Michael Nowak

Texas A&M University

Notes

Overview

- Passing arrays to functions
- Creating a one-dimensional array on the free store
- Creating a two-dimensional array on the free store
- Creating arrays in functions
  - What's problematic about this?
  - How's this any different
- "Resizing" an array
- Shallow vs deep copy

Notes

Overview

- Passing arrays to functions
- Creating a one-dimensional array on the free store
- Creating a two-dimensional array on the free store
- Creating arrays in functions
  - What's problematic about this?
  - How's this any different
- "Resizing" an array
- Shallow vs deep copy

Notes

Passing arrays to functions

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6      double arr[] {1.0, -3.0};
7      int amaxidx = amax(arr, 2);
8      return 0;
9  }
10
11 int amax(const double *x, const unsigned int len)
12 {
13     int maxidx = 0;
14     for (unsigned int i = 0; i < len; ++i) {
15         if (fabs(x[i]) > fabs(x[maxidx]))
16             maxidx = i;
17     }
18     return maxidx;
19 }
```

Notes

Passing arrays to functions

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6      double arr[] {1.0, -3.0};
7      int amaxidx = amax(arr, 2);
8      return 0;
9  }
10
11 int amax(const double *x, const unsigned int len)
12 {
13     int maxidx = 0;
14     for (unsigned int i = 0; i < len; ++i) {
15         if (fabs(x[i]) > fabs(x[maxidx]))
16             maxidx = i;
17     }
18     return maxidx;
19 }
```

Notes

Passing arrays to functions

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6      double arr[] {1.0, -3.0};
7      int amaxidx = amax(arr, 2);
8      return 0;
9  }
10
11 int amax(const double *x, const unsigned int len)
12 {
13     int maxidx = 0;
14     for (unsigned int i = 0; i < len; ++i) {
15         if (fabs(x[i]) > fabs(x[maxidx]))
16             maxidx = i;
17     }
18     return maxidx;
19 }
```

Notes

Passing arrays to functions

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6      double arr[] {1.0, -3.0};
7      int amaxidx = amax(arr, 2);
8      return 0;
9  }
10
11 int amax(const double *x, const unsigned int len)
12 {
13     int maxidx = 0;
14     for (unsigned int i = 0; i < len; ++i) {
15         if (fabs(x[i]) > fabs(x[maxidx]))
16             maxidx = i;
17     }
18     return maxidx;
19 }
```

Notes

Passing arrays to functions

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6      double arr[] {1.0, -3.0};
7      int amaxidx = amax(arr, 2);
8      return 0;
9  }
10
11 int amax(const double *x, const unsigned int len)
12 {
13     int maxidx = 0;
14     for (unsigned int i = 0; i < len; ++i) {
15         if (fabs(x[i]) > fabs(x[maxidx]))
16             maxidx = i;
17     }
18     return maxidx;
19 }
```

Notes

Passing arrays to functions

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6      double arr[] {1.0, -3.0};
7      int amaxidx = amax(arr, 2);
8      return 0;
9  }
10
11 int amax(const double *x, const unsigned int len)
12 {
13     int maxidx = 0;
14     for (unsigned int i = 0; i < len; ++i) {
15         if (fabs(x[i]) > fabs(x[maxidx]))
16             maxidx = i;
17     }
18     return maxidx;
19 }
```

Notes

Passing arrays to functions

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6      double arr[] {1.0, -3.0};
7      int amaxidx = amax(arr, 2);
8      return 0;
9  }
10
11 int amax(const double *x, const unsigned int len)
12 {
13     int maxidx = 0;
14     for (unsigned int i = 0; i < len; ++i) {
15         if (fabs(x[i]) > fabs(x[maxidx]))
16             maxidx = i;
17     }
18     return maxidx;
19 }
```

Notes

Passing arrays to functions

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6      double arr[] {1.0, -3.0};
7      int amaxidx = amax(arr, 2);
8      return 0;
9  }
10
11 int amax(const double *x, const unsigned int len)
12 {
13     int maxidx = 0;
14     for (unsigned int i = 0; i < len; ++i) {
15         if (fabs(x[i]) > fabs(x[maxidx]))
16             maxidx = i;
17     }
18     return maxidx;
19 }
```

Notes

Passing arrays to functions

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6      double arr[] {1.0, -3.0};
7      int amaxidx = amax(arr, 2);
8      return 0;
9  }
10
11 int amax(const double *x, const unsigned int len)
12 {
13     int maxidx = 0;
14     for (unsigned int i = 0; i < len; ++i) {
15         if (fabs(x[i]) > fabs(x[maxidx]))
16             maxidx = i;
17     }
18     return maxidx;
19 }
```

Notes

Passing arrays to functions

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6      double arr[] {1.0, -3.0};
7      int amaxidx = amax(arr, 2);
8      return 0;
9  }
10
11 int amax(const double *x, const unsigned int len)
12 {
13     int maxidx = 0;
14     for (unsigned int i = 0; i < len; ++i) {
15         if (fabs(x[i]) > fabs(x[maxidx]))
16             maxidx = i;
17     }
18     return maxidx;
19 }
```

Notes

Passing arrays to functions

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6      double arr[] {1.0, -3.0};
7      int amaxidx = amax(arr, 2);
8      return 0;
9  }
10
11 int amax(const double *x, const unsigned int len)
12 {
13     int maxidx = 0;
14     for (unsigned int i = 0; i < len; ++i) {
15         if (fabs(x[i]) > fabs(x[maxidx]))
16             maxidx = i;
17     }
18     return maxidx;
19 }
```

Notes

Overview

Passing arrays to functions

Creating a one-dimensional array on the free store

Creating a two-dimensional array on the free store

Creating arrays in functions
- What's problematic about this?
- How's this any different

"Resizing" an array

Shallow vs deep copy

Notes

Creating a one-dimensional array on the free store

```
1  int main()
2  {
3      unsigned int arr_sz = 2;
4      int *arr = new int[arr_sz];
5      for (unsigned int i = 0; i < arr_sz; ++i)
6          arr[i] = i;
7      /* does something interesting */
8      delete [] arr;
9
10     return 0;
11 }
```

Notes

Creating a one-dimensional array on the free store

```
1  int main()
2  {
3      unsigned int arr_sz = 2;
4      int *arr = new int[arr_sz];
5      for (unsigned int i = 0; i < arr_sz; ++i)
6          arr[i] = i;
7      /* does something interesting */
8      delete [] arr;
9
10     return 0;
11 }
```

Notes

Creating a one-dimensional array on the free store

```
1  int main()
2  {
3      unsigned int arr_sz = 2;
4      int *arr = new int[arr_sz];
5      for (unsigned int i = 0; i < arr_sz; ++i)
6          arr[i] = i;
7      /* does something interesting */
8      delete [] arr;
9
10     return 0;
11 }
```

Notes

Creating a one-dimensional array on the free store

```
1  int main()
2  {
3      unsigned int arr_sz = 2;
4      int *arr = new int[arr_sz];
5      for (unsigned int i = 0; i < arr_sz; ++i)
6          arr[i] = i;
7      /* does something interesting */
8      delete [] arr;
9
10     return 0;
11 }
```

Notes

Creating a one-dimensional array on the free store

```
1  int main()
2  {
3      unsigned int arr_sz = 2;
4      int *arr = new int[arr_sz];
5      for (unsigned int i = 0; i < arr_sz; ++i)
6          arr[i] = i;
7      /* does something interesting */
8      delete [] arr;
9
10     return 0;
11 }
```

Notes

Creating a one-dimensional array on the free store

```
1  int main()
2  {
3      unsigned int arr_sz = 2;
4      int *arr = new int[arr_sz];
5      for (unsigned int i = 0; i < arr_sz; ++i)
6          arr[i] = i;
7      /* does something interesting */
8      delete [] arr;
9
10     return 0;
11 }
```

Notes

Creating a one-dimensional array on the free store

```
1  int main()
2  {
3      unsigned int arr_sz = 2;
4      int *arr = new int[arr_sz];
5      for (unsigned int i = 0; i < arr_sz; ++i)
6          arr[i] = i;
7      /* does something interesting */
8      delete [] arr;
9
10     return 0;
11 }
```

Notes

Creating a one-dimensional array on the free store

```
1  int main()
2  {
3      unsigned int arr_sz = 2;
4      int *arr = new int[arr_sz];
5      for (unsigned int i = 0; i < arr_sz; ++i)
6          arr[i] = i;
7      /* does something interesting */
8      delete [] arr;
9
10     return 0;
11 }
```

Notes

Creating a one-dimensional array on the free store

```
1  int main()
2  {
3      unsigned int arr_sz = 2;
4      int *arr = new int[arr_sz];
5      for (unsigned int i = 0; i < arr_sz; ++i)
6          arr[i] = i;
7      /* does something interesting */
8      delete [] arr;
9
10     return 0;
11 }
```

Notes

Creating a one-dimensional array on the free store

```
1  int main()
2  {
3      unsigned int arr_sz = 2;
4      int *arr = new int[arr_sz];
5      for (unsigned int i = 0; i < arr_sz; ++i)
6          arr[i] = i;
7      /* does something interesting */
8      delete [] arr;
9
10     return 0;
11 }
```

Notes

Overview

Passing arrays to functions

Creating a one-dimensional array on the free store

Creating a two-dimensional array on the free store

Creating arrays in functions
- What's problematic about this?
- How's this any different

"Resizing" an array

Shallow vs deep copy

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7      int *(*arr) = new int*[NOROWS];
8      for (unsigned int i = 0; i < NOROWS; ++i)
9         arr[i] = new int[NOCOLS];
10
11     for (unsigned int i = 0; i < NOROWS; ++i)
12        delete [] arr[i];
13     delete [] arr;
14
15     return 0;
16 }
```

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7     int *(*arr) = new int*[NOROWS];
8     for (unsigned int i = 0; i < NOROWS; ++i)
9        arr[i] = new int[NOCOLS];
10
11    for (unsigned int i = 0; i < NOROWS; ++i)
12       delete [] arr[i];
13    delete [] arr;
14
15    return 0;
16 }
```

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7     int *(*arr) = new int*[NOROWS];
8     for (unsigned int i = 0; i < NOROWS; ++i)
9        arr[i] = new int[NOCOLS];
10
11    for (unsigned int i = 0; i < NOROWS; ++i)
12       delete [] arr[i];
13    delete [] arr;
14
15    return 0;
16 }
```

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7     int *(*arr) = new int*[NOROWS];
8     for (unsigned int i = 0; i < NOROWS; ++i)
9        arr[i] = new int[NOCOLS];
10
11    for (unsigned int i = 0; i < NOROWS; ++i)
12       delete [] arr[i];
13    delete [] arr;
14
15    return 0;
16 }
```

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7     int *(*arr) = new int*[NOROWS];
8     for (unsigned int i = 0; i < NOROWS; ++i)
9        arr[i] = new int[NOCOLS];
10
11    for (unsigned int i = 0; i < NOROWS; ++i)
12       delete [] arr[i];
13    delete [] arr;
14
15    return 0;
16 }
```

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7     int *(*arr) = new int*[NOROWS];
8     for (unsigned int i = 0; i < NOROWS; ++i)
9        arr[i] = new int[NOCOLS];
10
11    for (unsigned int i = 0; i < NOROWS; ++i)
12       delete [] arr[i];
13    delete [] arr;
14
15    return 0;
16 }
```

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7     int *(*arr) = new int*[NOROWS];
8     for (unsigned int i = 0; i < NOROWS; ++i)
9        arr[i] = new int[NOCOLS];
10
11    for (unsigned int i = 0; i < NOROWS; ++i)
12       delete [] arr[i];
13    delete [] arr;
14
15    return 0;
16 }
```

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7     int *(*arr) = new int*[NOROWS];
8     for (unsigned int i = 0; i < NOROWS; ++i)
9        arr[i] = new int[NOCOLS];
10
11    for (unsigned int i = 0; i < NOROWS; ++i)
12       delete [] arr[i];
13    delete [] arr;
14
15    return 0;
16 }
```

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7     int *(*arr) = new int*[NOROWS];
8     for (unsigned int i = 0; i < NOROWS; ++i)
9        arr[i] = new int[NOCOLS];
10
11    for (unsigned int i = 0; i < NOROWS; ++i)
12       delete [] arr[i];
13    delete [] arr;
14
15    return 0;
16 }
```

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7     int *(*arr) = new int*[NOROWS];
8     for (unsigned int i = 0; i < NOROWS; ++i)
9        arr[i] = new int[NOCOLS];
10
11    for (unsigned int i = 0; i < NOROWS; ++i)
12       delete [] arr[i];
13    delete [] arr;
14
15    return 0;
16 }
```

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7     int *(*arr) = new int*[NOROWS];
8     for (unsigned int i = 0; i < NOROWS; ++i)
9        arr[i] = new int[NOCOLS];
10
11    for (unsigned int i = 0; i < NOROWS; ++i)
12       delete [] arr[i];
13    delete [] arr;
14
15    return 0;
16 }
```

Notes

Creating a two-dimensional array on the free store

```
1  const unsigned int NOROWS = 2;
2  const unsigned int NOCOLS = 3;
3
4  int main()
5  {
6
7     int *(*arr) = new int*[NOROWS];
8     for (unsigned int i = 0; i < NOROWS; ++i)
9        arr[i] = new int[NOCOLS];
10
11    for (unsigned int i = 0; i < NOROWS; ++i)
12       delete [] arr[i];
13    delete [] arr;
14
15    return 0;
16 }
```

Notes

Overview

Passing arrays to functions

Creating a one-dimensional array on the free store

Creating a two-dimensional array on the free store

Creating arrays in functions
- What's problematic about this?
- How's this any different

"Resizing" an array

Shallow vs deep copy

Notes

Overview

Passing arrays to functions

Creating a one-dimensional array on the free store

Creating a two-dimensional array on the free store

Creating arrays in functions
- What's problematic about this?
- How's this any different

"Resizing" an array

Shallow vs deep copy

Notes

What's problematic about it?

```
1  char * problematic();
2
3  int main()
4  {
5      char *str = problematic();
6      /* tries to do something interesting with str */
7      return 0;
8  }
9
10 char * problematic()
11 {
12     char localStr[] = "Hello!";
13     return localStr;
14 }
```

Notes

What's problematic about it?

```
1  char * problematic();
2
3  int main()
4  {
5      char *str = problematic();
6      /* tries to do something interesting with str */
7      return 0;
8  }
9
10 char * problematic()
11 {
12     char localStr[] = "Hello!";
13     return localStr;
14 }
```

Notes

What's problematic about it?

```
1  char * problematic();
2
3  int main()
4  {
5      char *str = problematic();
6      /* tries to do something interesting with str */
7      return 0;
8  }
9
10 char * problematic()
11 {
12     char localStr[] = "Hello!";
13     return localStr;
14 }
```

Notes

What's problematic about it?

```
1  char * problematic();
2
3  int main()
4  {
5      char *str = problematic();
6      /* tries to do something interesting with str */
7      return 0;
8  }
9
10 char * problematic()
11 {
12     char localStr[] = "Hello!";
13     return localStr;
14 }
```

Notes

What's problematic about it?

```
1  char * problematic();
2
3  int main()
4  {
5      char *str = problematic();
6      /* tries to do something interesting with str */
7      return 0;
8  }
9
10 char * problematic()
11 {
12     char localStr[] = "Hello!";
13     return localStr;
14 }
```

Notes

What's problematic about it?

```
1  char * problematic();
2
3  int main()
4  {
5      char *str = problematic();
6      /* tries to do something interesting with str */
7      return 0;
8  }
9
10 char * problematic()
11 {
12     char localStr[] = "Hello!";
13     return localStr;
14 }
```

Notes

Overview

Passing arrays to functions

Creating a one-dimensional array on the free store

Creating a two-dimensional array on the free store

Creating arrays in functions

- What's problematic about this?
- How's this any different

"Resizing" an array

Shallow vs deep copy

Notes

How's this any different?

```
1  char * notProblematic();
2
3  int main()
4  {
5      char *str = notProblematic();
6      /* does something interesting with str */
7      return 0;
8  }
9
10 char * notProblematic()
11 {
12     char *localStr = new char[7] {'H', 'e', 'l', 'l', 'o', '
           !', '\0'};
13     return localStr;
14 }
```

Notes

How's this any different?

```
1  char * notProblematic();
2
3  int main()
4  {
5      char *str = notProblematic();
6      /* does something interesting with str */
7      return 0;
8  }
9
10 char * notProblematic()
11 {
12     char *localStr = new char[7] {'H', 'e', 'l', 'l', 'o', '
           !', '\0'};
13     return localStr;
14 }
```

Notes

How's this any different?

```
1  char * notProblematic();
2
3  int main()
4  {
5      char *str = notProblematic();
6      /* does something interesting with str */
7      return 0;
8  }
9
10 char * notProblematic()
11 {
12     char *localStr = new char[7] {'H', 'e', 'l', 'l', 'o', '
           !', '\0'};
13     return localStr;
14 }
```

Notes

How's this any different?

```
1  char * notProblematic();
2
3  int main()
4  {
5      char *str = notProblematic();
6      /* does something interesting with str */
7      return 0;
8  }
9
10 char * notProblematic()
11 {
12     char *localStr = new char[7] {'H', 'e', 'l', 'l', 'o', '
           !', '\0'};
13     return localStr;
14 }
```

Notes

How's this any different?

```
1  char * notProblematic();
2
3  int main()
4  {
5      char *str = notProblematic();
6      /* does something interesting with str */
7      return 0;
8  }
9
10 char * notProblematic()
11 {
12     char *localStr = new char[7] {'H', 'e', 'l', 'l', 'o', '
           !', '\0'};
13     return localStr;
14 }
```

Notes

How's this any different?

```
1  char * notProblematic();
2
3  int main()
4  {
5      char *str = notProblematic();
6      /* does something interesting with str */
7      return 0;
8  }
9
10 char * notProblematic()
11 {
12     char *localStr = new char[7] {'H', 'e', 'l', 'l', 'o', '
           !', '\0'};
13     return localStr;
14 }
```

Notes

Overview

Passing arrays to functions

Creating a one-dimensional array on the free store

Creating a two-dimensional array on the free store

Creating arrays in functions
- What's problematic about this?
- How's this any different

"Resizing" an array

Shallow vs deep copy

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

"Resizing" an array

```
1  void resize(int *&, unsigned int &);
2
3  int main()
4  {
5      unsigned int cap = 1;
6      unsigned int sz = 0;
7      int *arr = new int[cap];
8      for (unsigned int i = 0; i < 2; ++i) {
9          if (cap == sz)
10             resize(arr, cap);
11         arr[sz] = i; sz += 1;
12     }
13     return 0;
14 }
15
16 void resize(int *&array, unsigned int &capacity)
17 {
18     unsigned int newCapacity = capacity * 2;
19     int *temp = new int[capacity * 2];
20     for (unsigned int i = 0; i < capacity; ++i)
21         temp[i] = array[i];
22     delete [] array;
23     capacity = newCapacity;
24     array = temp;
25 }
```

Notes

Overview

Passing arrays to functions

Creating a one-dimensional array on the free store

Creating a two-dimensional array on the free store

Creating arrays in functions
- What's problematic about this?
- How's this any different

"Resizing" an array

Shallow vs deep copy

Notes

Shallow vs deep copy

- `Shallow copy`: The value stored in the pointer will be copied, but the memory it points to will not be duplicated
  - Result: two pointers pointing to the same object
- `Deep copy`: Makes copy of the dynamically allocated object pointed to and stores address in a pointer
  - Result: two pointers pointing to the different objects with the same values

Notes

Notes