	Notes
Michael Nowak	
Texas A&M University	
Overview	Notes
Passing arrays to functions	
Creating a one-dimensional array on the free store	
Creating a two-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	
Overview	
	Notes
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	

```
1  #include <cmath>
2  int amax(const double *, const unsigned int);
3
4  int main()
5  {
6    double arr[] {i.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

# Notes

### Passing arrays to functions

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

Notes			

## 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	#in	clude <cmath></cmath>
2		
3	int	amax(const double *, const unsigned int);
1 '		
4		main()
5	{	
6		double arr[] {1.0, -3.0};
7		<pre>int amaxidx = amax(arr, 2);</pre>
8		return 0;
9	}	
10		
11	int	amax(const double *x, const unsigned int len)
12	{	aman (solids acades in, solids and globa ins ion)
13	·	int maxidx = 0:
14		
		for (unsigned int i = 0; i < len; ++i) {
15		<pre>if (fabs(x[i]) &gt; fabs(x[maxidx]))</pre>
16		maxidx = i;
17		}
18		return maxidx;
19	}	

Notes			

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

Notes			

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			

```
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 }</pre>
```

Creating a one-dimensional array on the free store

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 }</pre>
```

Notes			

Notes

```
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 }</pre>
```

Creating a one-dimensional array on the free store

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 }</pre>
```

Notes		

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            return 0;
11        }</pre>
```

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    return 0;
11 }</pre>
```

Notes		
_		
Notes		

Notes \_\_\_\_\_\_

```
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 }</pre>
```

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	co	nst unsigned int NOROWS = 2;
2	CO	nst unsigned int NOCOLS = 3;
3		
4	in	t main()
5	{	
6		
7		<pre>int *(*arr) = new int*[NOROWS];</pre>
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		<pre>delete [] arr[i];</pre>
13		delete [] arr;
14		
15		return 0;
16	}	
I		

_			
_			
_			
_			
_			

```
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 }</pre>
```

### 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 }</pre>
```

### 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 }</pre>
```

Notes

Notes

```
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 }</pre>
```

### 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 }</pre>
```

### 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 }</pre>
```

### Notes

Notes

```
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 }</pre>
```

### 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 }</pre>
```

### 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 }</pre>
```

### Notes

Notes

-	·

```
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 }</pre>
```

### 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 }</pre>
```

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	Notes
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	
What's problematic about it?	Notes
<pre>1 char * problematic(); 2 3 int main()</pre>	
4 { 5     char *str = problematic(); 6     /* tries to do something interesting with str */	
7 return 0; 8 } 9	
10 char * problematic() 11 { 12	
13 return localStr; 14 }	
Milester III. et al. essa	
What's problematic about it?	Notes
<pre>1 char * problematic(); 2</pre>	
<pre>3 int main() 4 { 5     char *str = problematic();</pre>	
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 }	

### 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 }
```

### 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 }
```

### What's problematic about it?

Notes

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

-			
-			
-			
-			
_			

### How's this any different?

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

### How's this any different?

```
char *str = notProblematic();
    /* does something interesting with str */
return 0;
 13
14 }
```

### How's this any different?

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


Notes

### Notes

### How's this any different?

### Notes

### How's this any different?

### 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

Notes			

1 2	<pre>void resize(int *&amp;, unsigned int &amp;);</pre>
3	<pre>int main()</pre>
4	{
5	unsigned int cap = 1;
6	unsigned int sz = 0;
7	<pre>int *arr = new int[cap];</pre>
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	<pre>void resize(int *&amp;array, unsigned int &amp;capacity)</pre>
17	{
18	unsigned int newCapacity = capacity * 2;
19	<pre>int *temp = new int[capacity * 2];</pre>
20	for (unsigned int i = 0; i < capacity; ++i)
21	<pre>temp[i] = array[i];</pre>
22	delete [] array;
23	<pre>capacity = newCapacity;</pre>
24	array = temp;
25	}

Notes			

```
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 }</pre>
```

Notes

### "Resizing" an array

Notes			

1	<pre>void resize(int *&amp;, unsigned int &amp;);</pre>
3	<pre>int main()</pre>
4	{
5	unsigned int cap = 1;
6	unsigned int sz = 0;
7	<pre>int *arr = new int[cap];</pre>
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	<pre>int *temp = new int[capacity * 2];</pre>
20	for (unsigned int i = 0; i < capacity; ++i)
21	<pre>temp[i] = array[i];</pre>
22	delete [] array;
23	capacity = newCapacity;
24	array = temp;
25	}

Notes			

# Notes

### "Resizing" an array

Notes			

	9 ,
1	<pre>void resize(int *&amp;, unsigned int &amp;);</pre>
2	
3	int main()
4	{
5	unsigned int cap = 1;
6	unsigned int sz = 0;
7	<pre>int *arr = new int[cap];</pre>
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	<pre>void resize(int *&amp;array, unsigned int &amp;capacity)</pre>
17	{
18	unsigned int newCapacity = capacity * 2;
19	<pre>int *temp = new int[capacity * 2];</pre>
20	for (unsigned int i = 0; i < capacity; ++i)
21	<pre>temp[i] = array[i];</pre>
22	delete [] array;
23	capacity = newCapacity;
24	array = temp;
25	}
1	

Notes			

```
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     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    }</pre>
```

### "Resizing" an array

Notes			

Notes			

Notes		

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

Notes

### "Resizing" an array

```
unsigned int cap = 1;
unsigned int sz = 0;
int *arr = new int[cap];
for (unsigned int i = 0; i < 2; ++i) {
   if (cap == sz)
       resize(arr, cap);
   arr[sz] = i; sz += 1;
          void resize(int *&array, unsigned int &capacity)
{
                    unsigned int newCapacity = capacity * 2;
int *temp = new int[capacity * 2];
for (unsigned int i = 0; i < capacity; ++i)
    temp[i] = array[i];
delete [i] array;
capacity = newCapacity;
array = temp;</pre>
```

Notes			

1	<pre>void resize(int *&amp;, unsigned int &amp;);</pre>
2	
4	int main() {
5	unsigned int cap = 1;
6	unsigned int sz = 0;
7	<pre>int *arr = new int[cap];</pre>
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 14	return 0;
15	1
16	void resize(int *&array, unsigned int &capacity)
17	{
18	unsigned int newCapacity = capacity * 2;
19	<pre>int *temp = new int[capacity * 2];</pre>
20	for (unsigned int i = 0; i < capacity; ++i)
21	temp[i] = array[i];
22	delete [] array;
23 24	capacity = newCapacity;
25	array = temp;
	ı

Notes			

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

Notes

### "Resizing" an array

```
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; +
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 17 {
18 unsigned int newCapacity = capacity int *temp = new int[capacity * 2]
20 for (unsigned int i = 0; i < capacity = temp[i] = array[i];
21 delete [] array;
22 capacity = newCapacity;
23 array = temp;
25 }
                                             unsigned int cap = 1;
unsigned int sz = 0;
int *arr = new int[cap];
for (unsigned int i = 0; i < 2; ++i) {
   if (cap == sz)
       resize(arr, cap);
   arr[sz] = i; sz += 1;
                        void resize(int *&array, unsigned int &capacity)
{
                                              unsigned int newCapacity = capacity * 2;
int *temp = new int[capacity * 2];
for (unsigned int i = 0; i < capacity; ++i)
    temp[i] = array[i];
delete [i] array;
capacity = newCapacity;
array = temp;</pre>
```

Notes			

1 2	<pre>void resize(int *&amp;, unsigned int &amp;);</pre>
3	<pre>int main()</pre>
4	{
5	unsigned int cap = 1;
6	unsigned int sz = 0;
7	<pre>int *arr = new int[cap];</pre>
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	<pre>int *temp = new int[capacity * 2];</pre>
20	for (unsigned int i = 0; i < capacity; ++i)
21	<pre>temp[i] = array[i];</pre>
22	delete [] array;
23	capacity = newCapacity;
24	array = temp;
25	}

Notes			

# Notes \_\_\_\_\_\_

### "Resizing" an array

Notes			

void resize(int *&, unsigned int &);
int main()
{
unsigned int cap = 1;
unsigned int sz = 0;
<pre>int *arr = new int[cap];</pre>
for (unsigned int i = 0; i < 2; ++i) {
if (cap == sz)
resize(arr, cap);
arr[sz] = i; sz += 1;
}
return 0;
}
void resize(int *&array, unsigned int &capacity)
{
unsigned int newCapacity = capacity * 2;
<pre>int *temp = new int[capacity * 2];</pre>
for (unsigned int i = 0; i < capacity; ++i)
temp[i] = array[i];
delete [] array;
capacity = newCapacity;
array = temp;

Notes			

Overview	Notes
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	
Shallow vs deep copy	Notes
<ul> <li>▶ Shallow copy: The value stored in the pointer will be copied, but the memory it points to will not be duplicated</li> <li>▶ Result: two pointers pointing to the same object</li> </ul>	
<ul> <li>Deep copy: Makes copy of the dynamically allocated object pointed to and stores address in a pointer</li> <li>Result: two pointers pointing to the different objects with the same values</li> </ul>	
Same values	
	Notes