§5 Arrays

ENGG1111

Computer Programming and Applications

Dirk Schnieders

Outline

- Array Basics
- Arrays and Functions
- Searching and Sorting
- 2D Arrays
- Char Variables
- Char Arrays

Arrays

- Used to process a collection of data
- All the data is of the same type
- E.g., price, temperatures, names

prices

0	1	2	3	4	5	6	7	8	9
20	6.5	10	1.5	3.15	6.5	99.9	12	49.5	40.9

temperatures

0	1	2	3	4
20	6.5	10	1.5	3.15

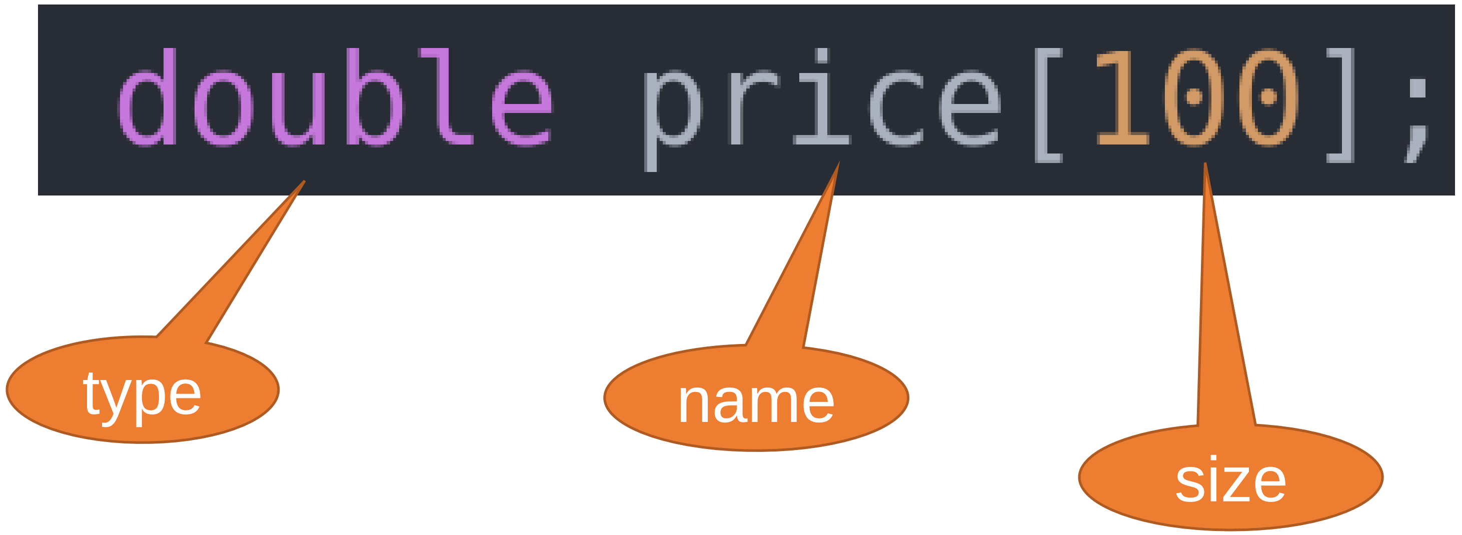
names

0	1	2	3
Dirk	Ben	Miko	Maria

Syntax

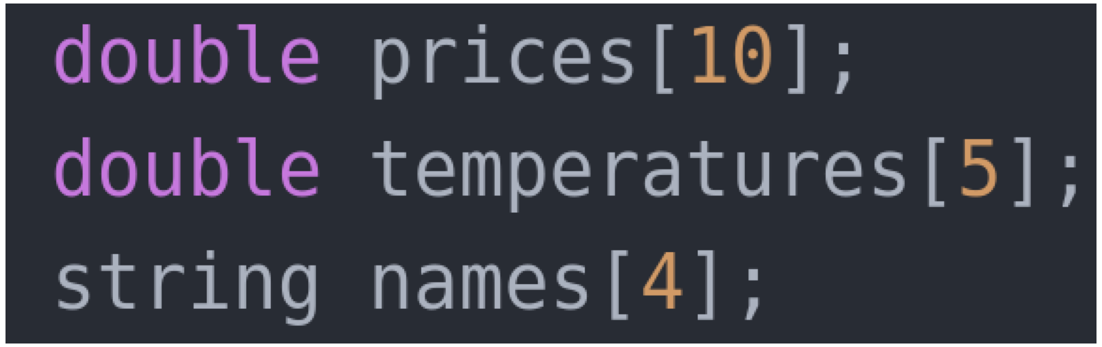
- Declaration
- Initialization
- Access the values stored in an array

Declaration



```
double price[100];
```

Declaration - Example



```
double prices[10];
double temperatures[5];
string names[4];
```

prices

0	1	2	3	4	5	6	7	8	9
20	6.5	10	1.5	3.15	6.5	99.9	12	49.5	40.9

temperatures

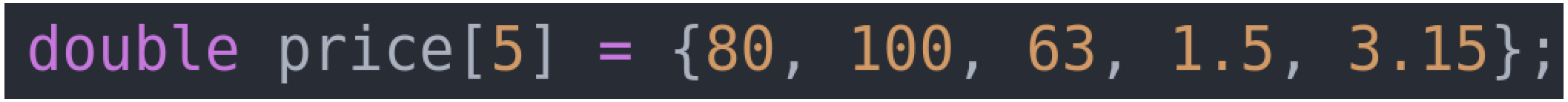
0	1	2	3	4
20	6.5	10	1.5	3.15

names

0	1	2	3
Dirk	Ben	Miko	Maria

Initialization

- An array may be initialized in its declaration by using an equal sign followed by a list of values enclosed within a pair of braces

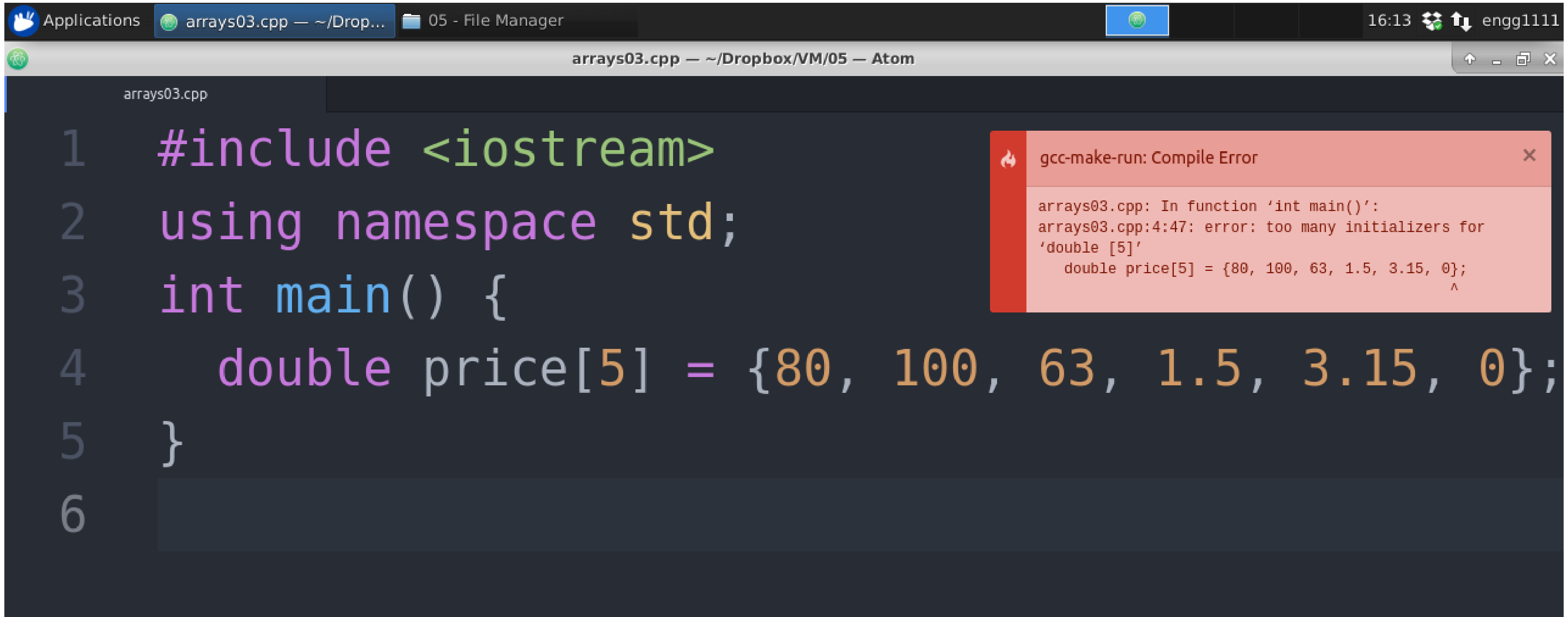


```
double price[5] = {80, 100, 63, 1.5, 3.15};
```

	0	1	2	3	4
price	80	100	63	1.5	3.15

Initialization

- The compiler will report an error if too many values are given in the initialization



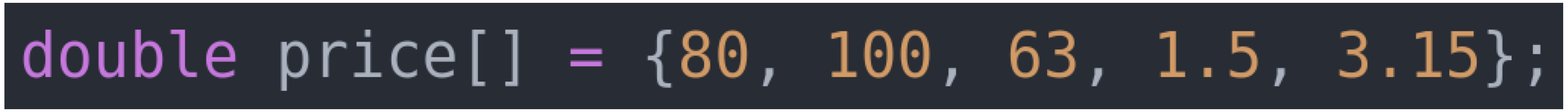
```cpp
#include <iostream>
using namespace std;
int main() {
  double price[5] = {80, 100, 63, 1.5, 3.15, 0};
}

```

gcc-make-run: Compile Error

```
arrays03.cpp: In function ‘int main()’:
arrays03.cpp:4:47: error: too many initializers for
‘double [5]’
   double price[5] = {80, 100, 63, 1.5, 3.15, 0};
                                               ^
```

Initialization

- If an array is initialized in its declaration, the size of the array may be omitted
- The array will automatically be declared to have the minimum size needed for the initialization values

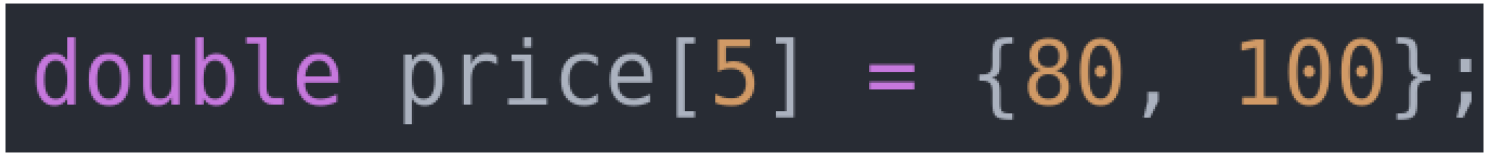


```
double price[] = {80, 100, 63, 1.5, 3.15};
```

	0	1	2	3	4
price	80	100	63	1.5	3.15

Initialization

- It is, however, legal to provide fewer values than the number of elements in the initialization
  - Those values will be used to initialize the first few elements
  - The remaining elements will be initialized to a zero

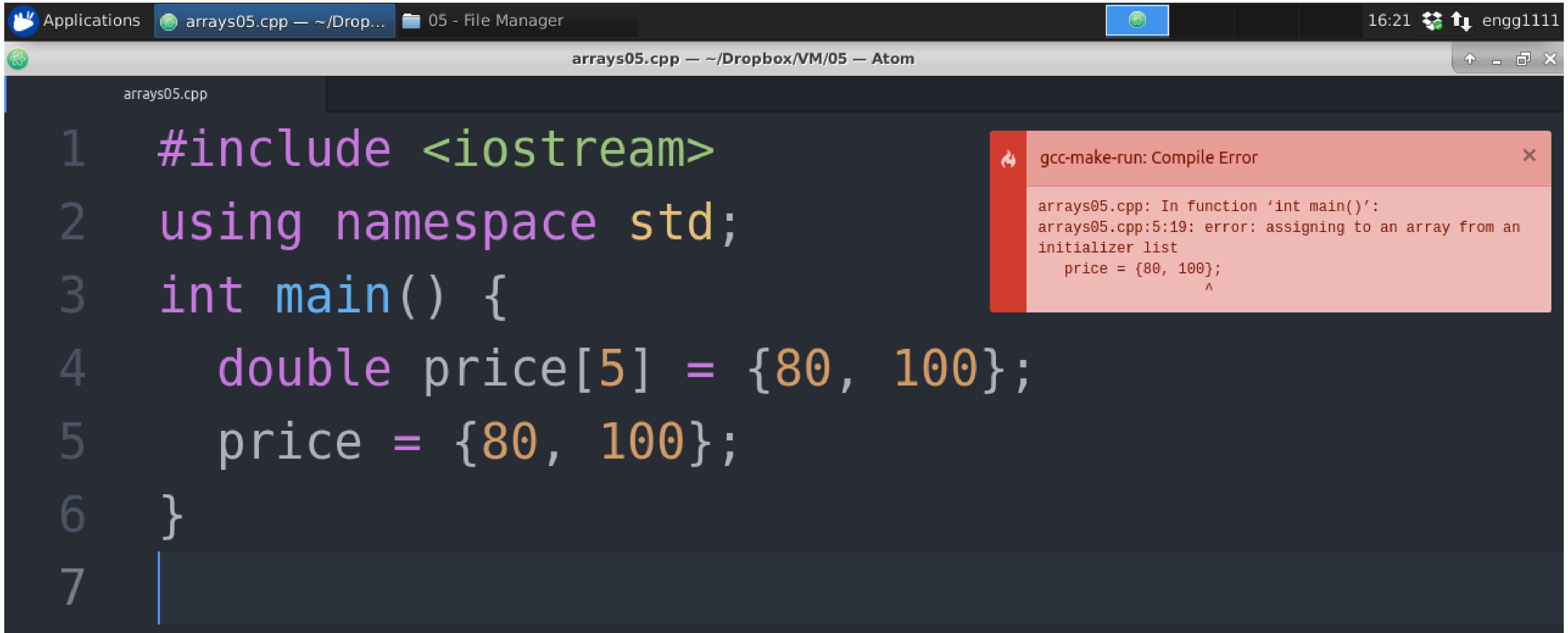


```
double price[5] = {80, 100};
```

	0	1	2	3	4
price	80	100	0	0	0

Initialization

- It is illegal to initialize or change the content of the whole array using an equal sign after its declaration



```cpp
#include <iostream>
using namespace std;
int main() {
  double price[5] = {80, 100};
  price = {80, 100};
}
```

```
arrays05.cpp: In function ‘int main()’:
arrays05.cpp:5:19: error: assigning to an array from an
initializer list
   price = {80, 100};
                   ^
```

Declaration - Example

```
double prices[] = {20, 6.5, 10, 1.5, 3.15, 6.5, 99.9, 12, 49.5, 40.9};
double temperatures[] = {20, 6.5, 10, 1.5, 3.15};
string names[] = {"Dirk", "Ben", "Miko", "Maria"};
```

prices

0	1	2	3	4	5	6	7	8	9
20	6.5	10	1.5	3.15	6.5	99.9	12	49.5	40.9

temperatures

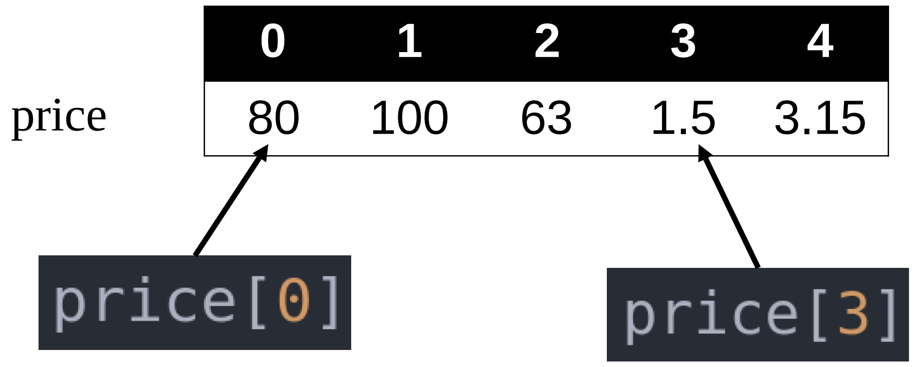
0	1	2	3	4
20	6.5	10	1.5	3.15

names

0	1	2	3
Dirk	Ben	Miko	Maria

Indexed Variable

- Each element of an array can be regarded as a variable of the base type, and is often called an indexed variable
- Array indexes always start from zero and end with the integer that is one less than the size of the array



Indexed Variable - Example

prices

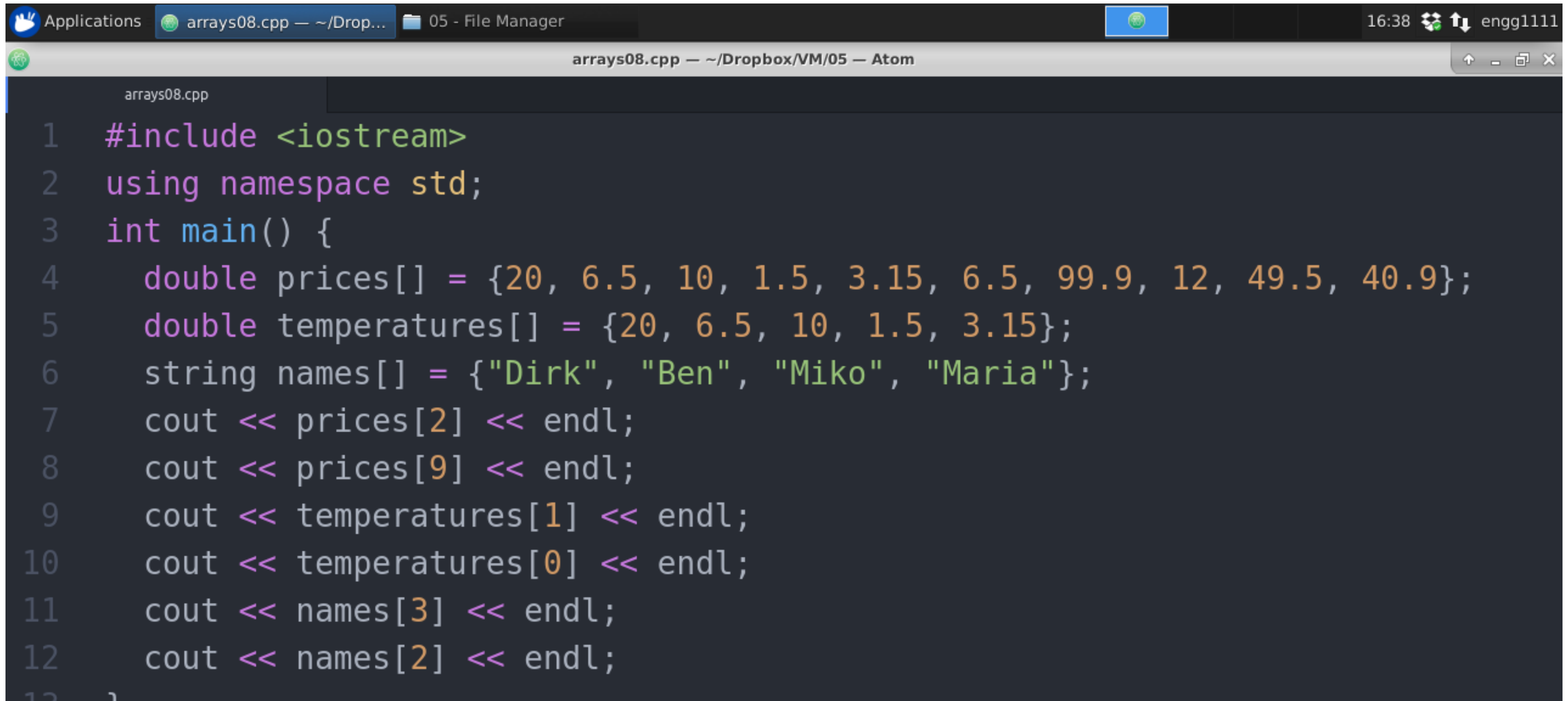
0	1	2	3	4	5	6	7	8	9
20	6.5	10	1.5	3.15	6.5	99.9	12	49.5	40.9

temperatures

0	1	2	3	4
20	6.5	10	1.5	3.15

names

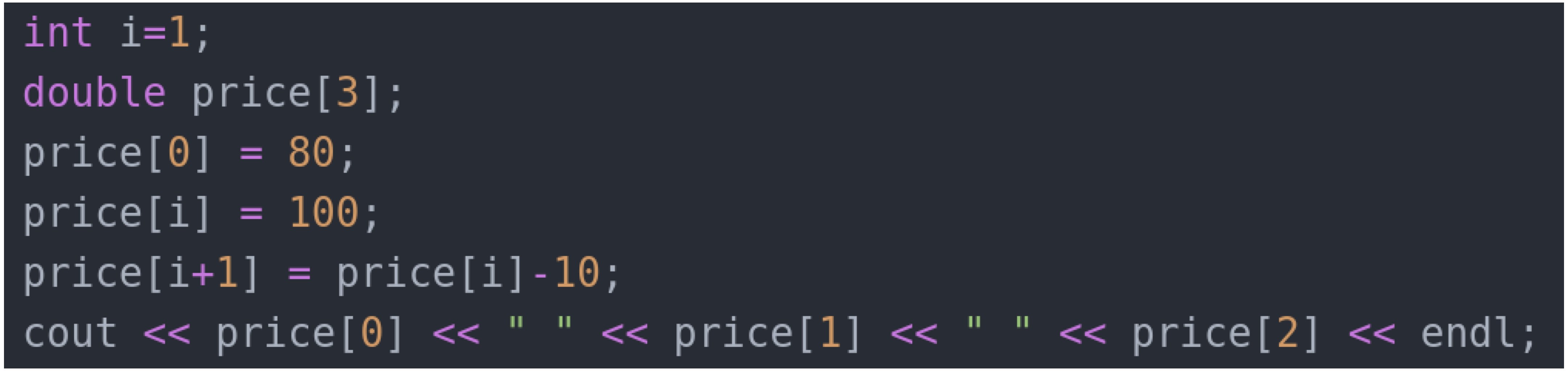
0	1	2	3
Dirk	Ben	Miko	Maria



```
#include <iostream>
using namespace std;
int main() {
  double prices[] = {20, 6.5, 10, 1.5, 3.15, 6.5, 99.9, 12, 49.5, 40.9};
  double temperatures[] = {20, 6.5, 10, 1.5, 3.15};
  string names[] = {"Dirk", "Ben", "Miko", "Maria"};
  cout << prices[2] << endl;
  cout << prices[9] << endl;
  cout << temperatures[1] << endl;
  cout << temperatures[0] << endl;
  cout << names[3] << endl;
  cout << names[2] << endl;
```

Indexed Variable

- An array index can be any integer expression, including integer constants and integer variables



```
int i=1;
double price[3];
price[0] = 80;
price[i] = 100;
price[i+1] = price[i]-10;
cout << price[0] << " " << price[1] << " " << price[2] << endl;
```

Quiz

- What is the index number of the last element of an array with 29 elements?
  - 29
  - 28
  - 0
  - Programmer-defined

Quiz

- Write code that will declare an array with 10 elements of name arr
- Assign the values 1 to 10 to the 10 elements
  - I.e.

	0	1	2	3	4	5	6	7	8	9
arr	1	2	3	4	5	6	7	8	9	10

- Use a for loop to output the content of the array

Quiz

- Describe the difference in the meaning of int a[5] and the meaning of a[4]
- What is the meaning of the [5] and [4] in each case?

Outline

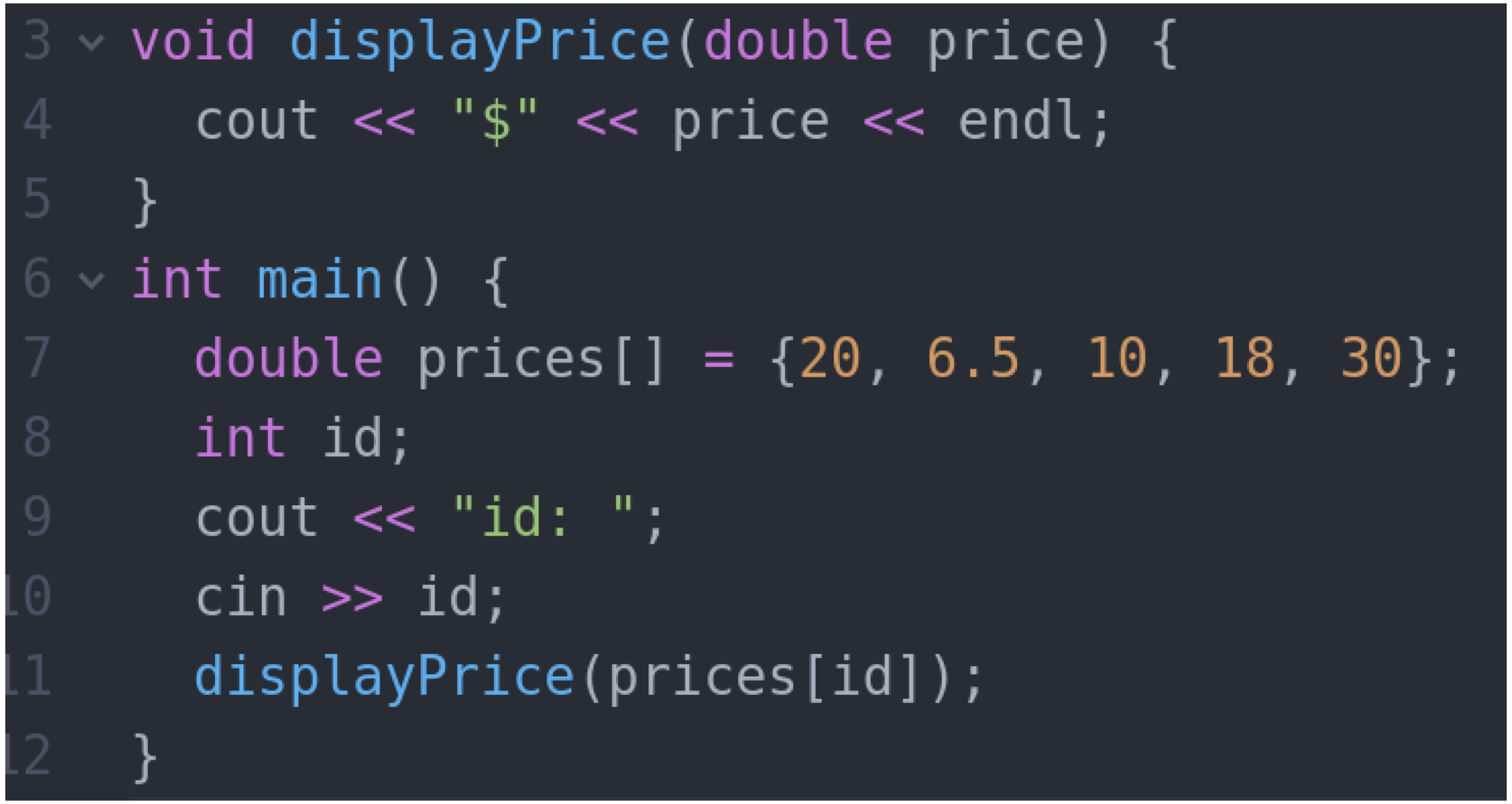
- Array Basics
- Arrays and Functions
- Searching and Sorting
- 2D Arrays
- Char Variables
- Char Arrays

Arrays and Functions

- You can (1) pass an indexed variable or (2) the entire array as an argument to a function

1. Indexed Variables as Arguments

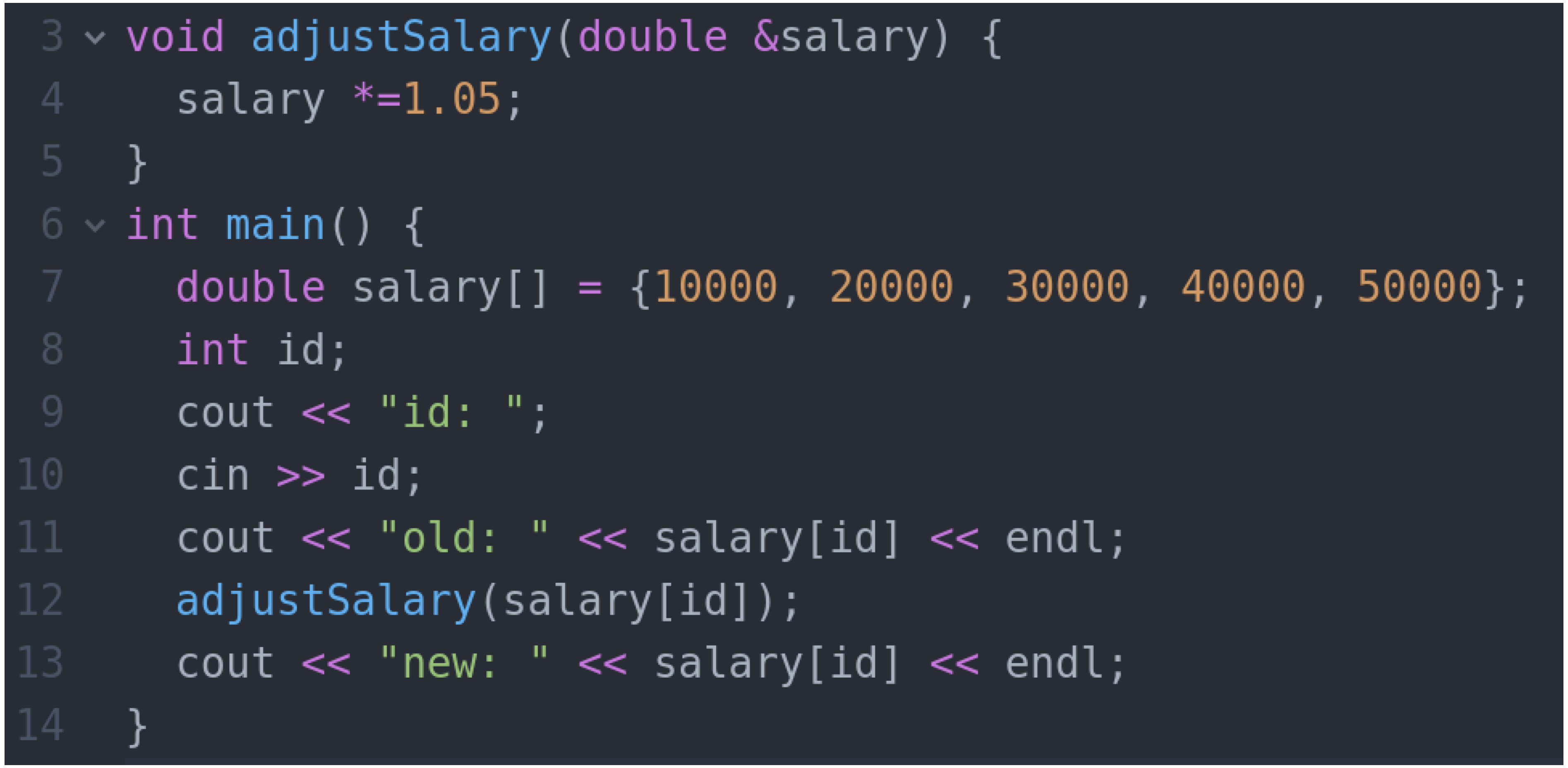
- An indexed variable can be pass-by-value



```
void displayPrice(double price) {
    cout << "$" << price << endl;
}
int main() {
    double prices[] = {20, 6.5, 10, 18, 30};
    int id;
    cout << "id: ";
    cin >> id;
    displayPrice(prices[id]);
}
```

1. Indexed Variables as Arguments

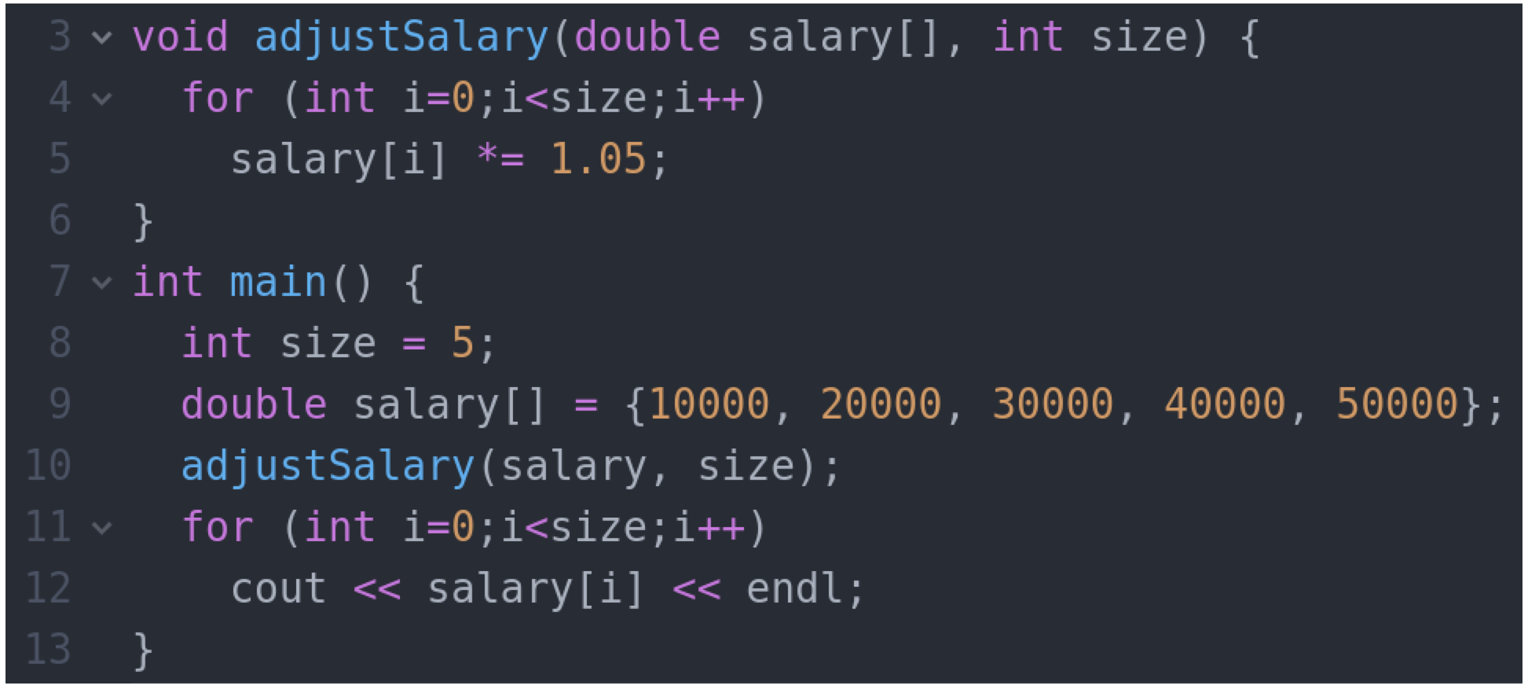
- An indexed variable can also be pass-by-reference



```cpp
void adjustSalary(double &salary) {
  salary *=1.05;
}
int main() {
  double salary[] = {10000, 20000, 30000, 40000, 50000};
  int id;
  cout << "id: ";
  cin >> id;
  cout << "old: " << salary[id] << endl;
  adjustSalary(salary[id]);
  cout << "new: " << salary[id] << endl;
}
```

2. Array Parameter

- If we pass the entire array into a function, the array parameter behaves very much like a pass-by-reference



```
void adjustSalary(double salary[], int size) {
  for (int i=0;i<size;i++)
    salary[i] *= 1.05;
}
int main() {
  int size = 5;
  double salary[] = {10000, 20000, 30000, 40000, 50000};
  adjustSalary(salary, size);
  for (int i=0;i<size;i++)
    cout << salary[i] << endl;
}
```

Task

- Write a function named countNum2s that takes as input an array of integers and an integer that specifies how many entries are in the array
- The function should return the number of 2’s in the array

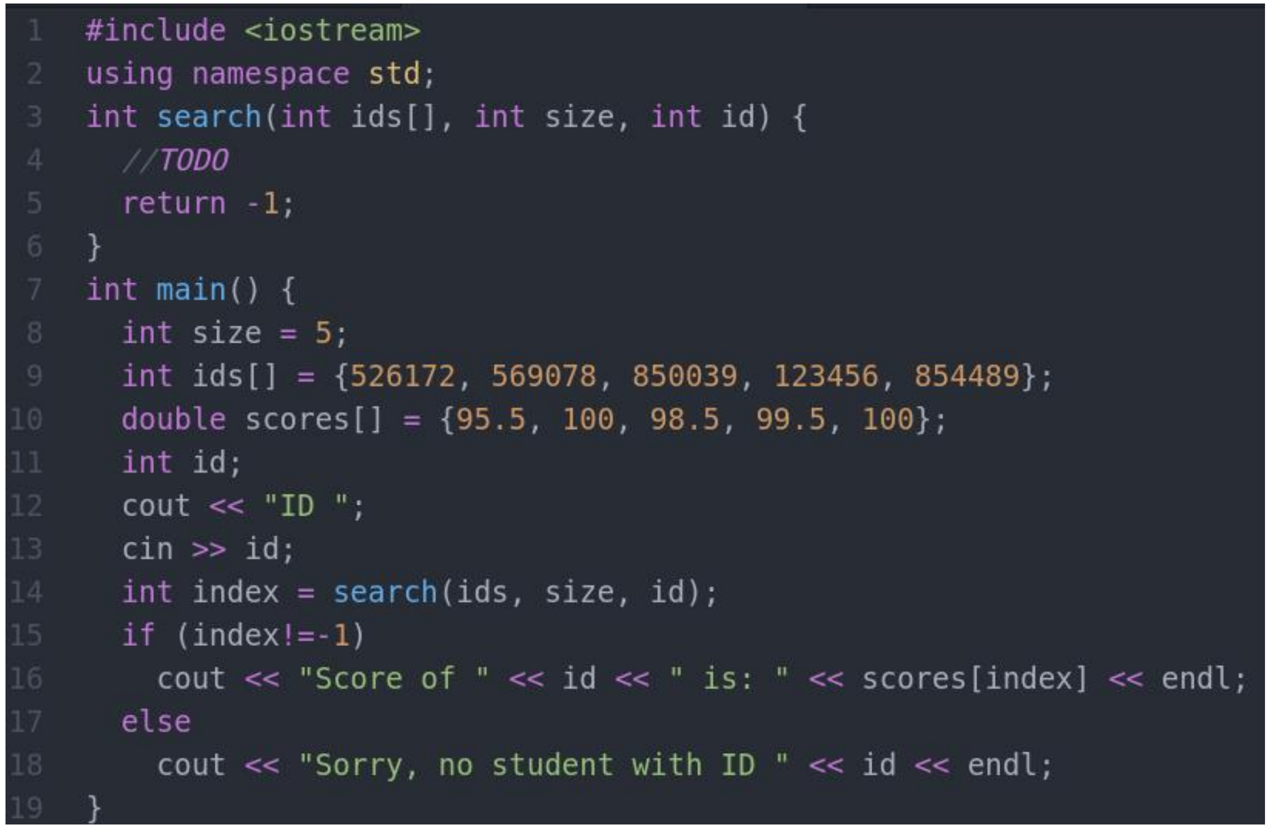
Outline

- Array Basics
- Arrays and Functions
- Searching and Sorting
- 2D Arrays
- Char Variables
- Char Arrays

Searching and Sorting

Searching an Array

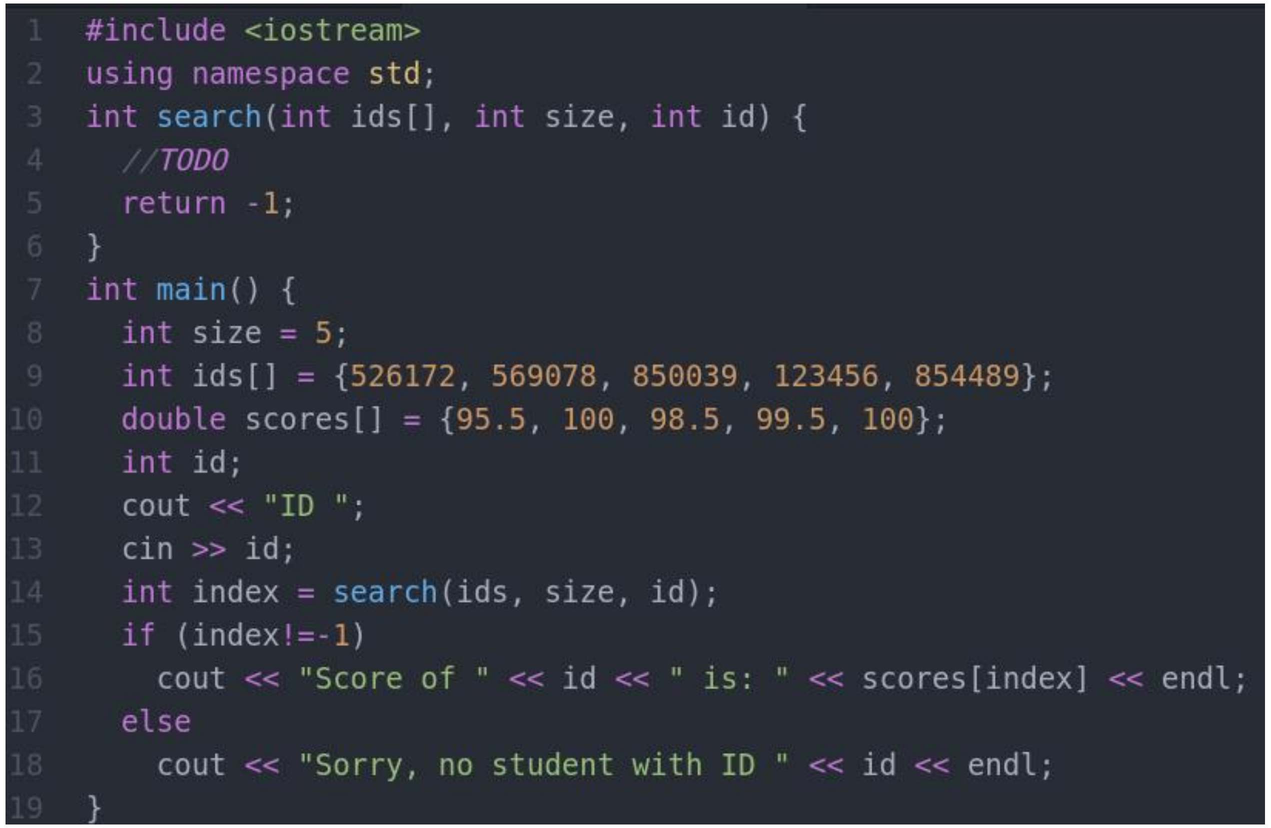
- A common programming task is to search an array for a given value



```cpp
#include <iostream>
using namespace std;
int search(int ids[], int size, int id) {
  //TODO
  return -1;
}
int main() {
  int size = 5;
  int ids[] = {526172, 569078, 850039, 123456, 854489};
  double scores[] = {95.5, 100, 98.5, 99.5, 100};
  int id;
  cout << "ID ";
  cin >> id;
  int index = search(ids, size, id);
  if (index!=-1)
    cout << "Score of " << id << " is: " << scores[index] << endl;
  else
    cout << "Sorry, no student with ID " << id << endl;
}
```

Task

- Implement the body of the function search



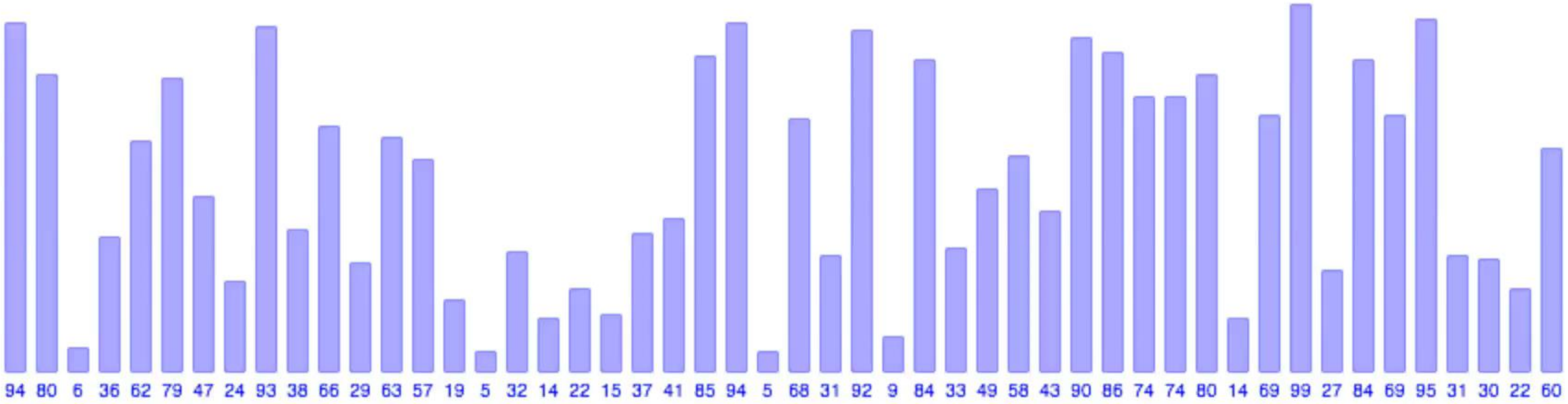
```cpp
#include <iostream>
using namespace std;
int search(int ids[], int size, int id) {
  //TODO
  return -1;
}
int main() {
  int size = 5;
  int ids[] = {526172, 569078, 850039, 123456, 854489};
  double scores[] = {95.5, 100, 98.5, 99.5, 100};
  int id;
  cout << "ID ";
  cin >> id;
  int index = search(ids, size, id);
  if (index!=-1)
    cout << "Score of " << id << " is: " << scores[index] << endl;
  else
    cout << "Sorry, no student with ID " << id << endl;
}
```

Sorting an Array

- Another widely encountered programming task is to sort the values in an array
- One of the easiest sorting algorithms is called *selection sort*
  - To sort an array a[] of n elements, perform n-1 iterations
  - In the i-th iteration, select the i-th smallest element and swap it with the i-th indexed variable a[i-1]
    - e.g., in the 1st iteration, find the smallest element, swap it with the slot a[0]

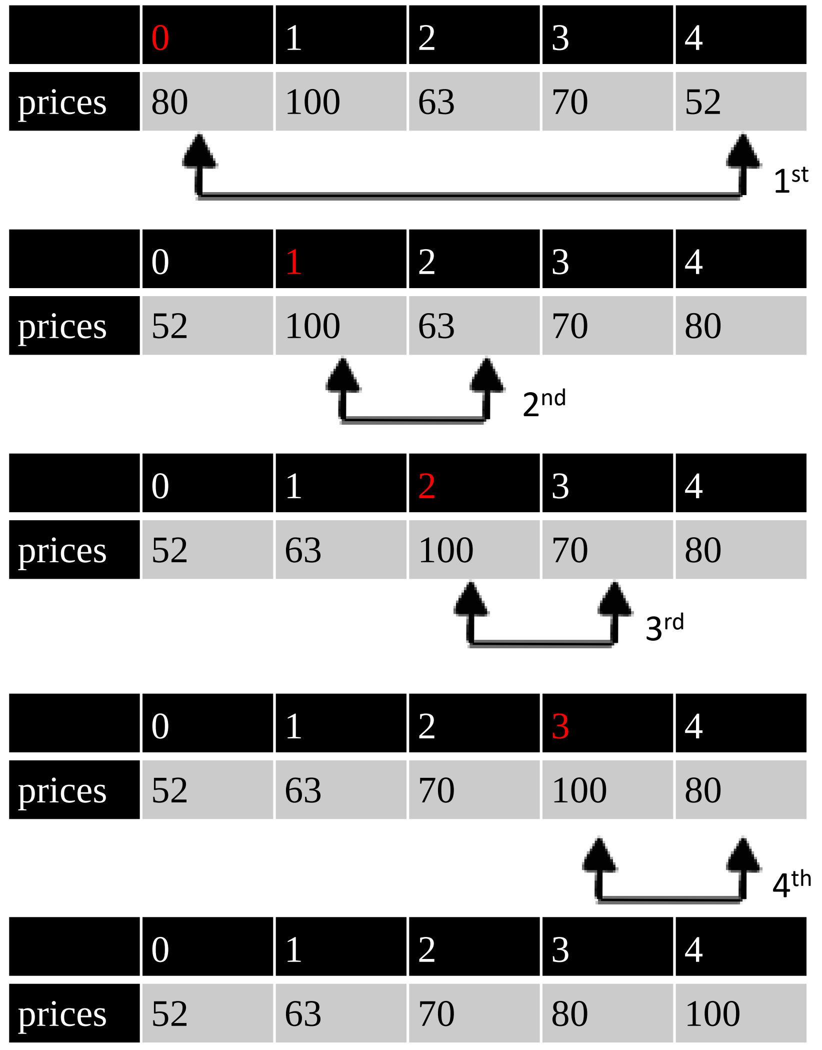
Sorting an Array

- Another widely encountered programming task is to sort the values in an array
- One of the easiest sorting algorithms is called selection sort
- To sort an array a[] of n elements, perform n-1 iterations
- In the i-th iteration, select the i-th smallest element and swap it with the i-th indexed variable a[i-1]
    - e.g., in the 1st iteration, find the smallest element, swap it with the slot a[0]


94
80
6
36
62
79
47
24
93
38
66
29
63
57
19
5
32
14
22
15
37
41
85
94
5
68
31
92
9
84
33
49
58
43
90
86
74
74
80
14
69
99
27
84
69
95
31
30
22
60

Selection Sort

- Find the 1st smallest element in the array and swap it with the element in the 1st slot
- Find the 2nd smallest element in the array and swap it with the element in the 2nd slot
- Find the 3rd smallest element in the array and swap it with the element in the 3rd slot
- Find the 4th smallest element in the array and swap it with the element in the 4th slot



	0	1	2	3	4
prices	80	100	63	70	52

1st

	0	1	2	3	4
prices	52	100	63	70	80

2nd

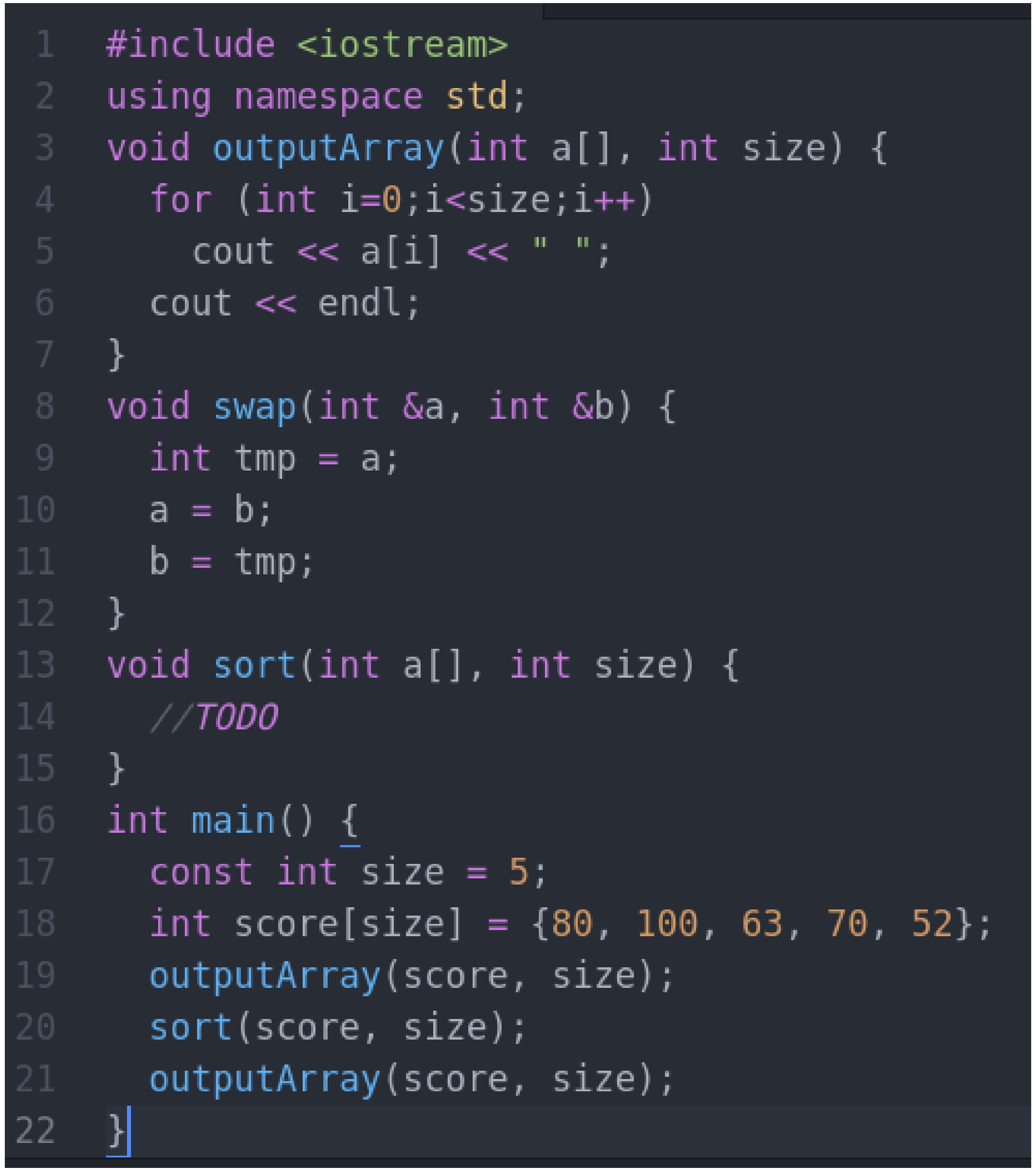
	0	1	2	3	4
prices	52	63	100	70	80

3rd

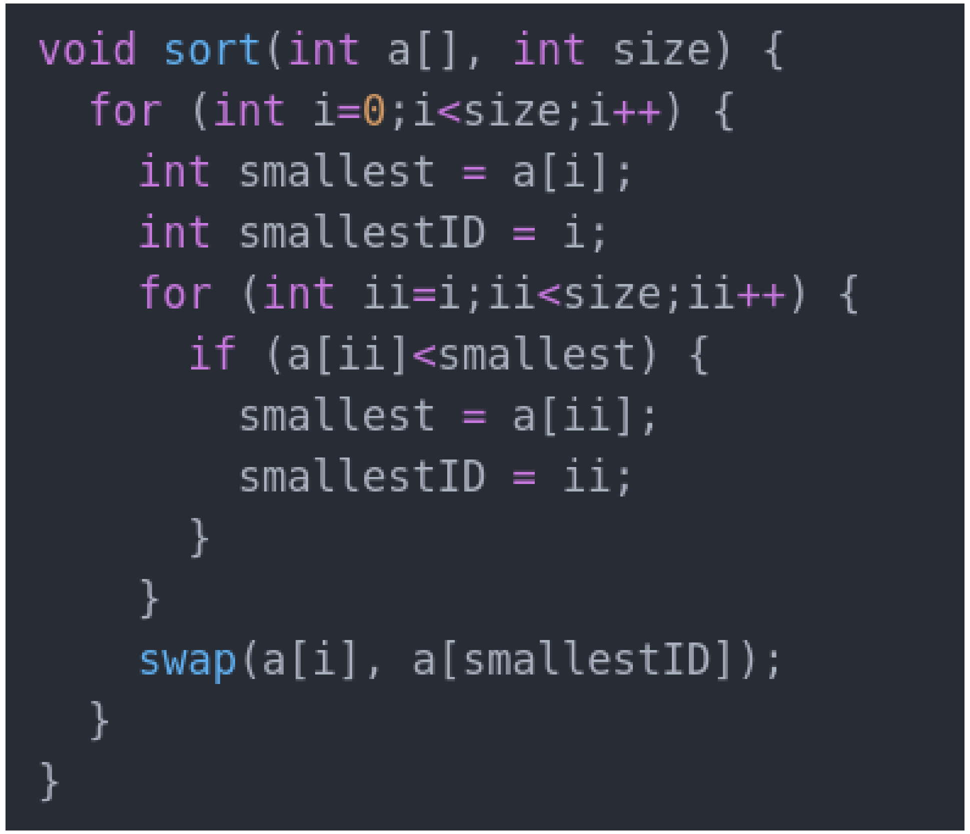
	0	1	2	3	4
prices	52	63	70	100	80

4th

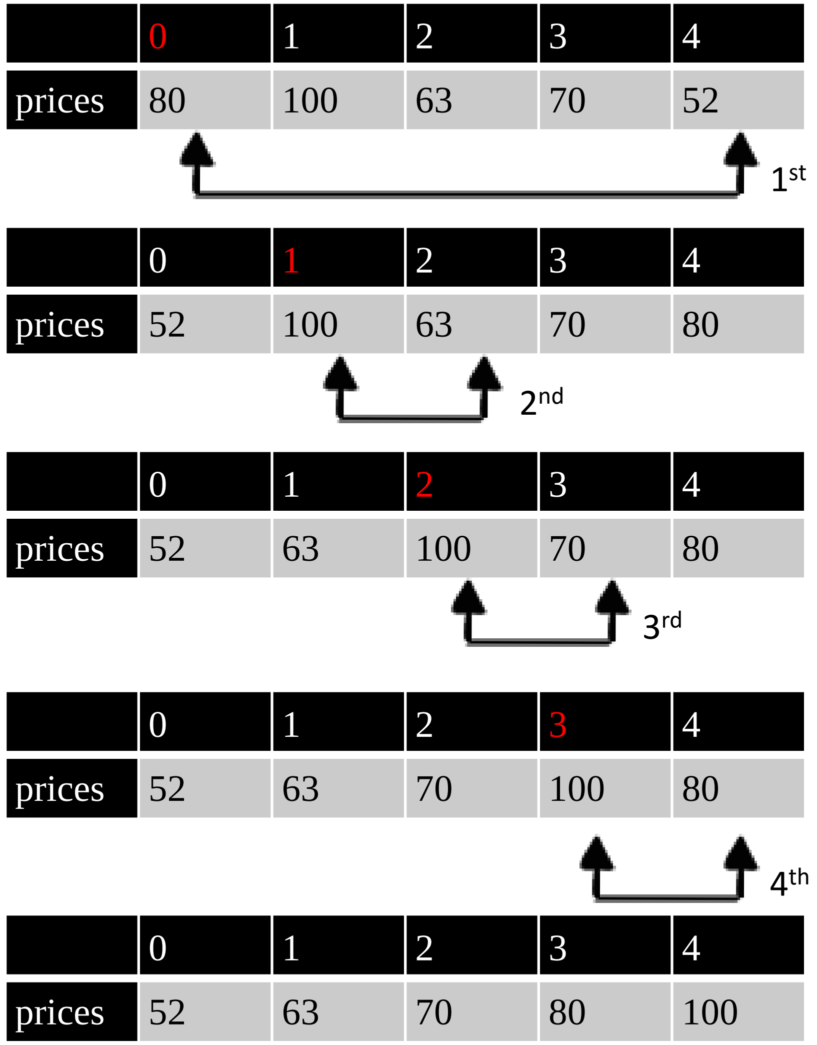
	0	1	2	3	4
prices	52	63	70	80	100



```cpp
#include <iostream>
using namespace std;
void outputArray(int a[], int size) {
  for (int i=0;i<size;i++)
    cout << a[i] << " ";
  cout << endl;
}
void swap(int &a, int &b) {
  int tmp = a;
  a = b;
  b = tmp;
}
void sort(int a[], int size) {
  //TODO
}
int main() {
  const int size = 5;
  int score[size] = {80, 100, 63, 70, 52};
  outputArray(score, size);
  sort(score, size);
  outputArray(score, size);
}
```



```
void sort(int a[], int size) {
  for (int i=0;i<size;i++) {
    int smallest = a[i];
    int smallestID = i;
    for (int ii=i;ii<size;ii++) {
      if (a[ii]<smallest) {
        smallest = a[ii];
        smallestID = ii;
      }
    }
    swap(a[i], a[smallestID]);
  }
}
```



	0	1	2	3	4
prices	80	100	63	70	52

	0	1	2	3	4
prices	52	100	63	70	80

	0	1	2	3	4
prices	52	63	100	70	80

	0	1	2	3	4
prices	52	63	70	100	80

	0	1	2	3	4
prices	52	63	70	80	100

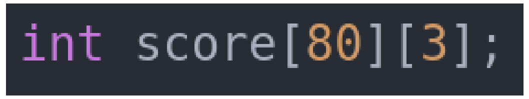
Outline

- Array Basics
- Arrays and Functions
- Searching and Sorting
- 2D Arrays
- Char Variables
- Char Arrays

2D Arrays

2D Arrays - Declaration

- Consider the following declaration that defines a two dimensional (2D) array with (80 x 3) = 240 cells



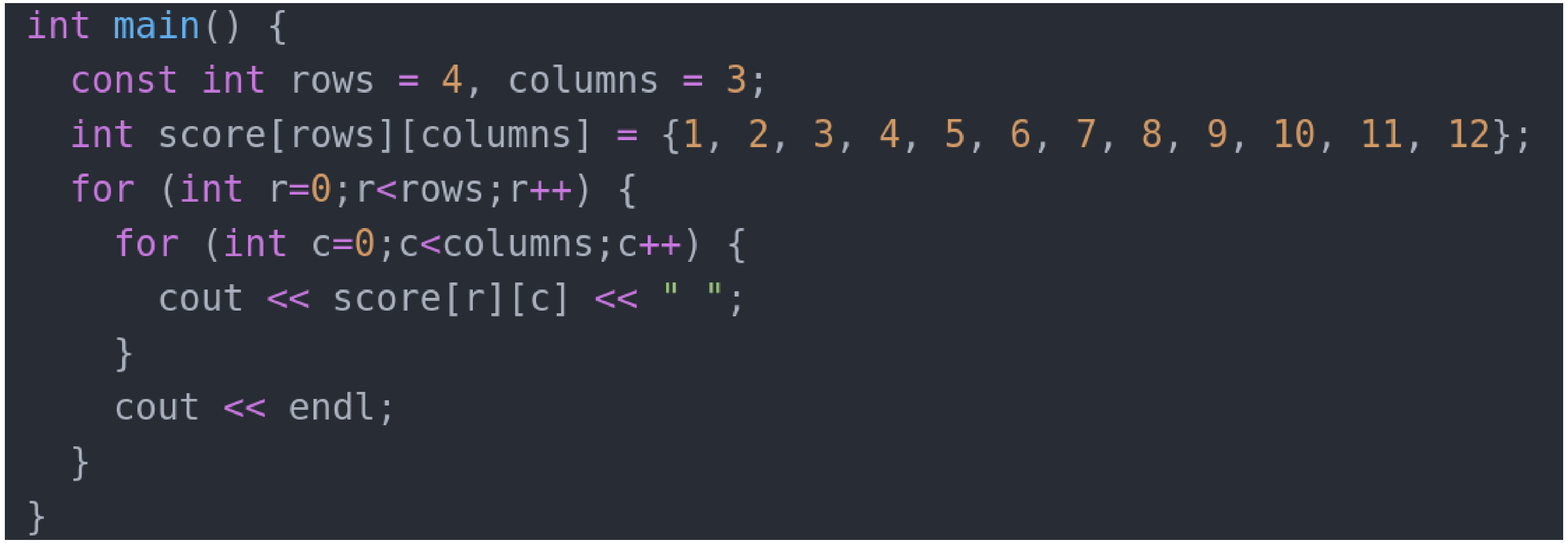
```
int score[80][3];
```

- A 2D array can be visualized as a matrix, with the first index giving the row and the second index giving the column

	0	**1**	**2**
0	score[0][0]	score[0][1]	score[0][2]
1	score[1][0]	score[1][1]	score[1][2]
2	score[2][0]	score[2][1]	score[2][2]
3	score[3][0]	score[3][1]	score[3][2]
…	...	…	…
79	score[79][0]	score[79][1]	score[79][2]

2D Arrays - Initialization

- Similar to the 1D case, a 2D array can be initialized in its declaration by using an equal sign followed by a list of values along each row

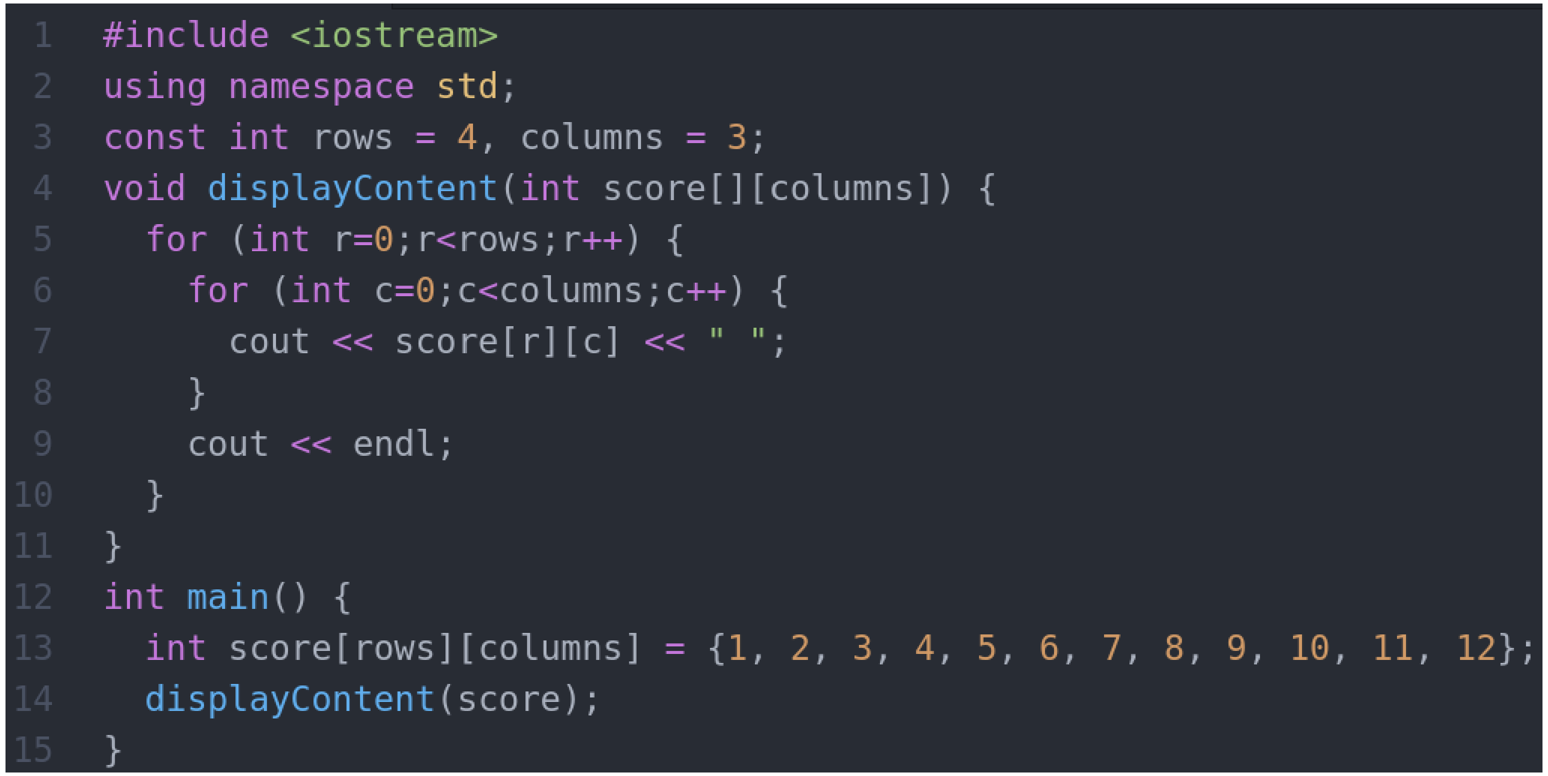


```
int main() {
  const int rows = 4, columns = 3;
  int score[rows][columns] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
  for (int r=0;r<rows;r++) {
    for (int c=0;c<columns;c++) {
      cout << score[r][c] << " ";
    }
    cout << endl;
  }
}
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

2D Arrays

- When a 2D array parameter is given in a function declaration, the size of the first dimension is not given, but the remaining dimension size must be given in square brackets



```cpp
#include <iostream>
using namespace std;
const int rows = 4, columns = 3;
void displayContent(int score[][columns]) {
  for (int r=0;r<rows;r++) {
    for (int c=0;c<columns;c++) {
      cout << score[r][c] << " ";
    }
    cout << endl;
  }
}
int main() {
  int score[rows][columns] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
  displayContent(score);
}
```

Outline

- Array Basics
- Arrays and Functions
- Searching and Sorting
- 2D Arrays
- Char Variables
- Char Arrays

Char Variables

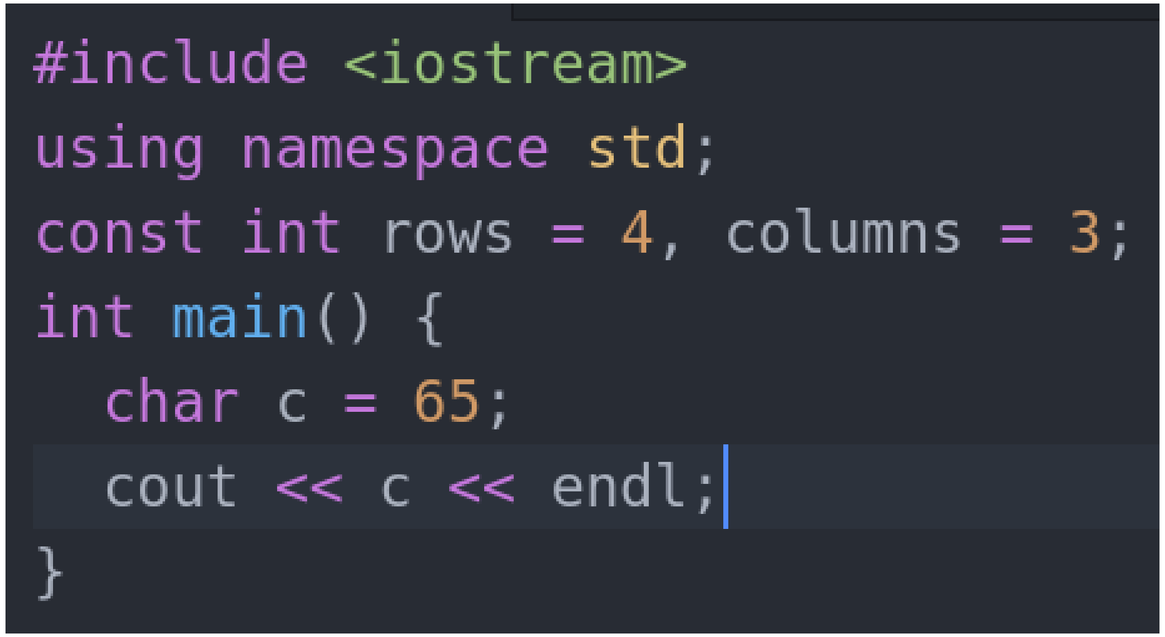
Variable Type

- Tells the computer how to interpret the data stored in a variable
- Determines the size of storage needed to store the data
- Some (not all) basic variable types in C++

Name	Description	Size	Range
char	Character or small integer	1 byte	0 to 255 or -128 to 127
bool	Boolean value	1 byte	True(1) or False(0)
int	Integer	4 bytes	-2147483648 to 2147483647
double	Double precision floating point number	8 bytes	~(15 digits)

Char

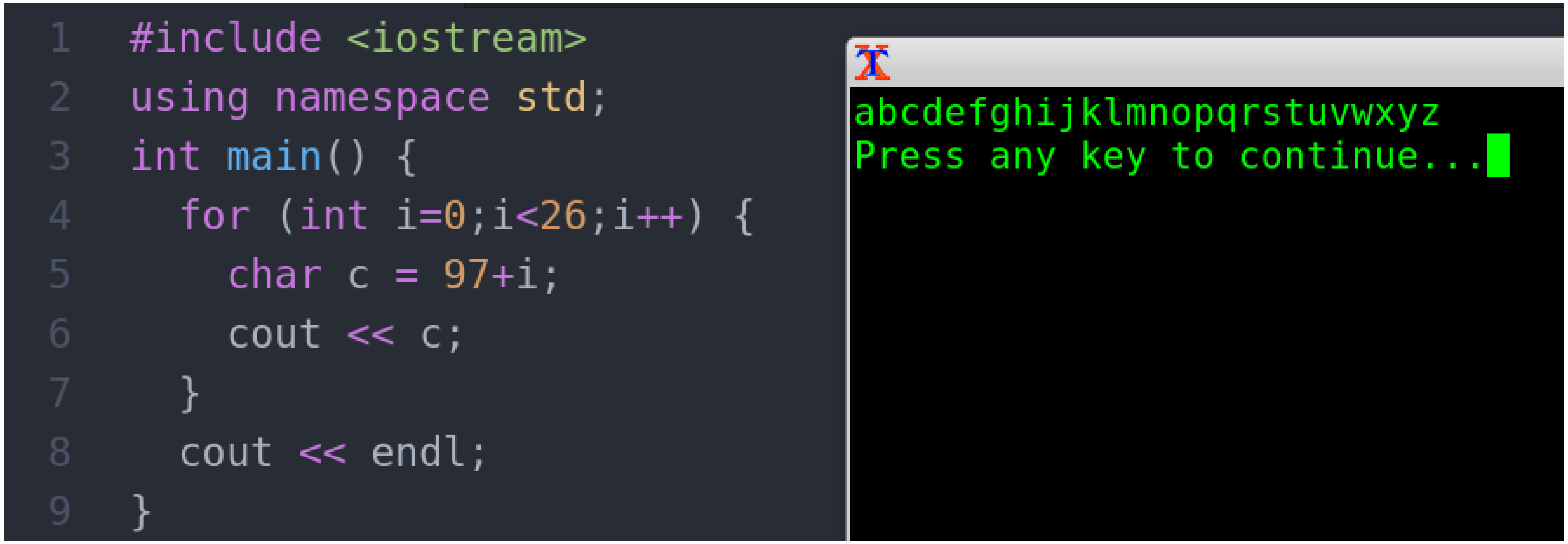
- Internally, a character (char) is represented as an integer (int)



```
#include <iostream>
using namespace std;
const int rows = 4, columns = 3;
int main() {
  char c = 65;
  cout << c << endl;
}
```

Char

- Internally, a character (char) is represented as an integer (int)

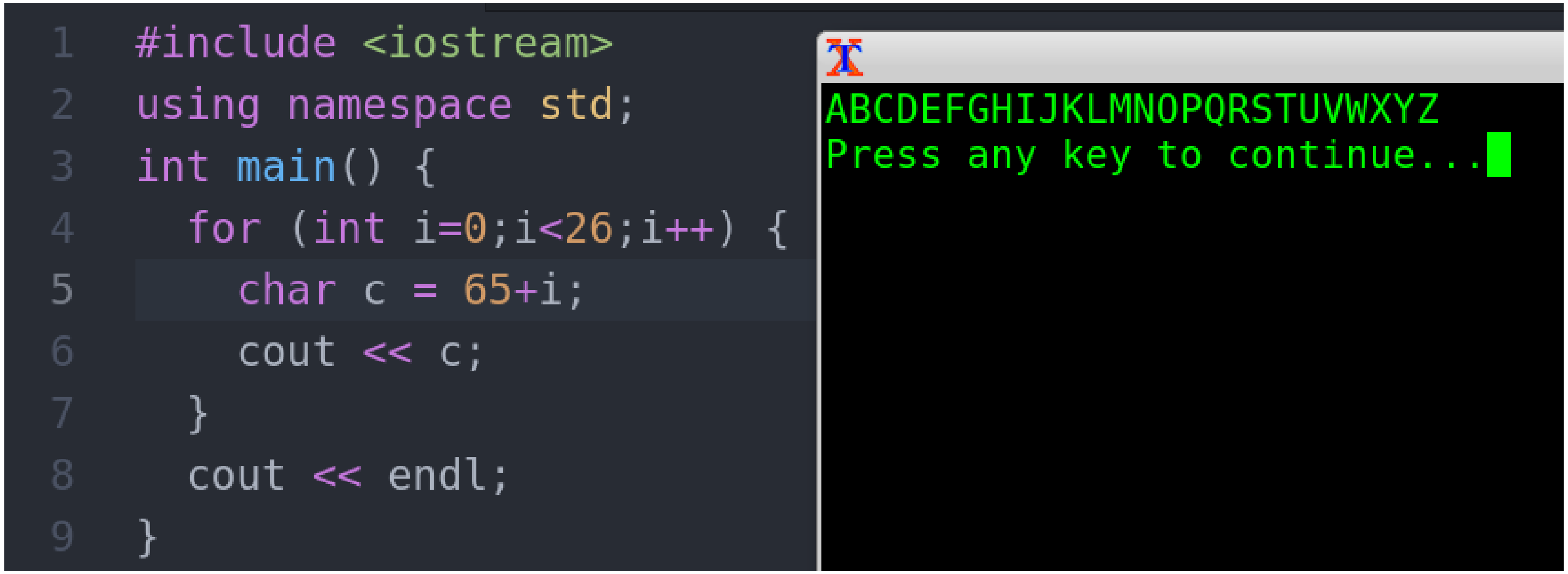


```cpp
#include <iostream>
using namespace std;
int main() {
  for (int i=0;i<26;i++) {
    char c = 97+i;
    cout << c;
  }
  cout << endl;
}
```

```
abcdefghijklmnopqrstuvwxyz
Press any key to continue...
```

Char

- Internally, a character (char) is represented as an integer (int)

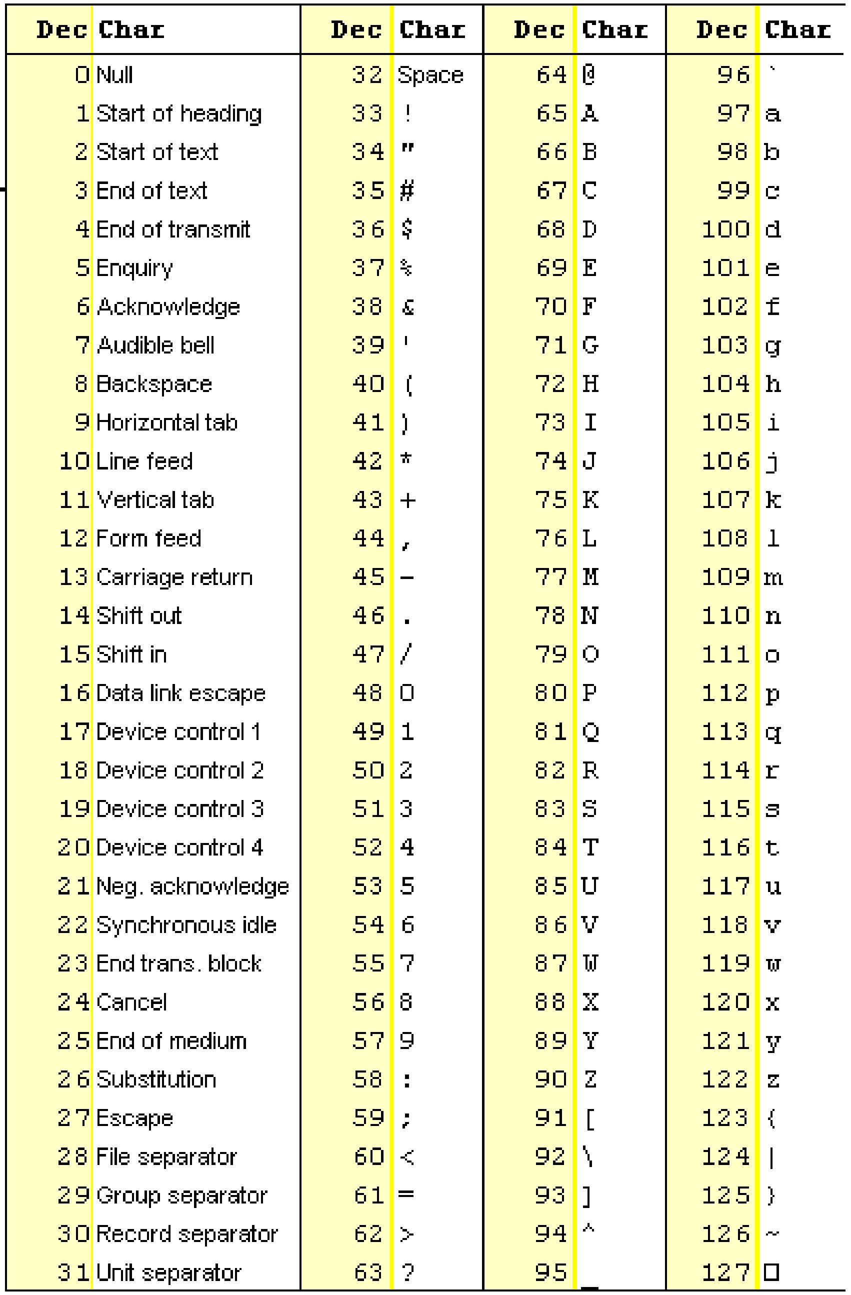


```cpp
#include <iostream>
using namespace std;
int main() {
  for (int i=0;i<26;i++) {
    char c = 65+i;
    cout << c;
  }
  cout << endl;
}
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Press any key to continue...
```

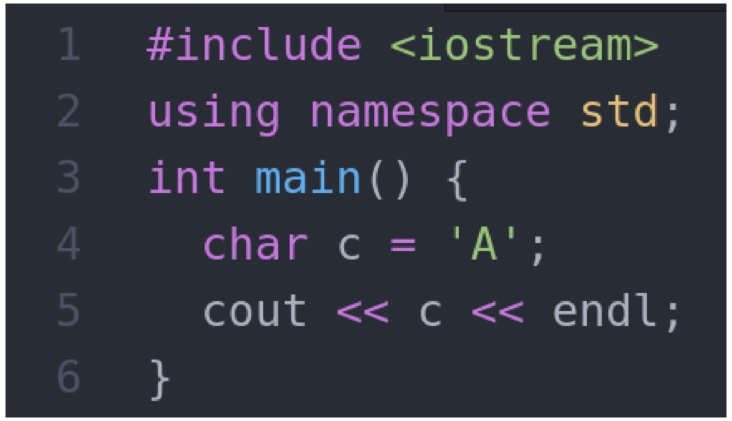
ASCII

- Char values are represented by ASCII values
- ASCII character table for the first 128 character
- Remember that there are 256 total
- Some systems are only able to display the first 128 symbols

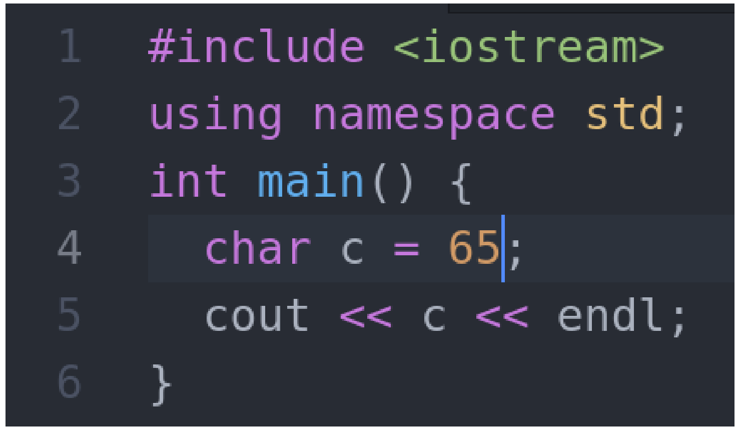


Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	Null	32	Space	64	@	96	`
1	Start of heading	33	!	65	A	97	a
2	Start of text	34	"	66	B	98	b
3	End of text	35	#	67	C	99	c
4	End of transmit	36	$	68	D	100	d
5	Enquiry	37	%	69	E	101	e
6	Acknowledge	38	&	70	F	102	f
7	Audible bell	39	'	71	G	103	g
8	Backspace	40	(	72	H	104	h
9	Horizontal tab	41	)	73	I	105	i
10	Line feed	42	*	74	J	106	j
11	Vertical tab	43	+	75	K	107	k
12	Form feed	44	,	76	L	108	l
13	Carriage return	45	-	77	M	109	m
14	Shift out	46	.	78	N	110	n
15	Shift in	47	/	79	O	111	o
16	Data link escape	48	0	80	P	112	p
17	Device control 1	49	1	81	Q	113	q
18	Device control 2	50	2	82	R	114	r
19	Device control 3	51	3	83	S	115	s
20	Device control 4	52	4	84	T	116	t
21	Neg. acknowledge	53	5	85	U	117	u
22	Synchronous idle	54	6	86	V	118	v
23	End trans. block	55	7	87	W	119	w
24	Cancel	56	8	88	X	120	x
25	End of medium	57	9	89	Y	121	y
26	Substitution	58	:	90	Z	122	z
27	Escape	59	;	91	[	123	{
28	File separator	60	<	92	\	124	|
29	Group separator	61	=	93	]	125	}
30	Record separator	62	>	94	^	126	~
31	Unit separator	63	?	95	_	127	□

Char



```cpp
#include <iostream>
using namespace std;
int main() {
  char c = 'A';
  cout << c << endl;
}
```



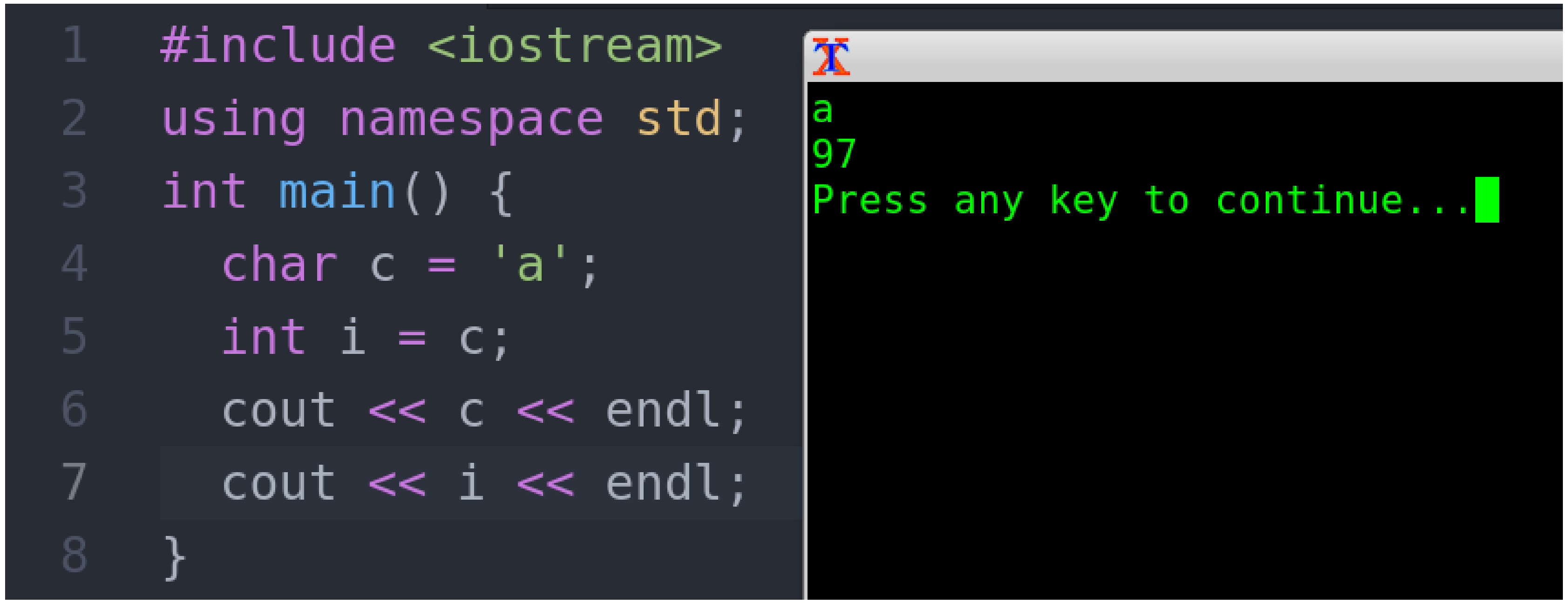
```cpp
#include <iostream>
using namespace std;
int main() {
  char c = 65;
  cout << c << endl;
}
```

The single quotes 'A' tells the compiler that A is a character

If we assign an integer value to a char, say 65, the compiler will treat that as the ASCII code

Char

- It is okay to use an int to store the value of a char
- The ASCII value (an int) of the char will be stored

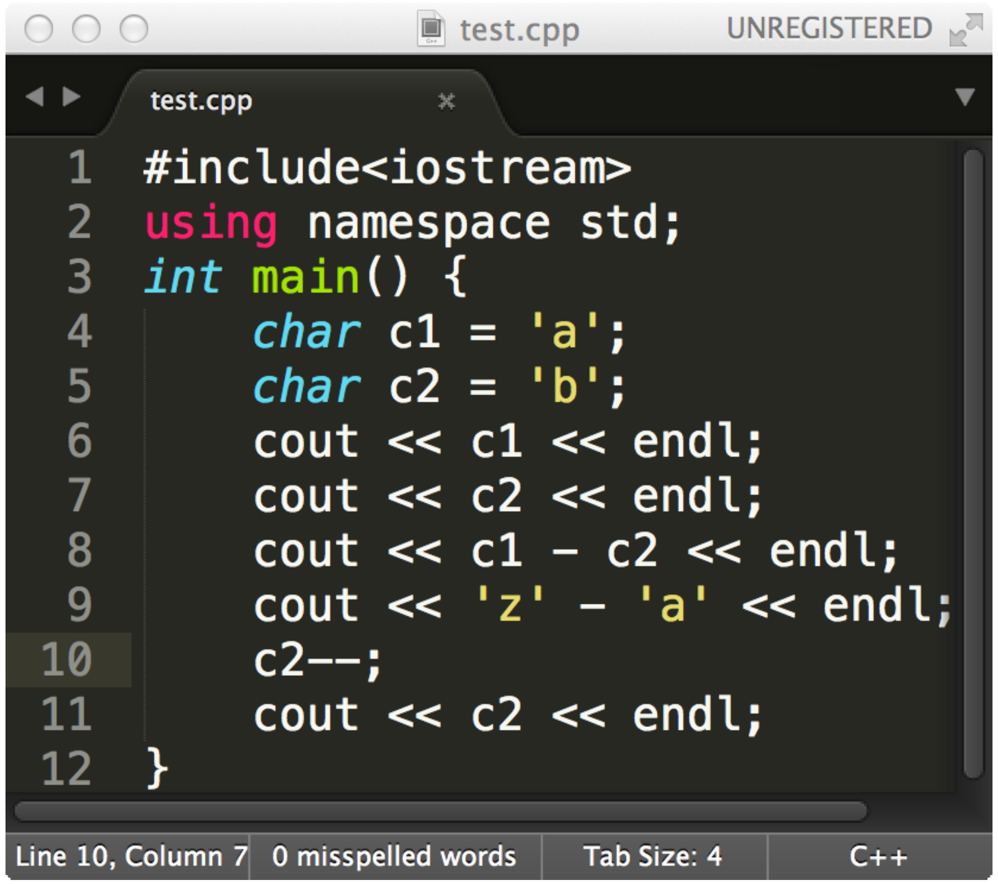


```cpp
#include <iostream>
using namespace std;
int main() {
  char c = 'a';
  int i = c;
  cout << c << endl;
  cout << i << endl;
}
```

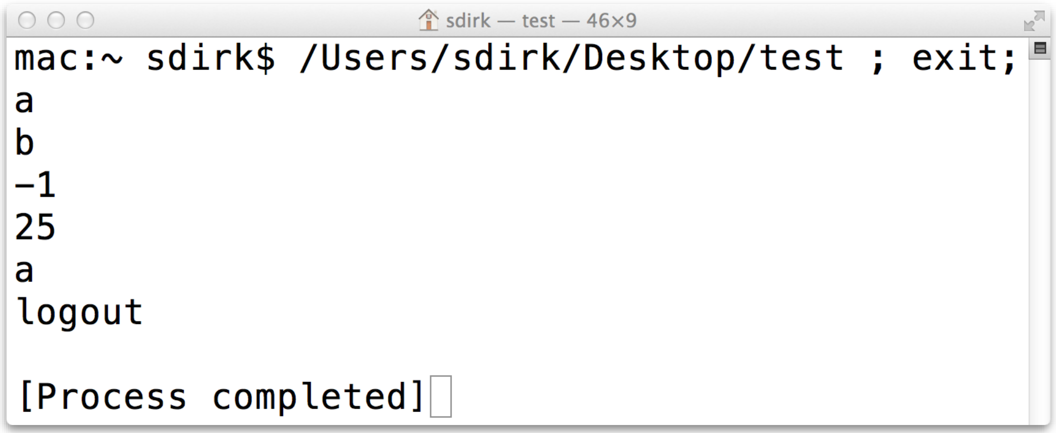
```
a
97
Press any key to continue...
```

Char

- Arithmetic operators operate on the ASCII values of the char



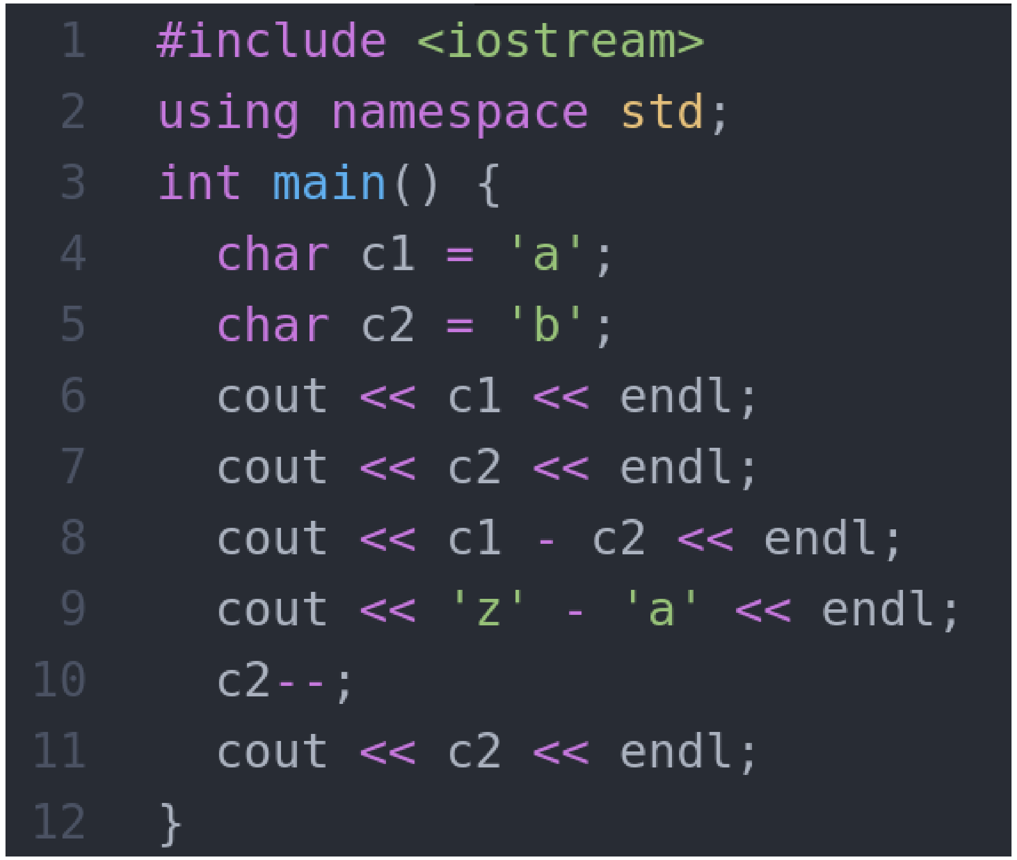
```cpp
#include<iostream>
using namespace std;
int main() {
    char c1 = 'a';
    char c2 = 'b';
    cout << c1 << endl;
    cout << c2 << endl;
    cout << c1 - c2 << endl;
    cout << 'z' - 'a' << endl;
    c2--;
    cout << c2 << endl;
}
```



```
mac:~ sdirk$ /Users/sdirk/Desktop/test ; exit;
a
b
-1
25
a
logout

[Process completed]
```

Char

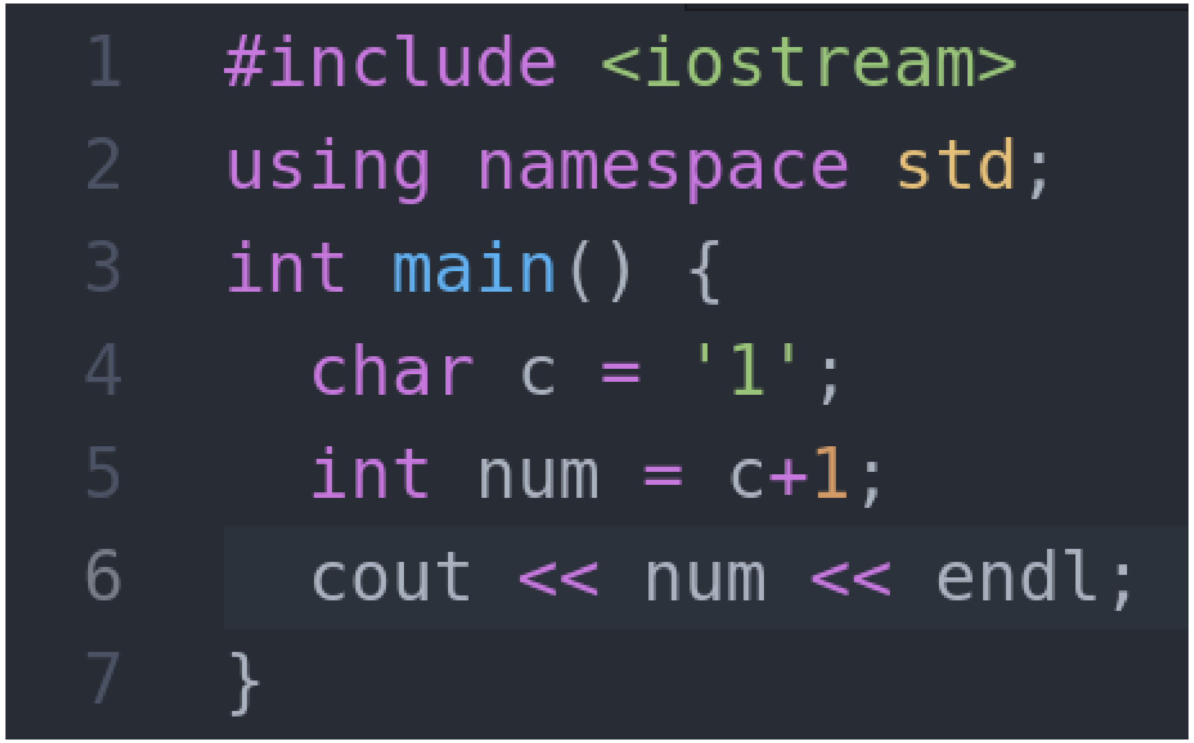
- Arithmetic operators operate on the ASCII values of the char



```cpp
#include <iostream>
using namespace std;
int main() {
  char c1 = 'a';
  char c2 = 'b';
  cout << c1 << endl;
  cout << c2 << endl;
  cout << c1 - c2 << endl;
  cout << 'z' - 'a' << endl;
  c2--;
  cout << c2 << endl;
}
```

Char

- Arithmetic operators operate on the ASCII values of the char



```cpp
#include <iostream>
using namespace std;
int main() {
  char c = '1';
  int num = c+1;
  cout << num << endl;
}
```

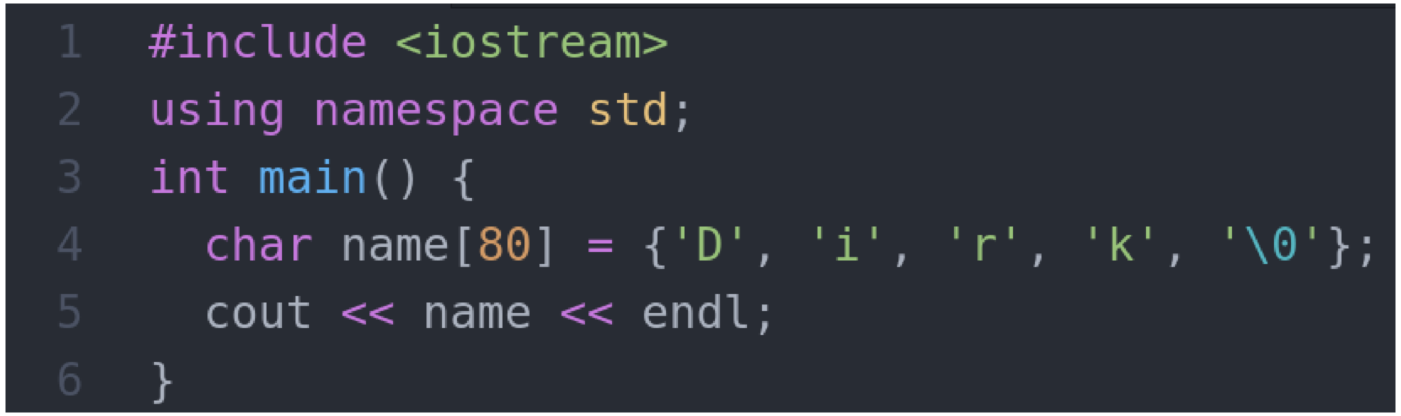
Outline

- Array Basics
- Arrays and Functions
- Searching and Sorting
- 2D Arrays
- Char Variables
- Char Arrays

Char Arrays

Char Array

- To represent a text in C++, one option is to use a char array
  - A char array can store any sequence shorter than its length
  - A special character called the null character, written as '\0', is used to signal the end of the sequence

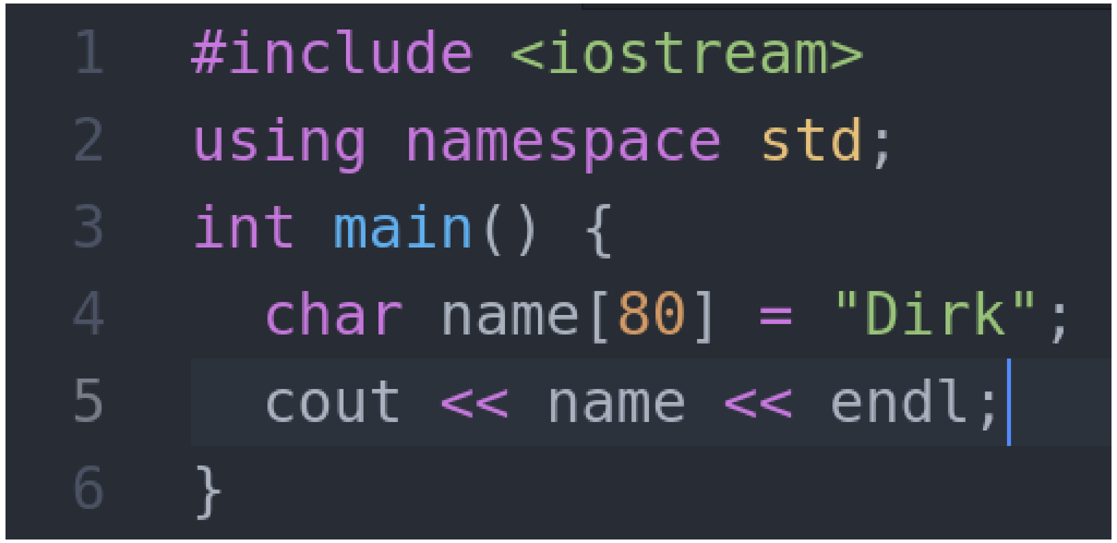


```cpp
#include <iostream>
using namespace std;
int main() {
  char name[80] = {'D', 'i', 'r', 'k', '\0'};
  cout << name << endl;
}
```

	0	1	2	3	4	…	79
name	D	i	r	k	\0	…	

Char Array

- Alternatively, double quotes can be used for the initialization



```cpp
#include <iostream>
using namespace std;
int main() {
  char name[80] = "Dirk";
  cout << name << endl;
}
```

	0	1	2	3	4	…	79
name	D	i	r	k	\0	…	

Task 1

- Write a function charToInt that will take a char integer and returns an int.
  - E.g., charToInt(‘9’) will return 9

Task 2

- Write a function toUpper that will take a lower case char and returns its upper case
  - E.g., toUpper(‘a’) will return ‘A’.

Task 3

- Write a function toUpper2 that will take a lower case char array and changes it upper case equivalent
- You may assume that the char array is filled with chars from ‘a’ to ‘z’