dinamis (Deallocate Dynamic Memory)

Menghapus memori dinamis adalah proses mengembalikan memori yang telah dialokasikan secara dinamis kembali ke heap memory agar bisa digunakan kembali oleh program atau sistem operasi.

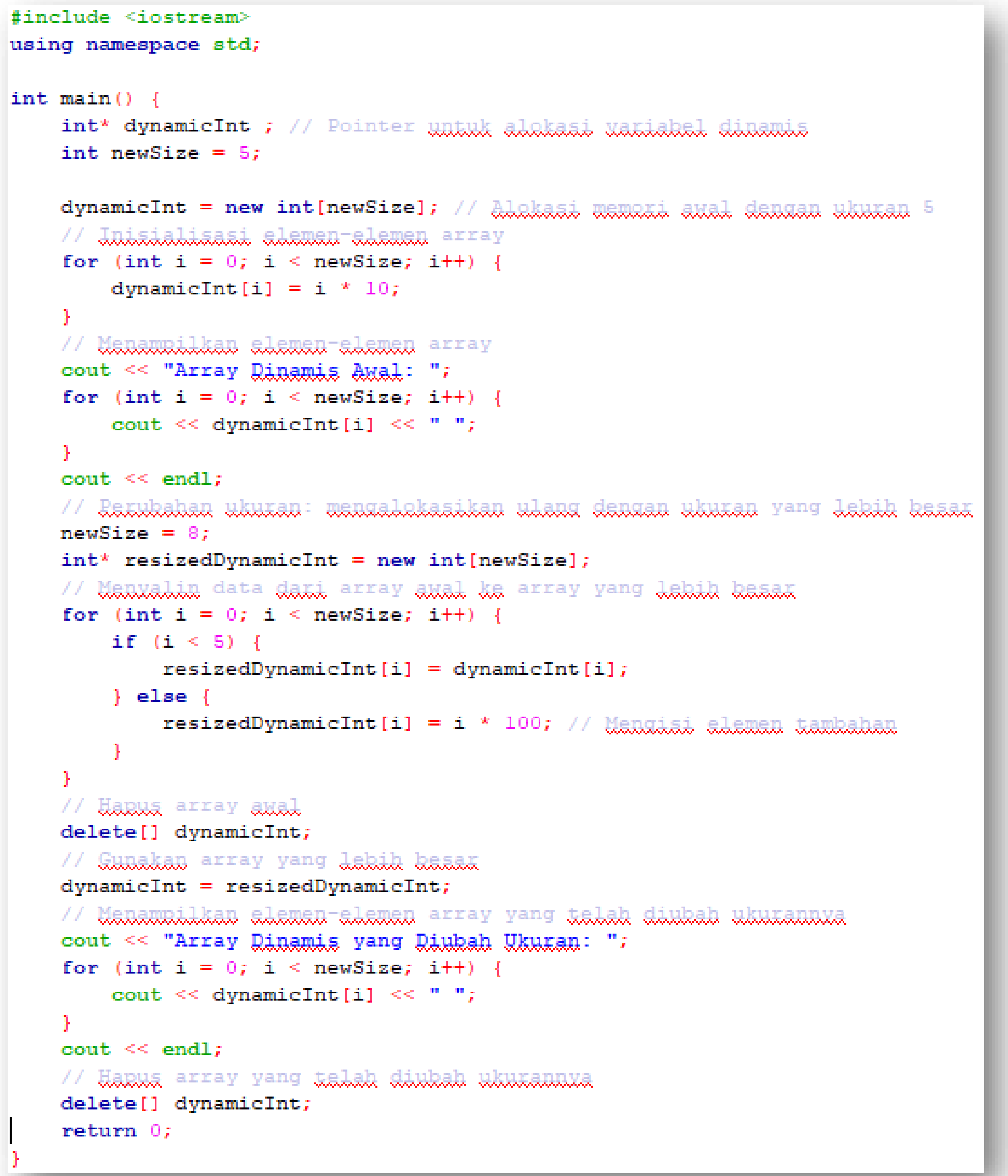


```
delete nump;
delete letp;
delete planetp;
```



```
delete[] num_array;
delete[] let_array;
delete[] planet_array;
```

Example



```cpp
#include <iostream>
using namespace std;

int main() {
    int* dynamicInt ; // Pointer untuk alokasi variabel dinamis
    int newSize = 5;

    dynamicInt = new int[newSize]; // Alokasi memori awal dengan ukuran 5
    // Inisialisasi elemen-elemen array
    for (int i = 0; i < newSize; i++) {
        dynamicInt[i] = i * 10;
    }
    // Menampilkan elemen-elemen array
    cout << "Array Dinamis Awal: ";
    for (int i = 0; i < newSize; i++) {
        cout << dynamicInt[i] << " ";
    }
    cout << endl;
    // Perubahan ukuran: mengalokasikan ulang dengan ukuran yang lebih besar
    newSize = 8;
    int* resizedDynamicInt = new int[newSize];
    // Menyalin data dari array awal ke array yang lebih besar
    for (int i = 0; i < newSize; i++) {
        if (i < 5) {
            resizedDynamicInt[i] = dynamicInt[i];
        } else {
            resizedDynamicInt[i] = i * 100; // Mengisi elemen tambahan
        }
    }
    // Hapus array awal
    delete[] dynamicInt;
    // Gunakan array yang lebih besar
    dynamicInt = resizedDynamicInt;
    // Menampilkan elemen-elemen array yang telah diubah ukurannya
    cout << "Array Dinamis yang Diubah Ukuran: ";
    for (int i = 0; i < newSize; i++) {
        cout << dynamicInt[i] << " ";
    }
    cout << endl;
    // Hapus array yang telah diubah ukurannya
    delete[] dynamicInt;
    return 0;
}
```

Array Dinamis

Static Array

Static array is **static!**

```
int my_array[100];
```

Semi-solution: dynamically-allocated arrays:

```
int *my_array = new int[size];
```

Dynamic Array

Problem: might not know max size when allocating an array

Solution: dynamic arrays (also known as *resizable arrays)*

Idea: store a pointer to a dynamically allocated array and replace it with a newly-allocated array as needed.

Dynamic Array Operation

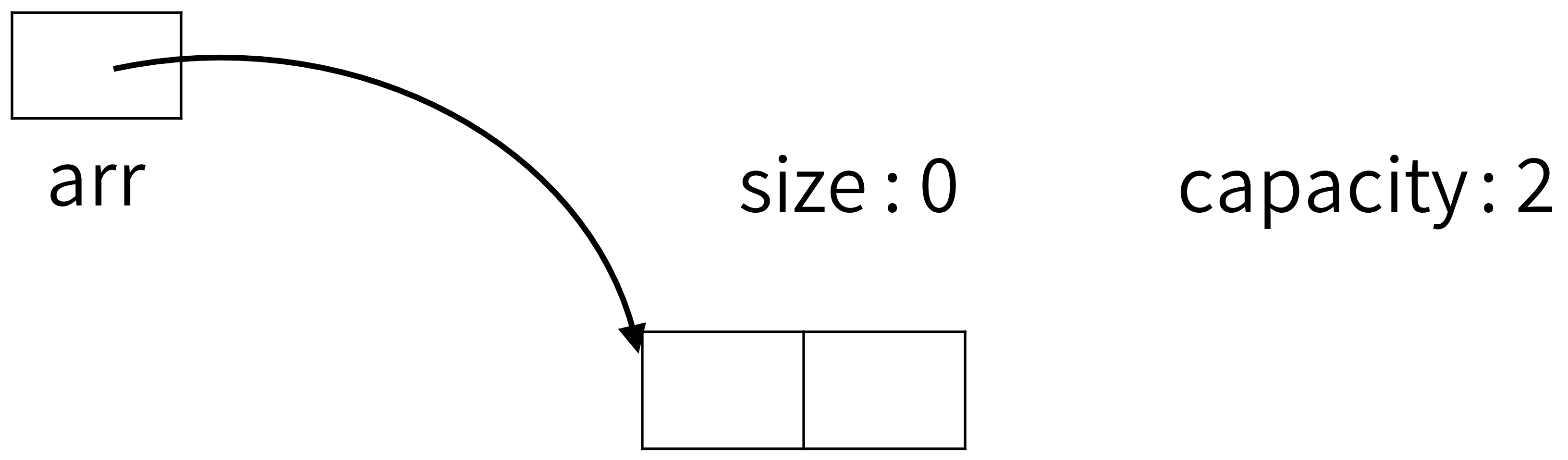
`get(i)`	Returns element at location i*
`set(i,val)`	Sets element i to val*
`PushBack(val)`	Add val to the end
`Remove(i)`	Removes element at location i
`Size()`	The number of elements

Implementation

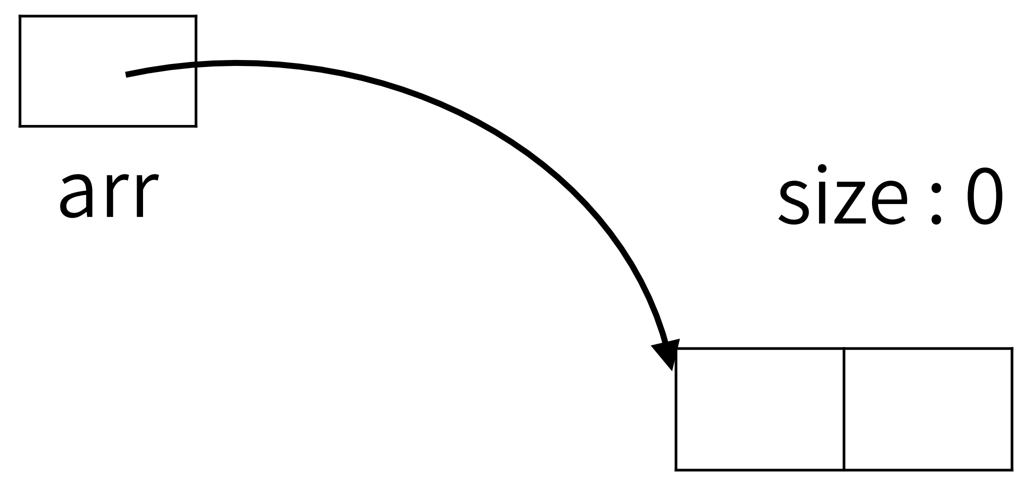
Store:

- `arr`: dynamically-allocated array
- `capacity`: size of the dynamically-allocated array
- `size`: number of elements currently in the array

Example: Dynamic Array Resizing



Example: Dynamic Array Resizing

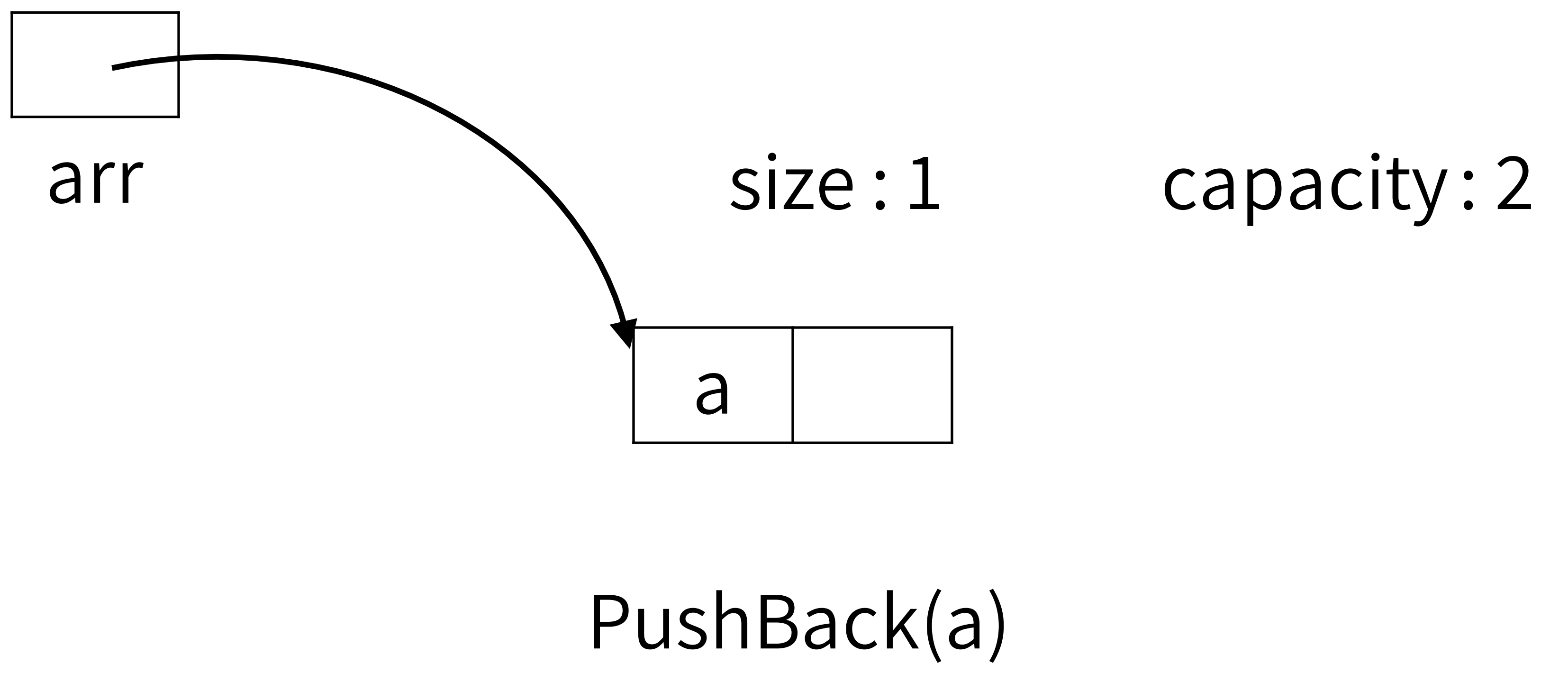


size : 0

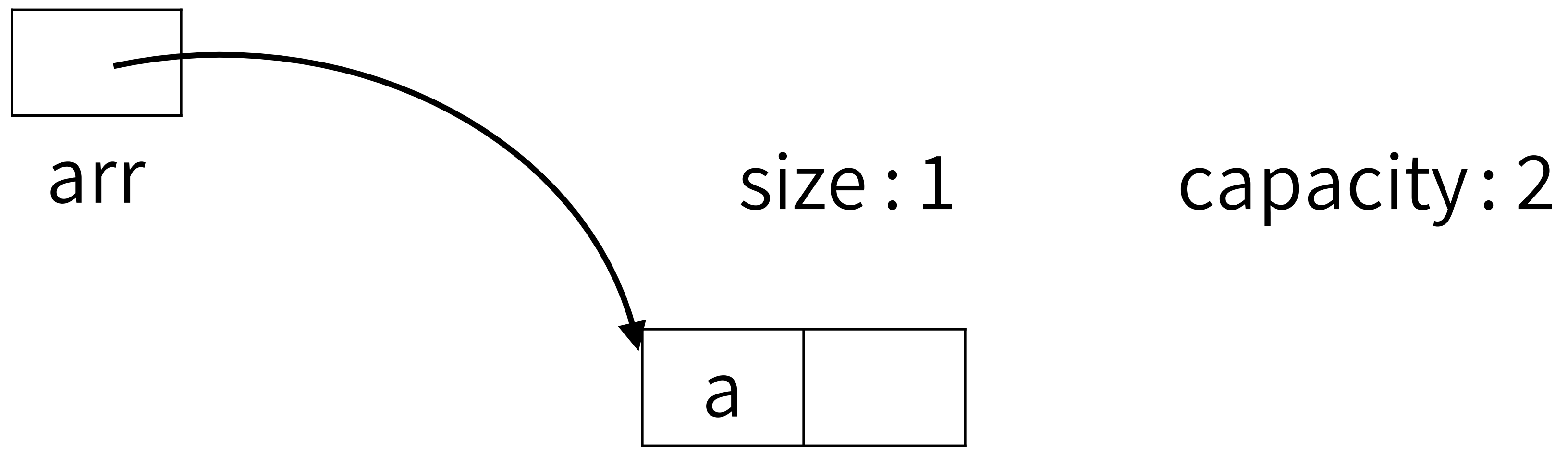
capacity : 2

PushBack(a)

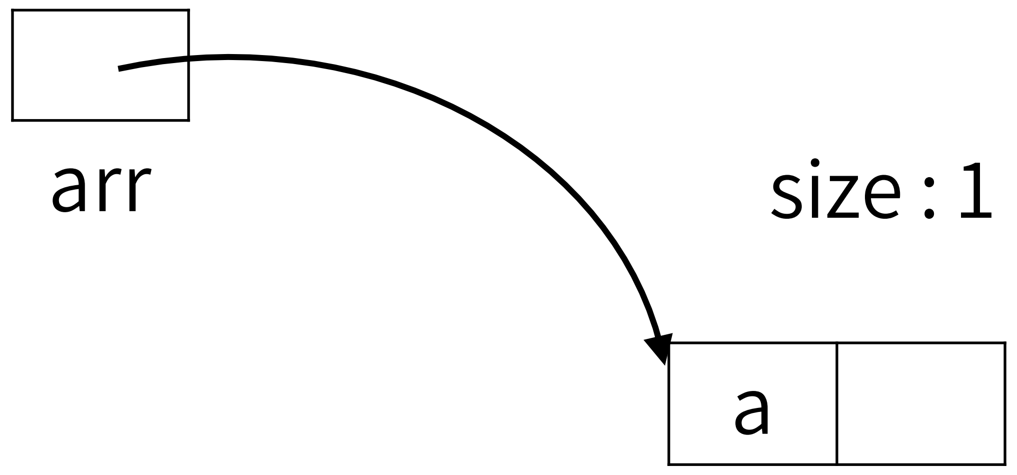
Example: Dynamic Array Resizing



Example: Dynamic Array Resizing



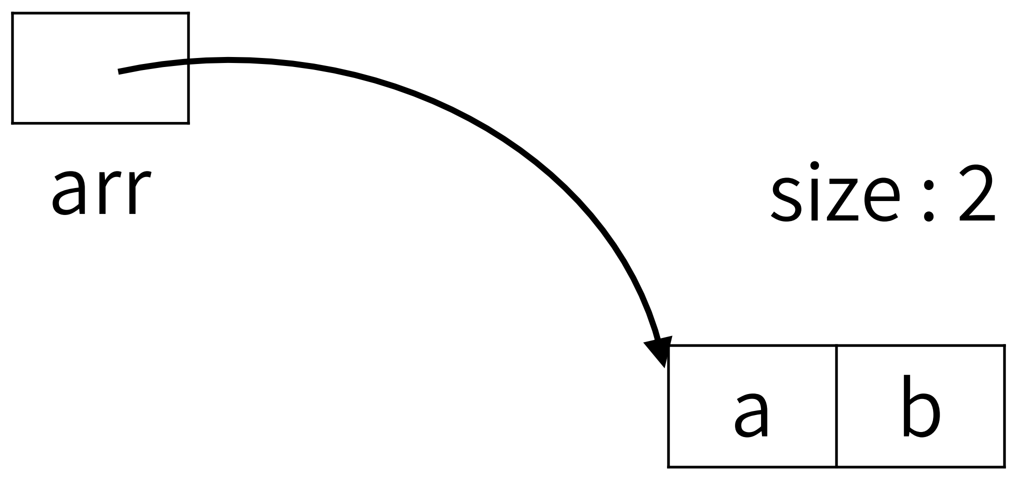
Example: Dynamic Array Resizing



size : 1 capacity : 2

PushBack(b)

Example: Dynamic Array Resizing

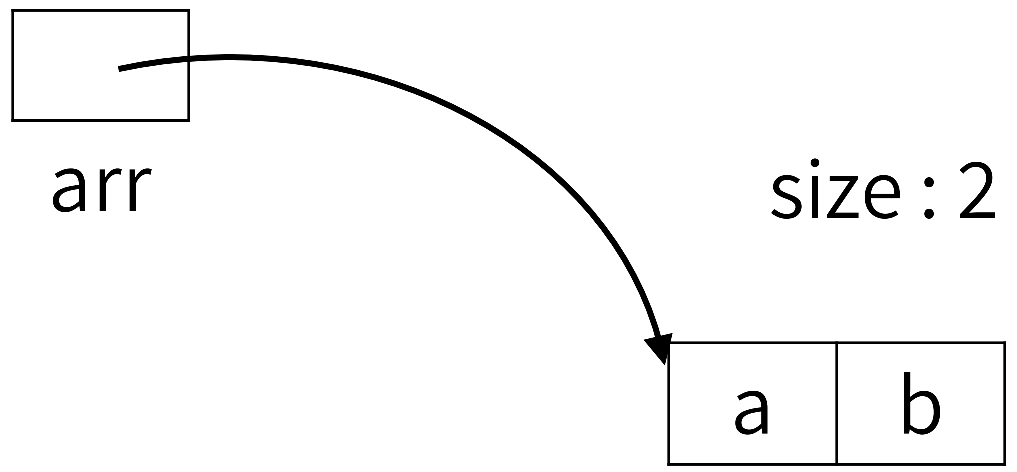


size : 2

capacity : 2

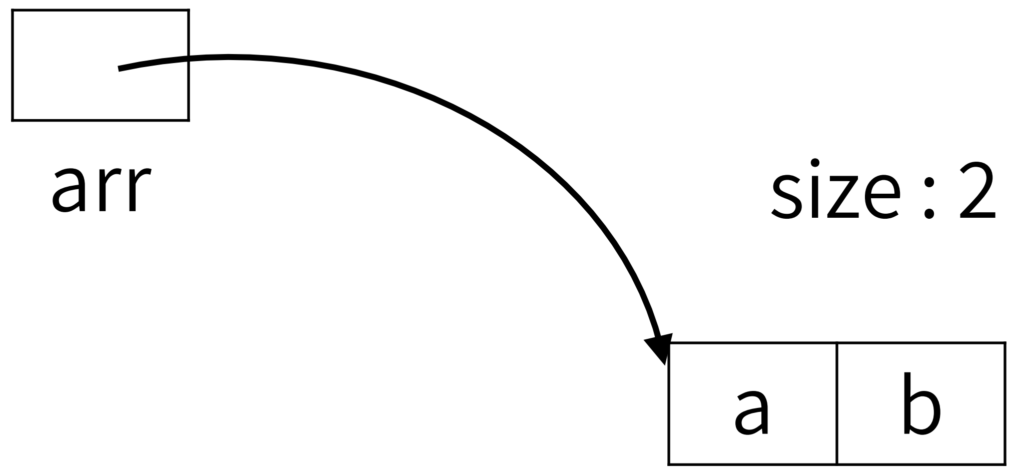
PushBack(b)

Example: Dynamic Array Resizing



size : 2 capacity : 2

Example: Dynamic Array Resizing

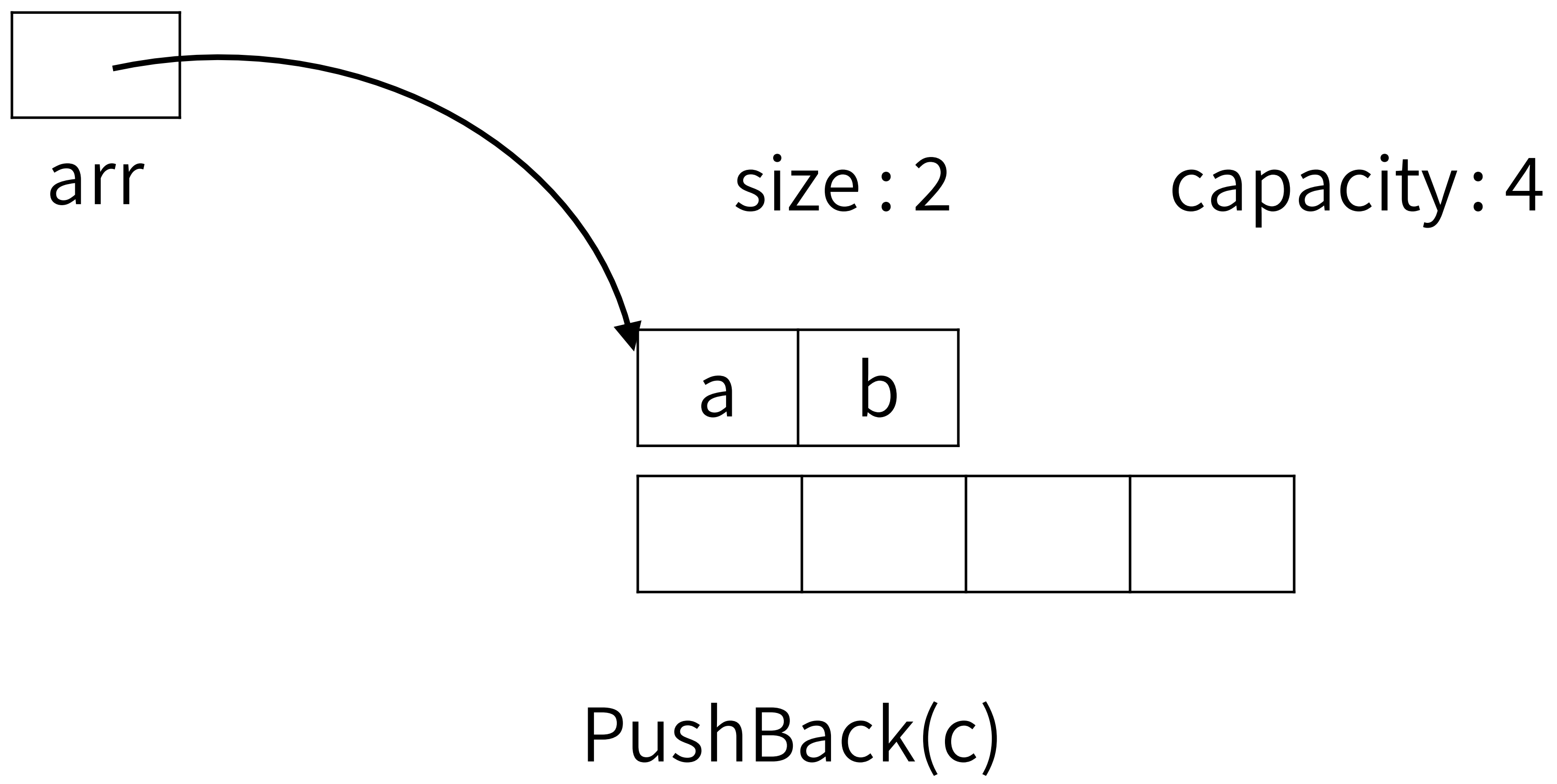


size : 2

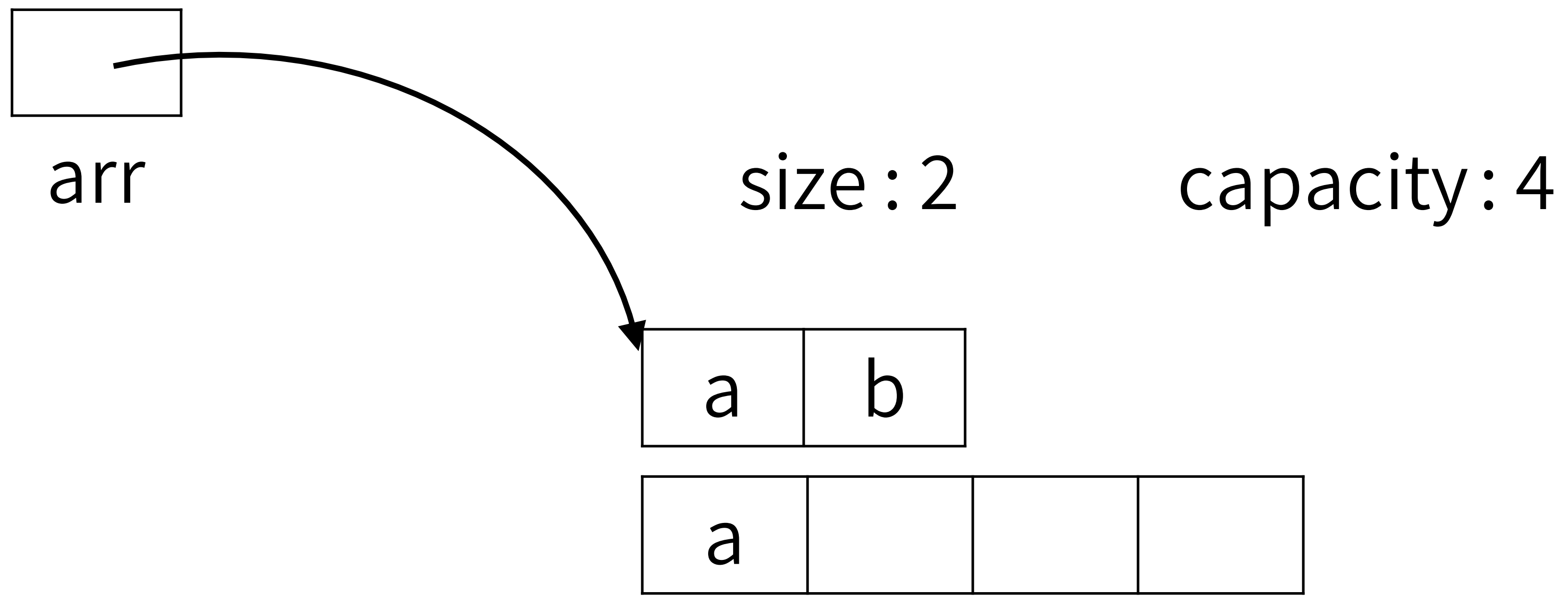
capacity : 2

PushBack(c)

Example: Dynamic Array Resizing

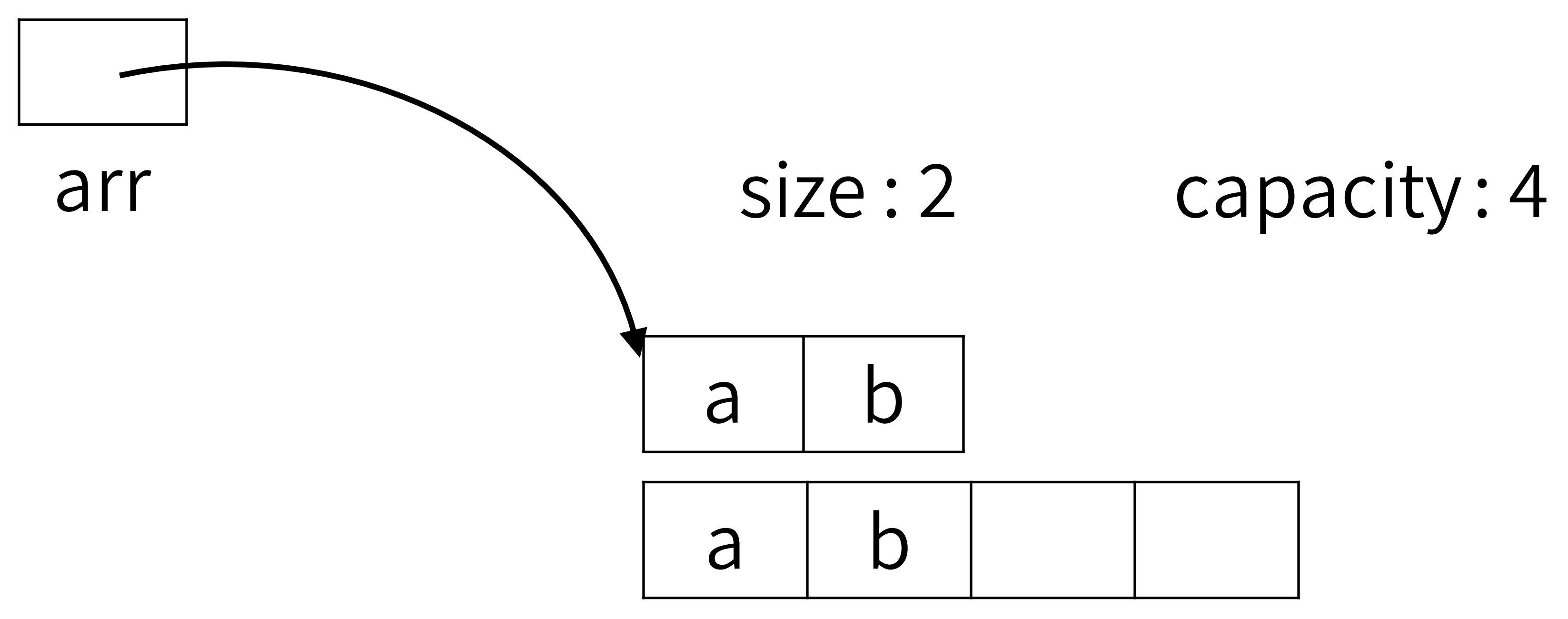


Example: Dynamic Array Resizing



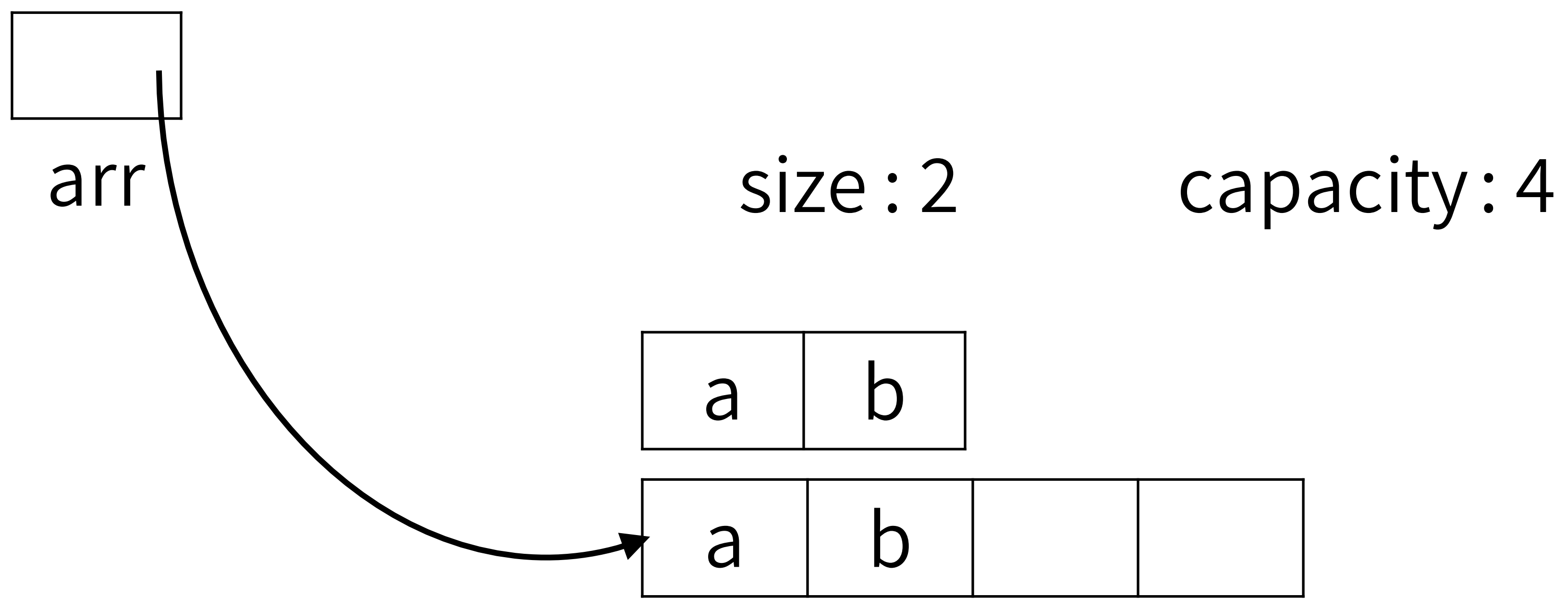
PushBack(c)

Example: Dynamic Array Resizing



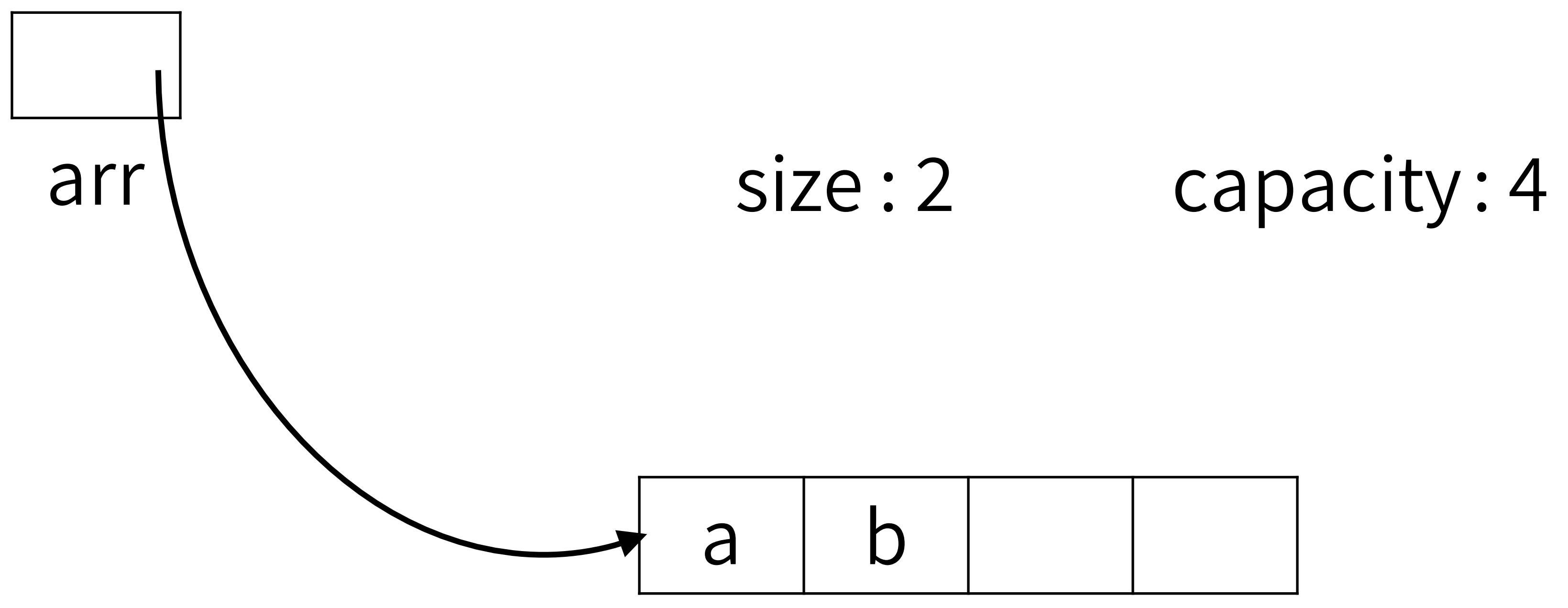
PushBack(c)

Example: Dynamic Array Resizing



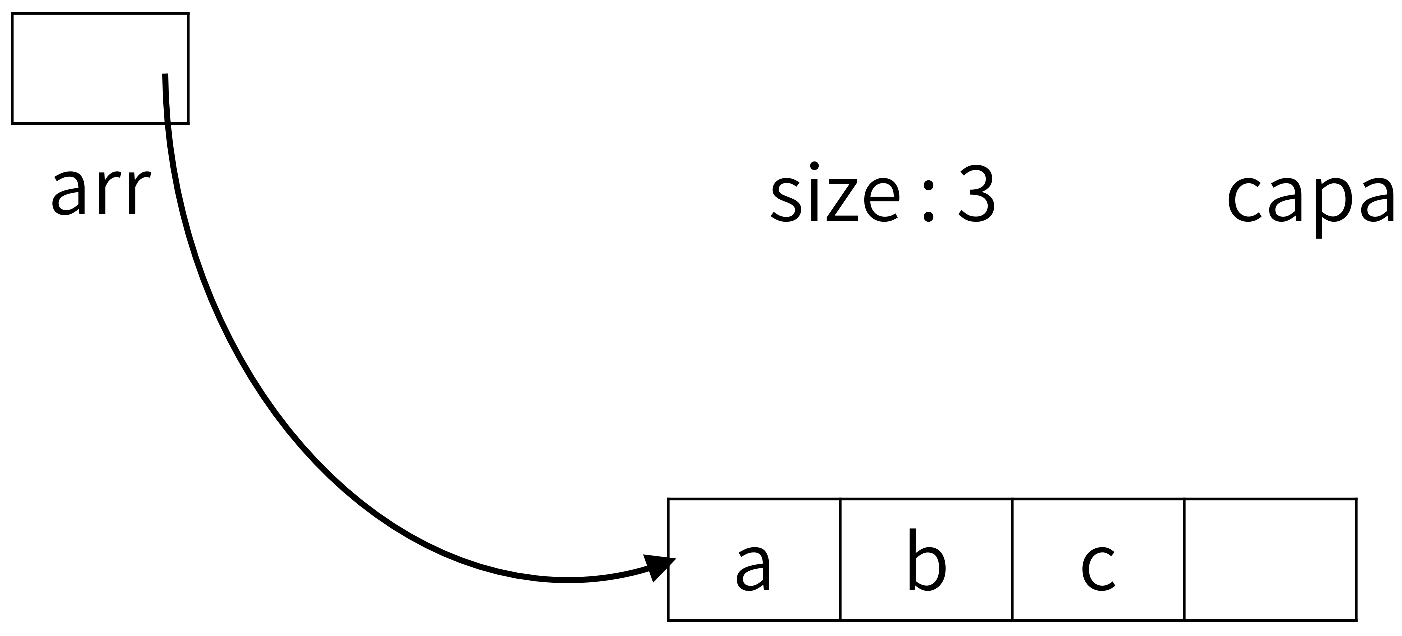
PushBack(c)

Example: Dynamic Array Resizing



PushBack(c)

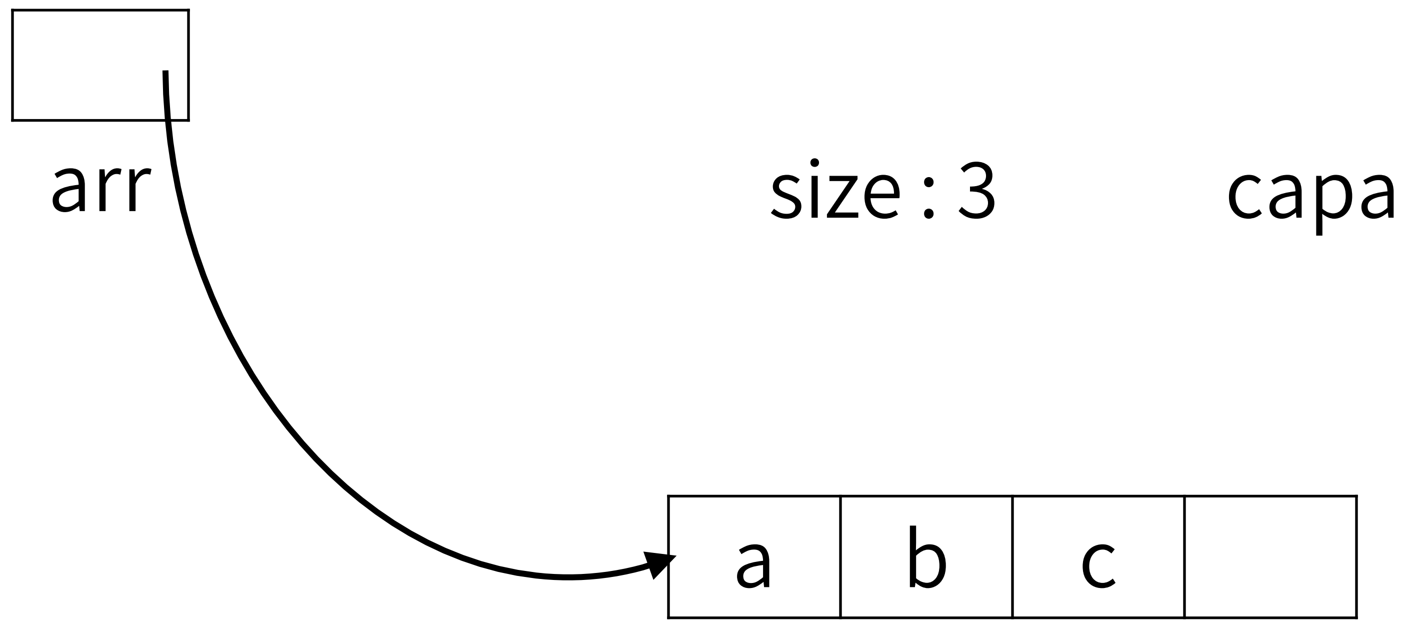
Example: Dynamic Array Resizing



size : 3 capacity : 4

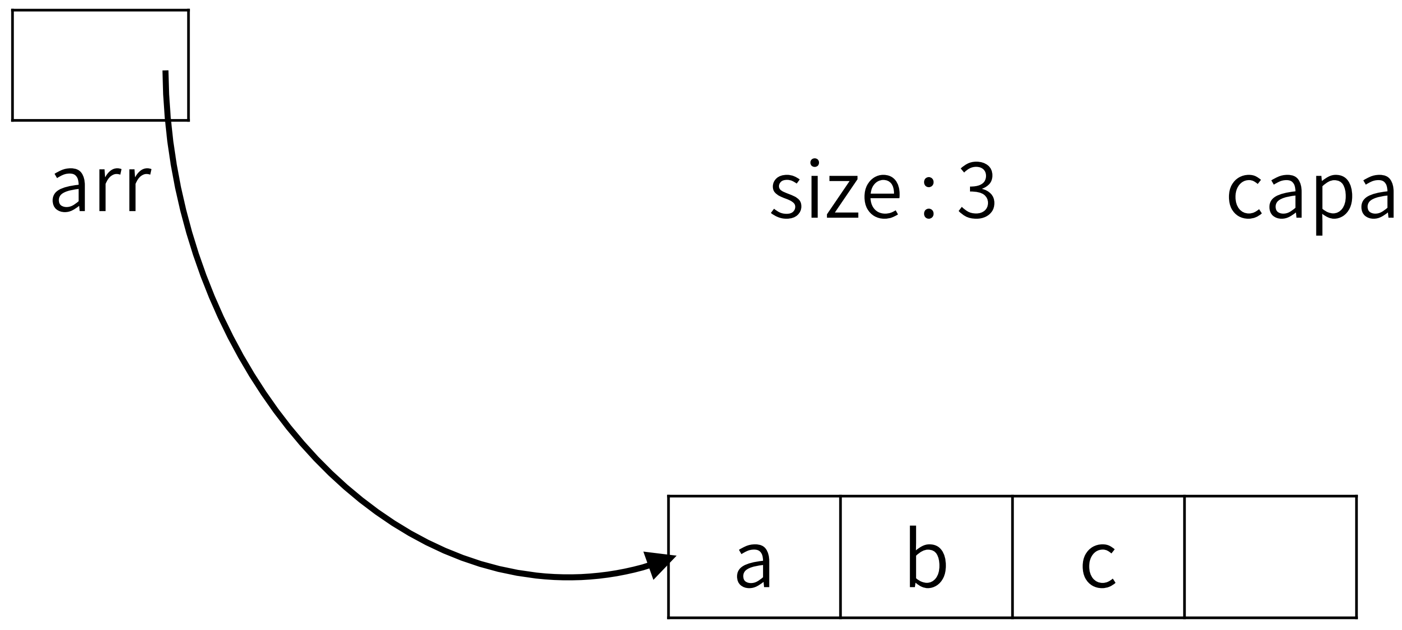
PushBack(c)

Example: Dynamic Array Resizing



size : 3 capacity : 4

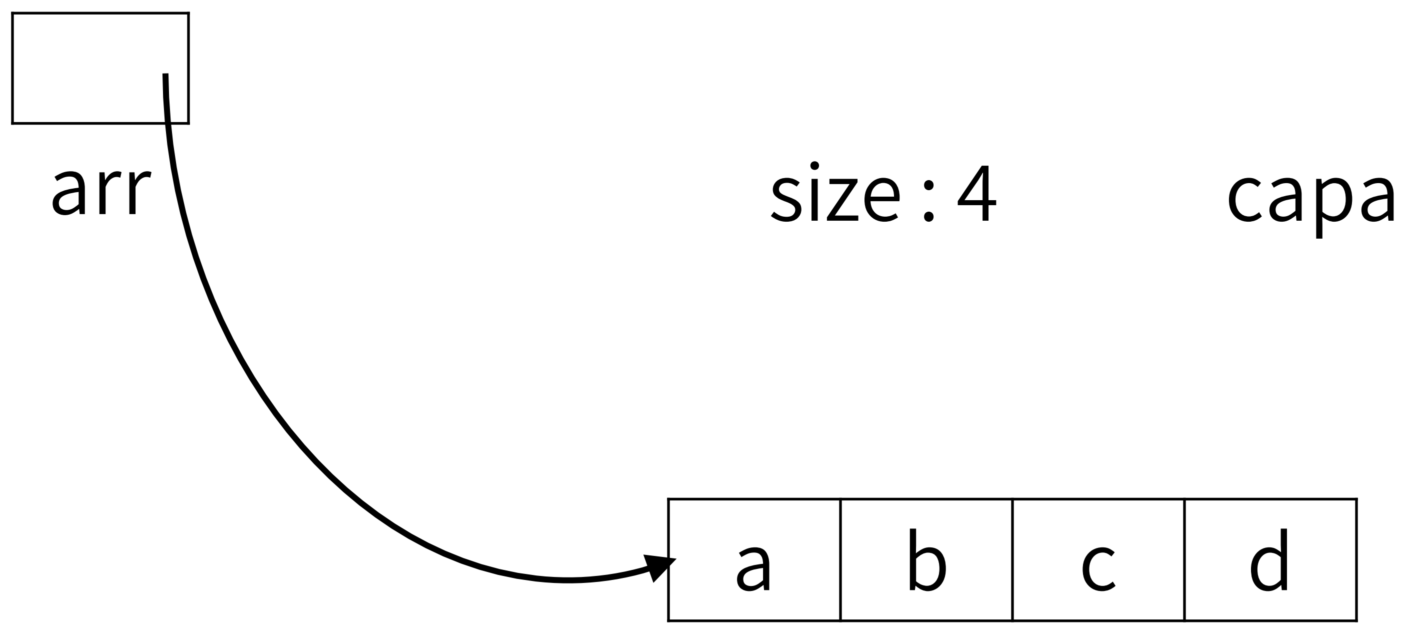
Example: Dynamic Array Resizing



size : 3 capacity : 4

PushBack(d)

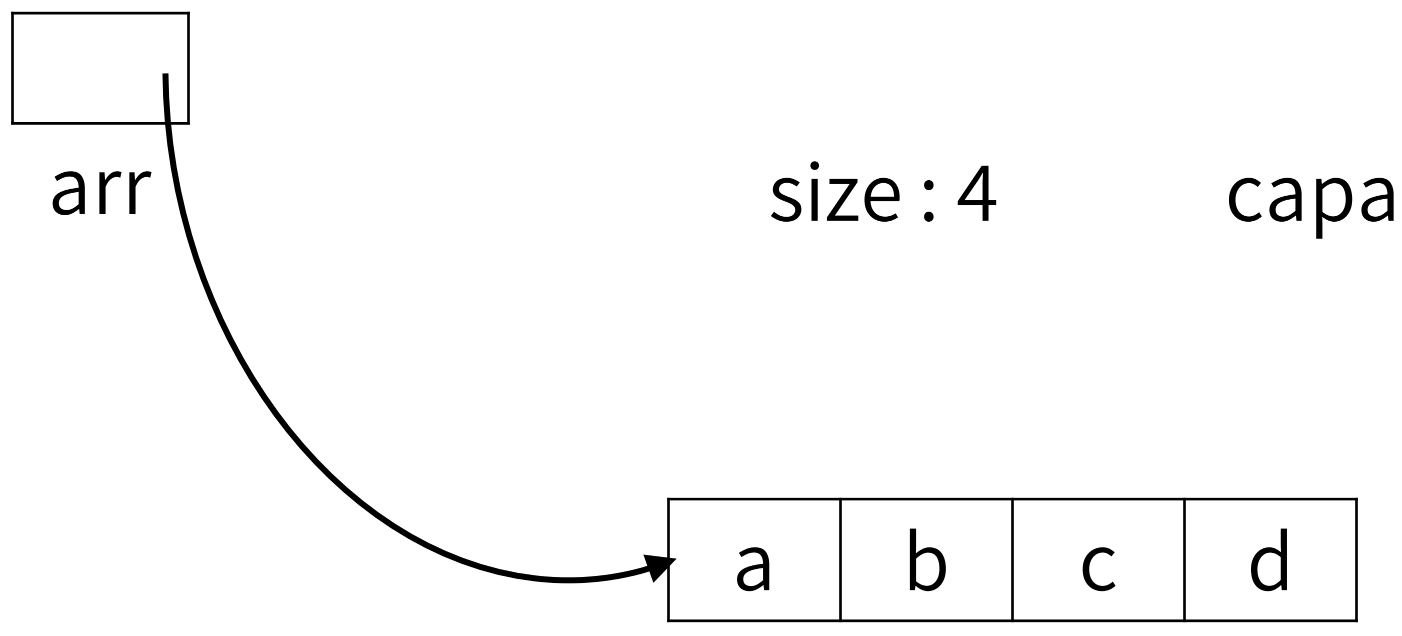
Example: Dynamic Array Resizing



size : 4 capacity : 4

PushBack(d)

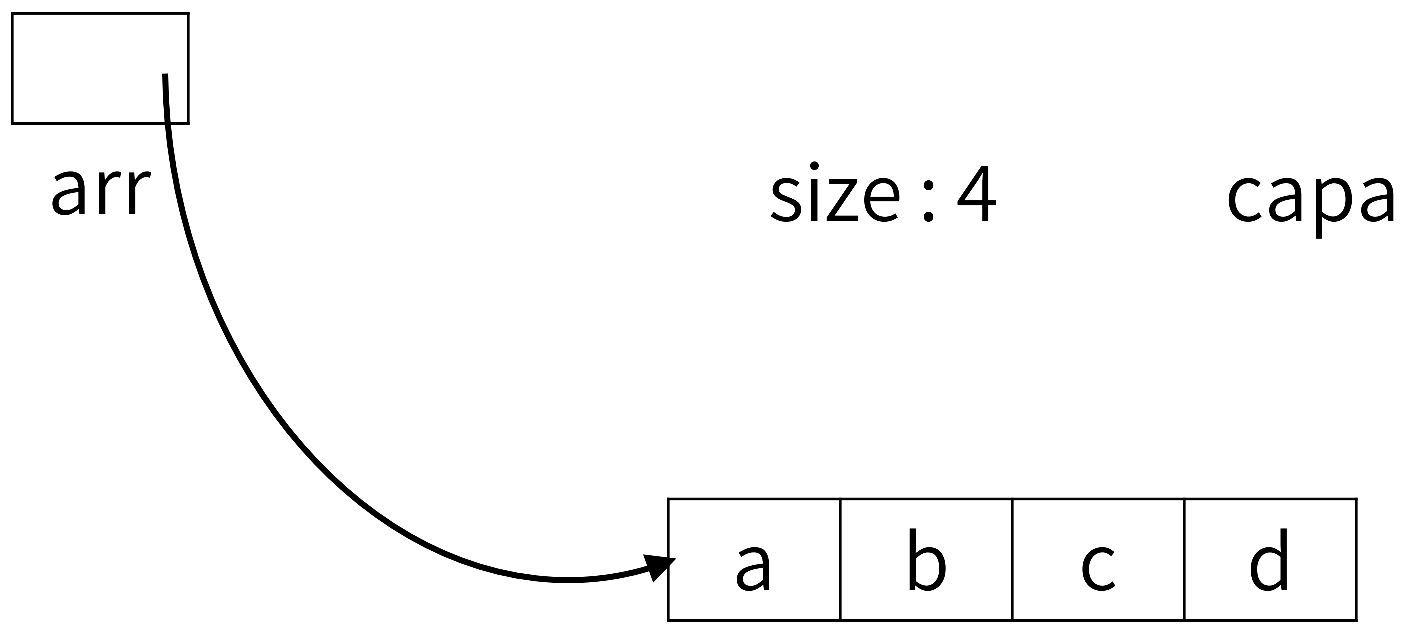
Example: Dynamic Array Resizing



size : 4 capacity : 4

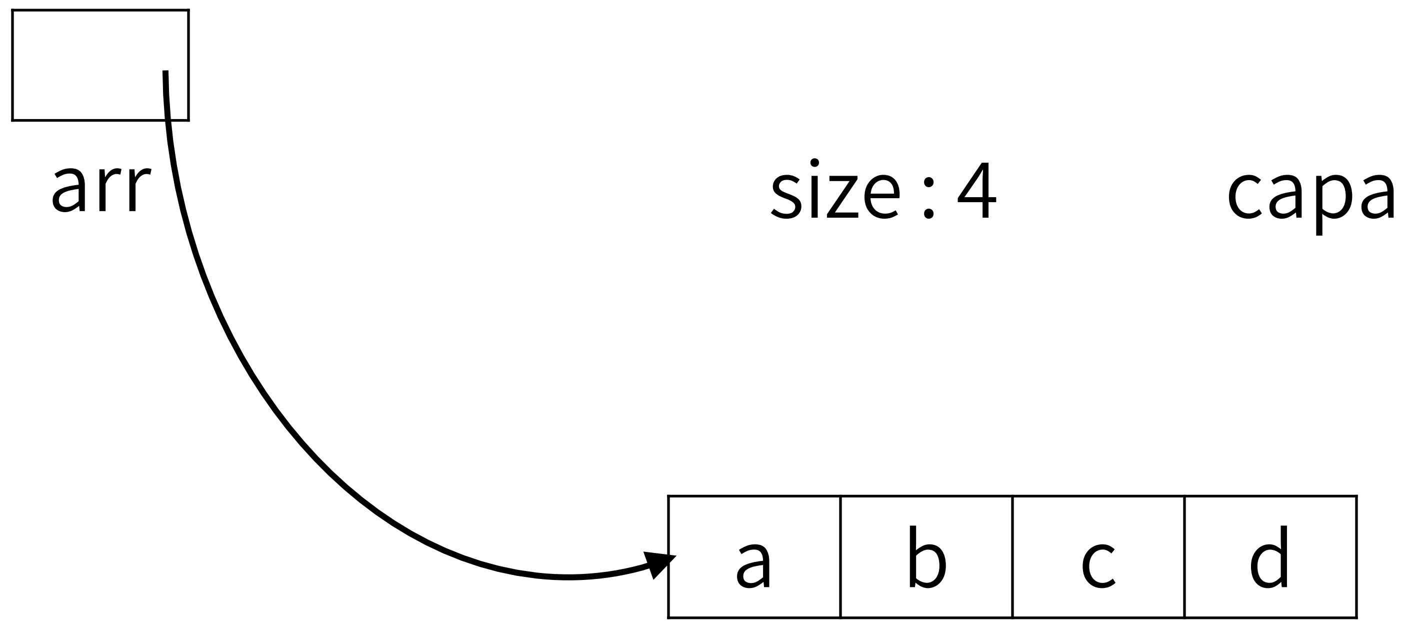
PushBack(d)

Example: Dynamic Array Resizing



size : 4 capacity : 4

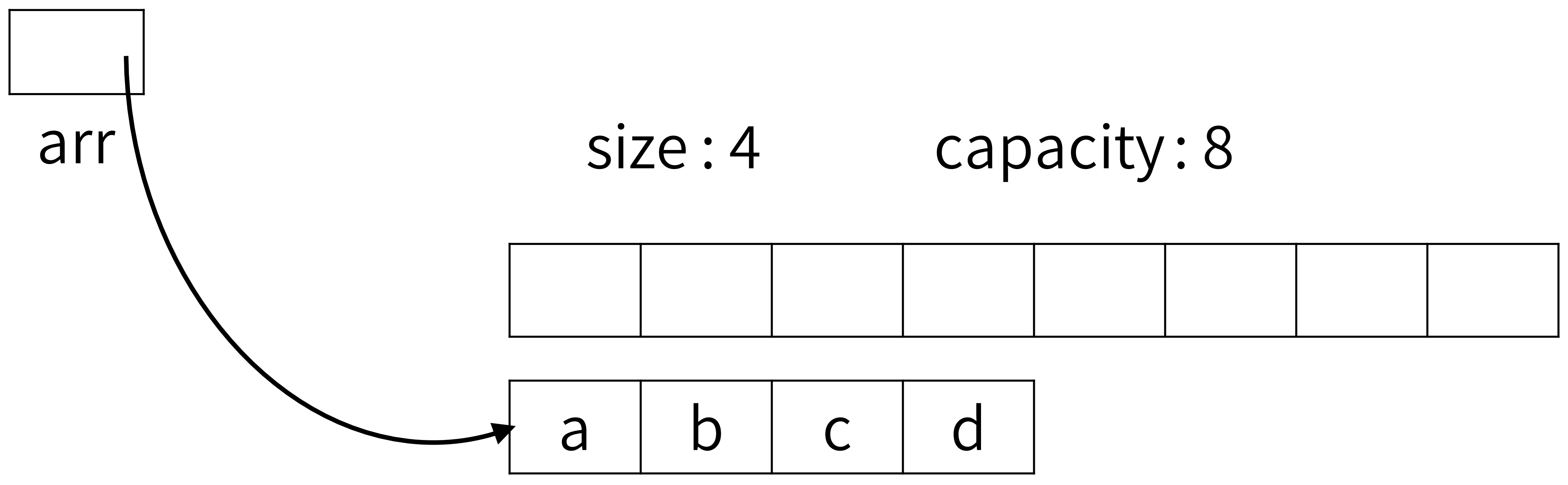
Example: Dynamic Array Resizing



size : 4 capacity : 4

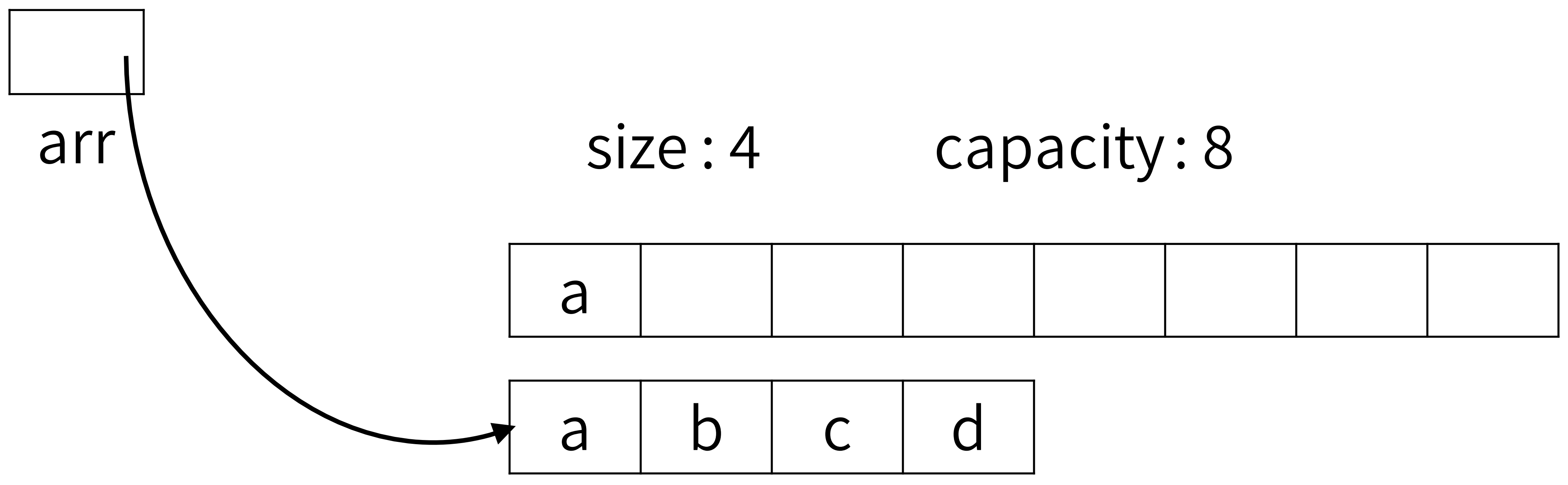
PushBack(e)

Example: Dynamic Array Resizing



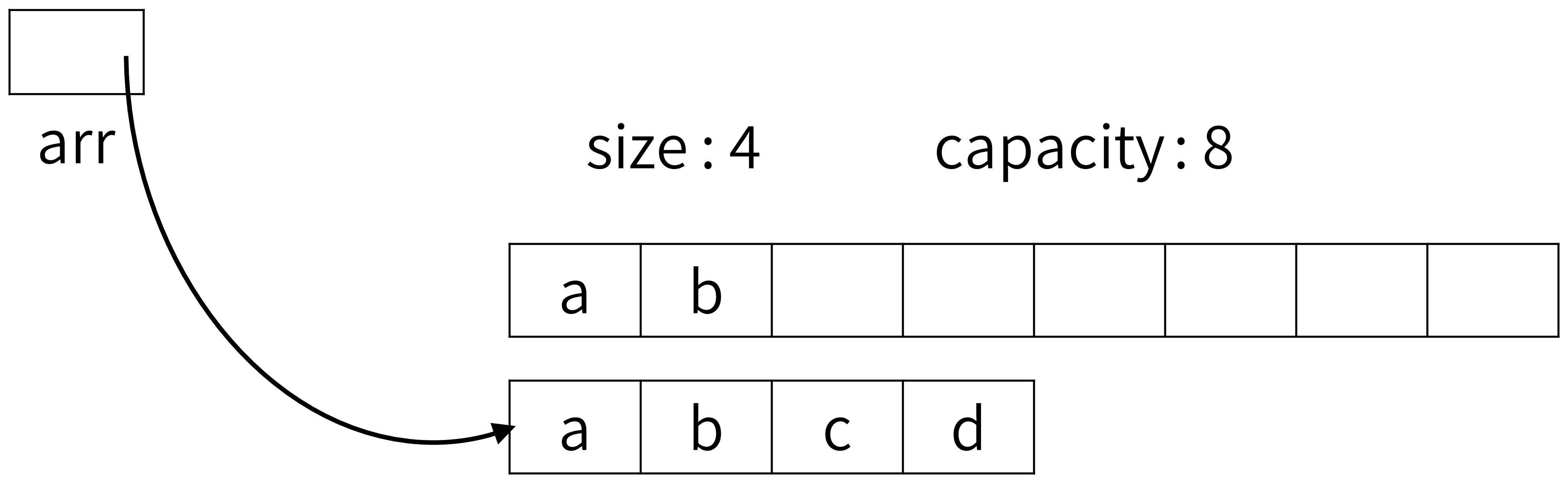
PushBack(e)

Example: Dynamic Array Resizing



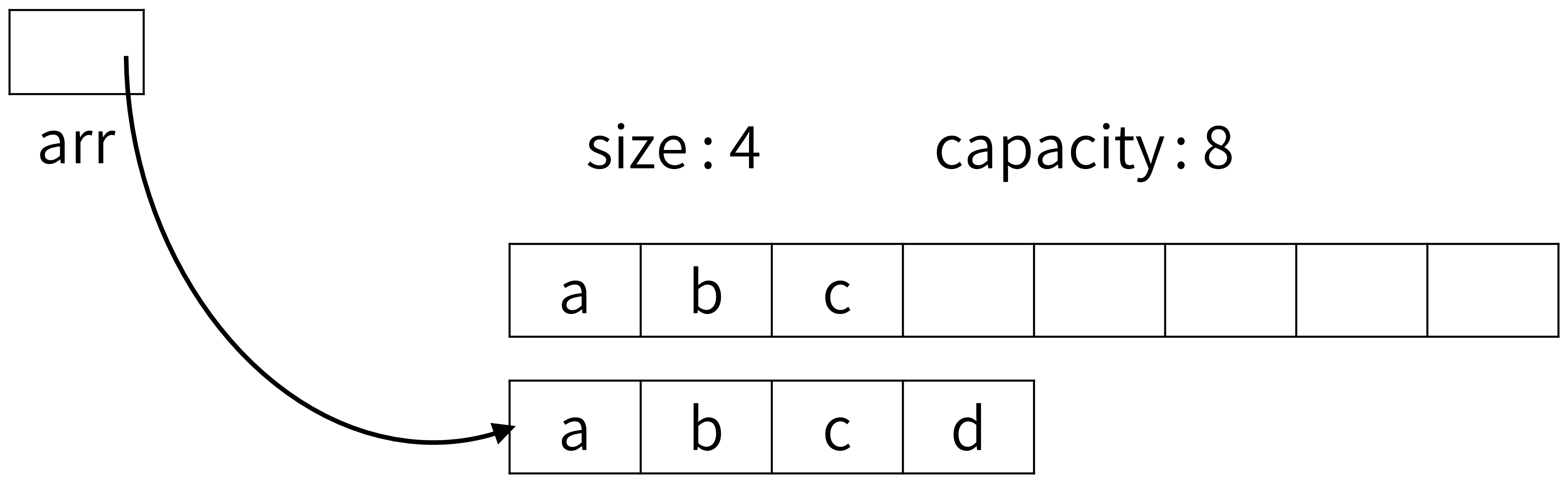
PushBack(e)

Example: Dynamic Array Resizing



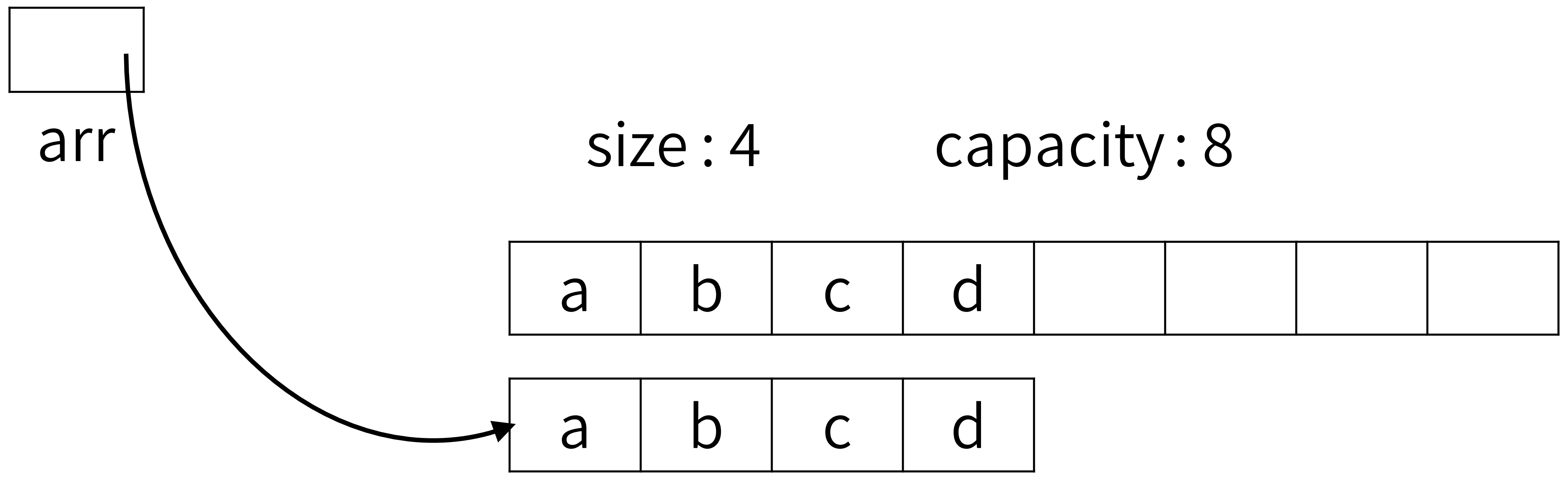
PushBack(e)

Example: Dynamic Array Resizing



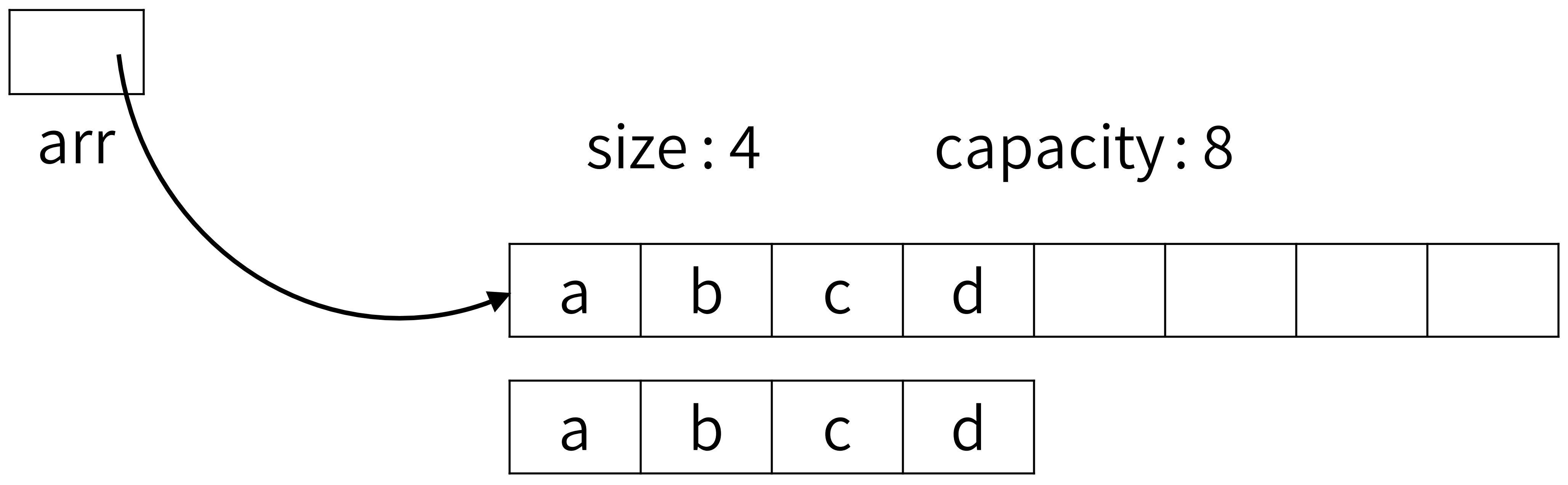
PushBack(e)

Example: Dynamic Array Resizing



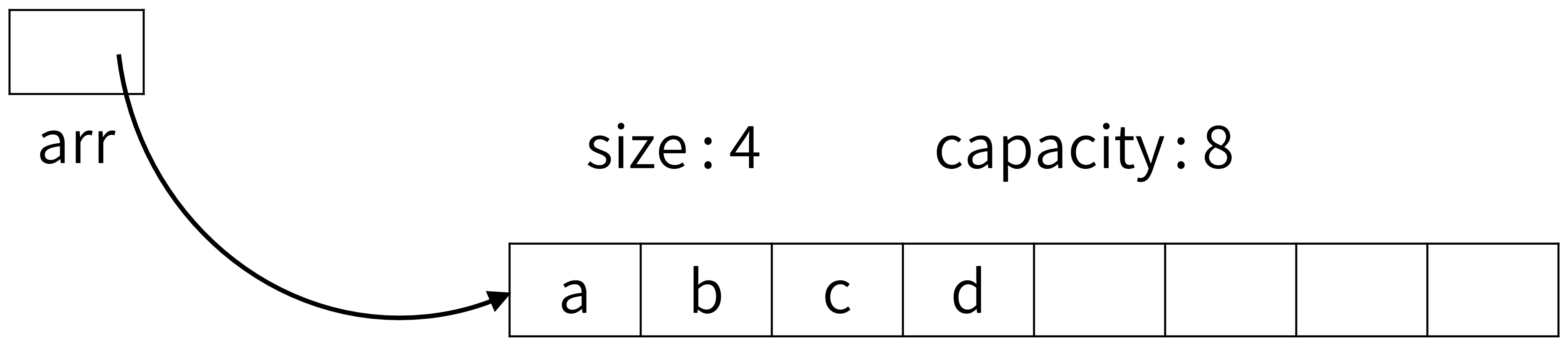
PushBack(e)

Example: Dynamic Array Resizing



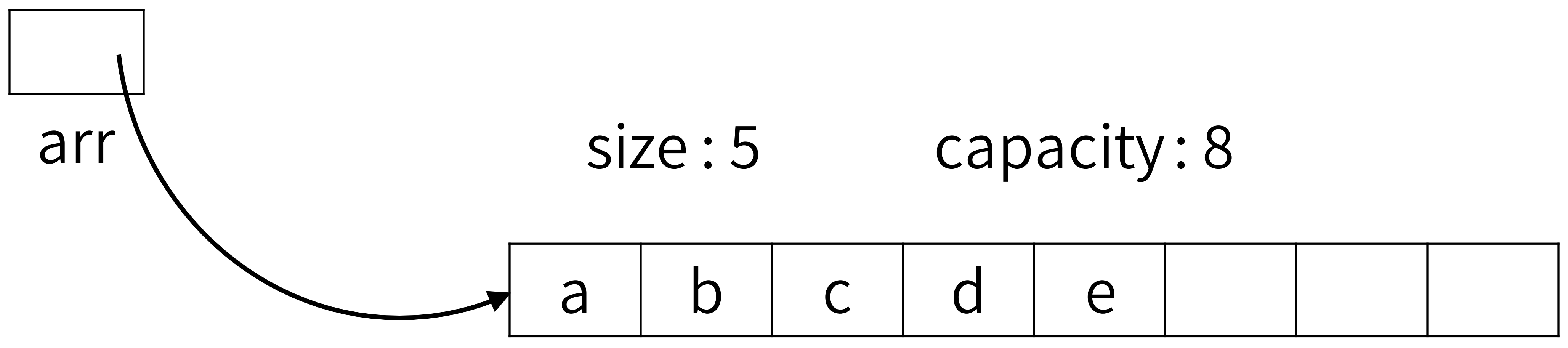
PushBack(e)

Example: Dynamic Array Resizing



PushBack(e)

Example: Dynamic Array Resizing



PushBack(e)

Implementation

```
Get($i$)

if $i < 0$ or $i \geq size$:
    ERROR: index out of range
return $arr[i]$
```

Implementation

```
Set($i, val$)
if $i < 0$ or $i \geq size$:
  ERROR: index out of range
$arr[i] = val$
```

Implementation

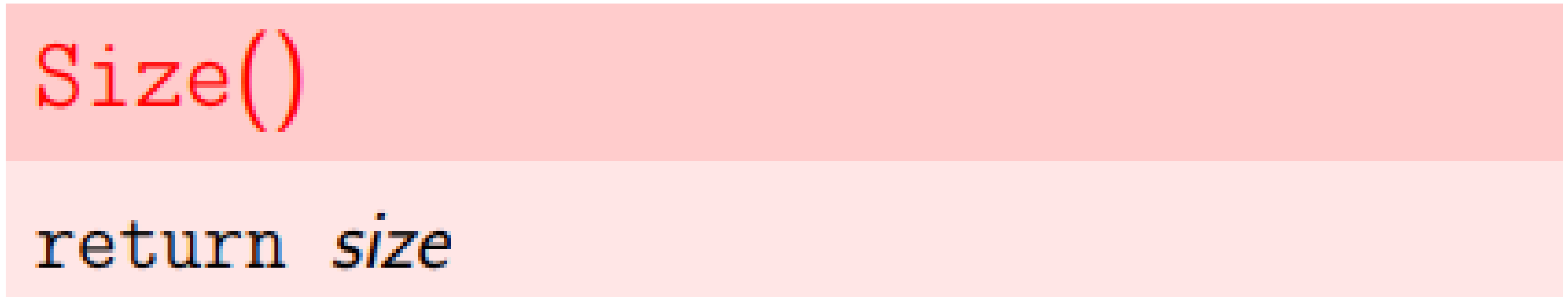
PushBack(val)

```
if $size = capacity$:
    allocate $new\_arr[2 \times capacity]$
    for $i$ from 0 to $size - 1$:
        $new\_arr[i] \leftarrow arr[i]$
    free $arr$
    $arr \leftarrow new\_arr$; $capacity \leftarrow 2 \times capacity$
$arr[size] \leftarrow val$
$size \leftarrow size + 1$
```

Implementation

```
Remove($i$)

if $i < 0$ or $i \geq size$:
    ERROR: index out of range
for $j$ from $i$ to $size - 2$:
    $arr[j] \leftarrow arr[j + 1]$
$size \leftarrow size - 1$
```

Implementation

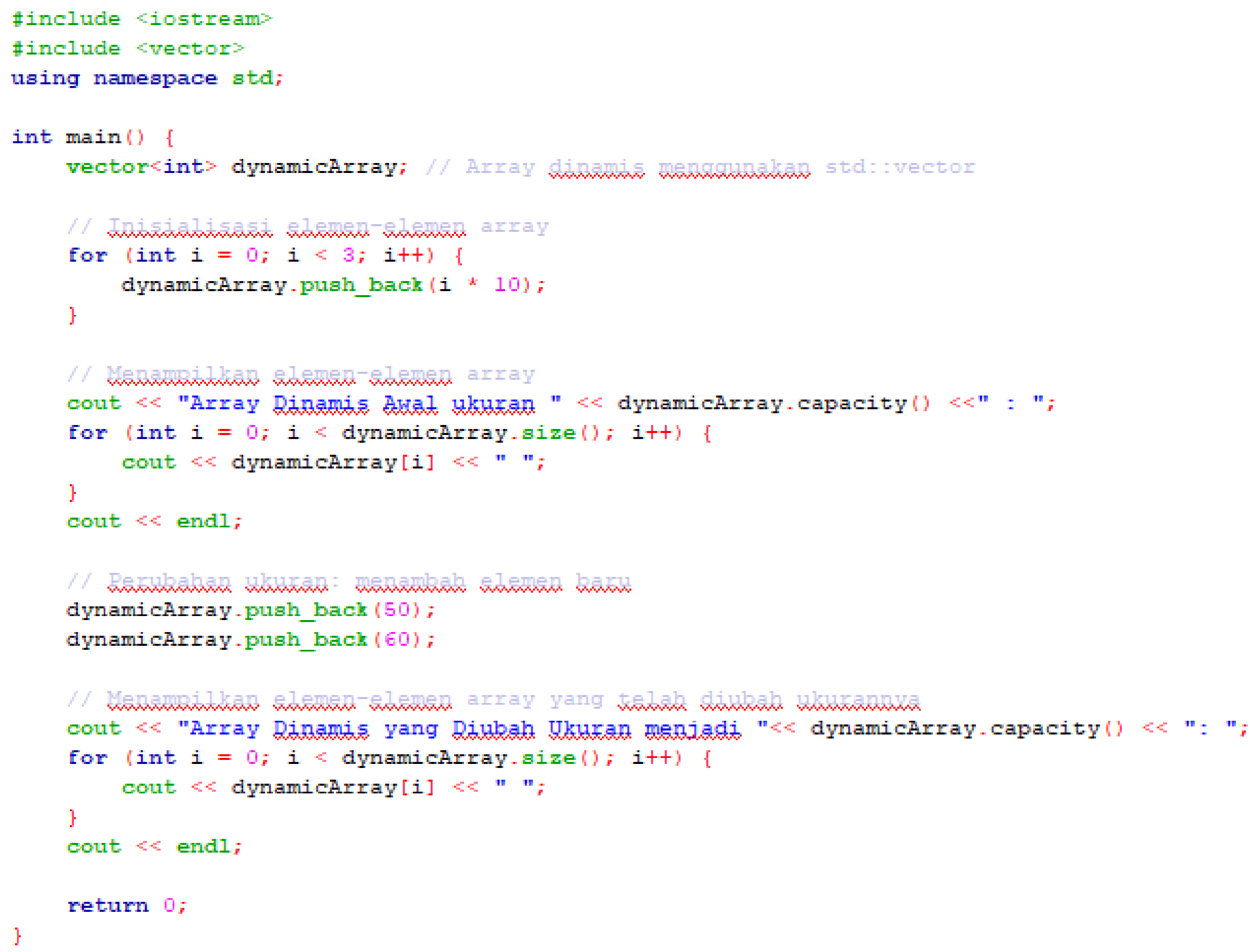


```
Size()
    return size
```

Common Implementation

- C++: `vector`
- Java: `ArrayList`
- Python: `list`

Implementation in C++



```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> dynamicArray; // Array dinamis menggunakan std::vector

    // Inisialisasi elemen-elemen array
    for (int i = 0; i < 3; i++) {
        dynamicArray.push_back(i * 10);
    }

    // Menampilkan elemen-elemen array
    cout << "Array Dinamis Awal ukuran " << dynamicArray.capacity() <<" : ";
    for (int i = 0; i < dynamicArray.size(); i++) {
        cout << dynamicArray[i] << " ";
    }
    cout << endl;

    // Perubahan ukuran: menambah elemen baru
    dynamicArray.push_back(50);
    dynamicArray.push_back(60);

    // Menampilkan elemen-elemen array yang telah diubah ukurannya
    cout << "Array Dinamis yang Diubah Ukuran menjadi "<< dynamicArray.capacity() << ": ";
    for (int i = 0; i < dynamicArray.size(); i++) {
        cout << dynamicArray[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Summary

- Dynamic array ≠ dynamically-allocated array
- Unlike static array, dynamic arrays can be resized
- Some space in dynamic arrays is wasted