ESc 101: Fundamentals of Computing

Lecture 20

Feb 17, 2010

Outline

1 Pointers

Resolving the Anomaly for Arrays

- Declaration
  `int z[3]`
  reserves 3 memory locations (each of 4 bytes).
- These are named `z[0]` to `z[2]`.
- In addition to this, another memory location is reserved!
- The name of this location is `z`.
- It stores the pointer to `z[0]`.
- This is why passing name of array as parameter allows us to change its content inside a function.

Resolving the Anomaly for Arrays

- Declaration
  `int z[3]`
  reserves 3 memory locations (each of 4 bytes).
- These are named `z[0]` to `z[2]`.
- In addition to this, another memory location is reserved!
- The name of this location is z.
- It stores the pointer to `z[0]`.
- This is why passing name of array as parameter allows us to change its content inside a function.

Resolving the Anomaly for Arrays

- Declaration
  `int z[3]`
  reserves 3 memory locations (each of 4 bytes).
- These are named `z[0]` to `z[2]`.
- In addition to this, another memory location is reserved!
- The name of this location is z.
- It stores the pointer to `z[0]`.
- This is why passing name of array as parameter allows us to change its content inside a function.

Resolving the Anomaly for Arrays

- Declaration
  `int z[3]`
  reserves 3 memory locations (each of 4 bytes).
- These are named `z[0]` to `z[2]`.
- In addition to this, another memory location is reserved!
- The name of this location is `z`.
- It stores the pointer to `z[0]`.
- This is why passing name of array as parameter allows us to change its content inside a function.

Resolving the Anomaly for Arrays

- Declaration

  `int z[3]`

  reserves 3 memory locations (each of 4 bytes).
- These are named `z[0]` to `z[2]`.
- In addition to this, another memory location is reserved!
- The name of this location is `z`.
- It stores the pointer to `z[0]`.
- This is why passing name of array as parameter allows us to change its content inside a function.

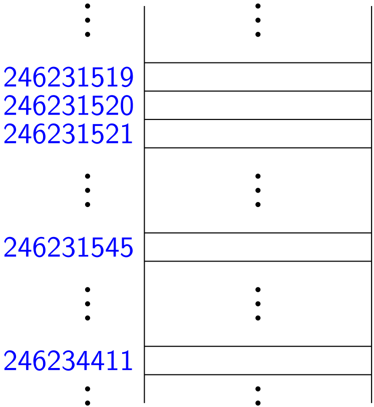
Resolving the Anomaly for Arrays

- Declaration

  `int z[3]`

  reserves 3 memory locations (each of 4 bytes).
- These are named `z[0]` to `z[2]`.
- In addition to this, another memory location is reserved!
- The name of this location is `z`.
- It stores the pointer to `z[0]`.
- This is why passing name of array as parameter allows us to change its content inside a function.

Example



246231519

246231520

246231521

246231545

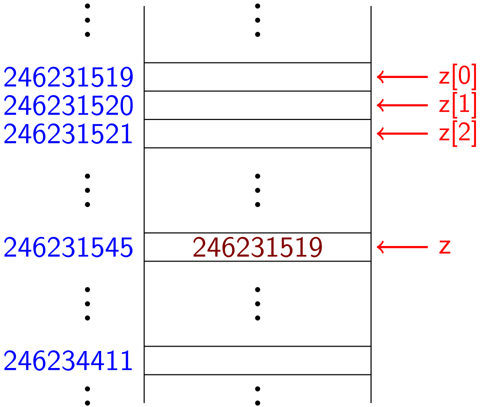
246234411

MEMORY

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

PROGRAM

Example



246231519 ← z[0]

246231520 ← z[1]

246231521 ← z[2]

246231545 | 246231519 ← z

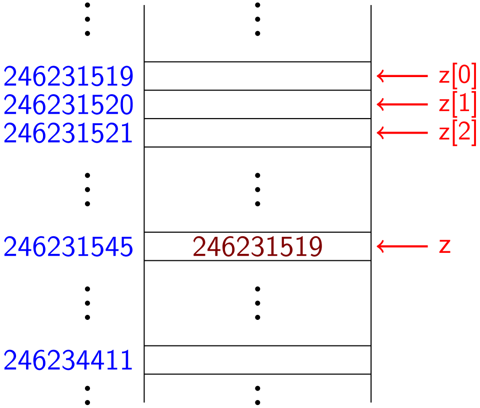
246234411

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example

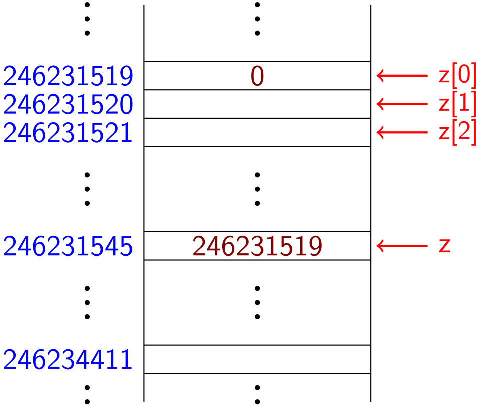


```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example



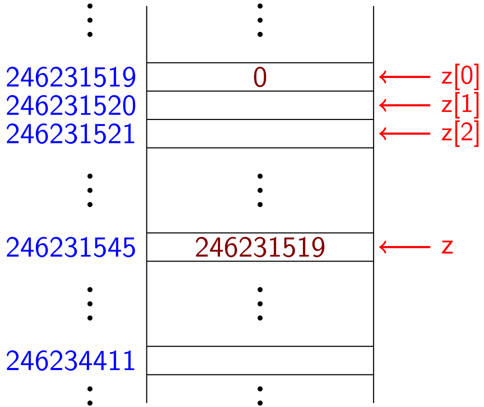
⋮	⋮	
246231519	0	⟵ z[0]
246231520		⟵ z[1]
246231521		⟵ z[2]
⋮	⋮	
246231545	246231519	⟵ z
⋮	⋮	
246234411		
⋮	⋮	

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example



246231519 | 0 ⟵ z[0]

246231520 | ⟵ z[1]

246231521 | ⟵ z[2]

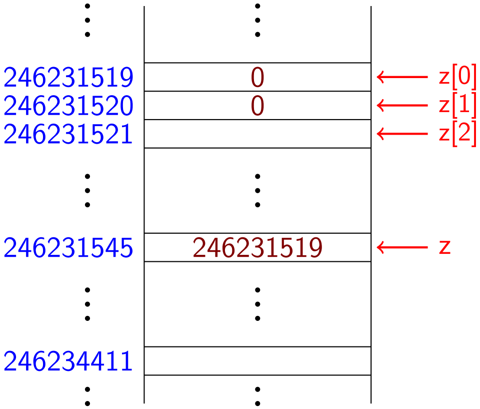
246231545 | 246231519 ⟵ z

246234411 |

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY **PROGRAM**

Example



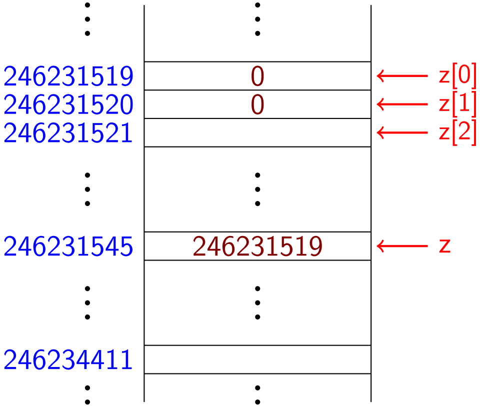
⋮	⋮	
246231519	0	⟵ z[0]
246231520	0	⟵ z[1]
246231521		⟵ z[2]
⋮	⋮	
246231545	246231519	⟵ z
⋮	⋮	
246234411		
⋮	⋮	

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example



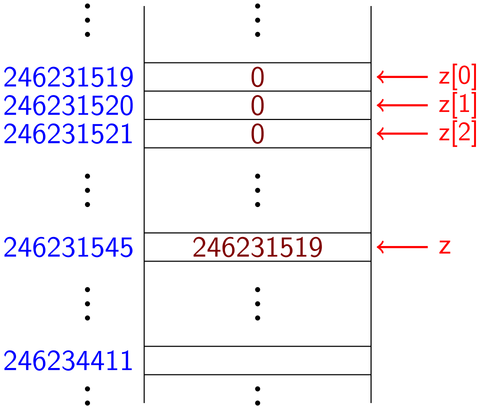
⋮	⋮	
246231519	0	← z[0]
246231520	0	← z[1]
246231521		← z[2]
⋮	⋮	
246231545	246231519	← z
⋮	⋮	
246234411		
⋮	⋮	

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example



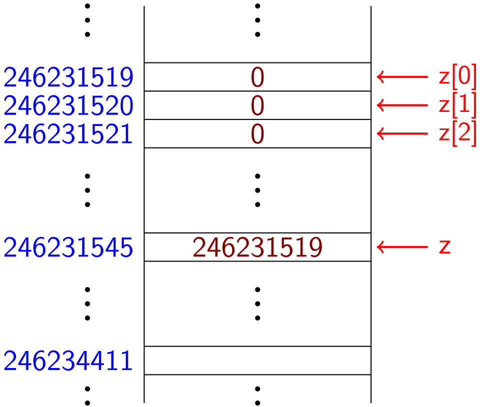
Address	Value	
⋮	⋮	
246231519	0	⟵ z[0]
246231520	0	⟵ z[1]
246231521	0	⟵ z[2]
⋮	⋮	
246231545	246231519	⟵ z
⋮	⋮	
246234411		
⋮	⋮	

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example

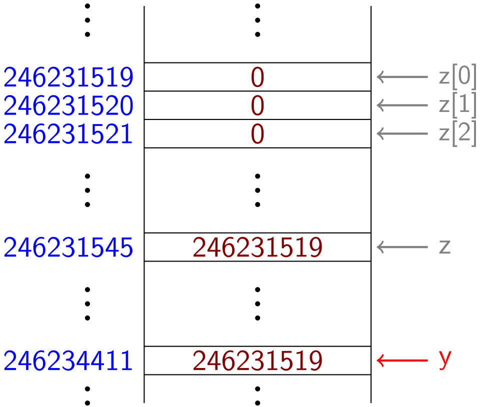


```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example



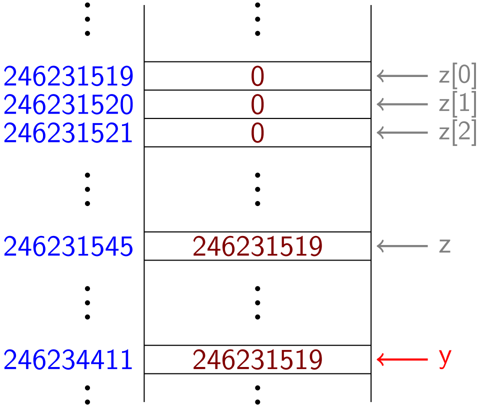
⋮	⋮	
246231519	0	⟵ z[0]
246231520	0	⟵ z[1]
246231521	0	⟵ z[2]
⋮	⋮	
246231545	246231519	⟵ z
⋮	⋮	
246234411	246231519	⟵ y
⋮	⋮	

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example



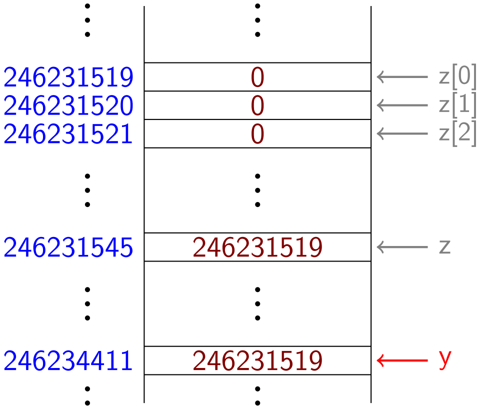
⋮	⋮	
246231519	0	⟵ z[0]
246231520	0	⟵ z[1]
246231521	0	⟵ z[2]
⋮	⋮	
246231545	246231519	⟵ z
⋮	⋮	
246234411	246231519	⟵ y
⋮	⋮	

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example



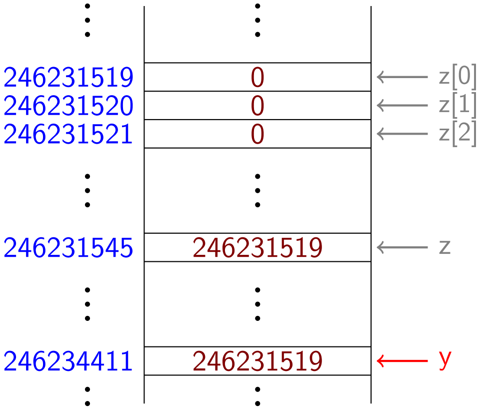
246231519 0 ← z[0]
246231520 0 ← z[1]
246231521 0 ← z[2]
246231545 246231519 ← z
246234411 246231519 ← y

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

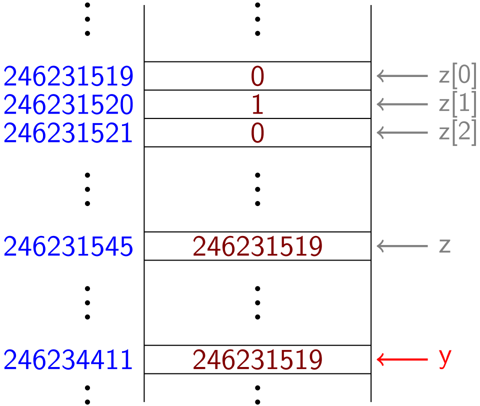
Example



```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY **PROGRAM**

Example



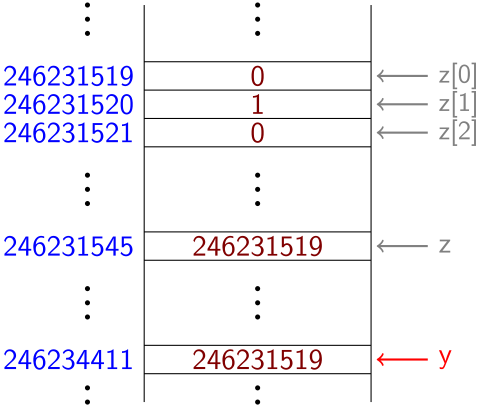
⋮	⋮	
246231519	0	← z[0]
246231520	1	← z[1]
246231521	0	← z[2]
⋮	⋮	
246231545	246231519	← z
⋮	⋮	
246234411	246231519	← y
⋮	⋮	

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example



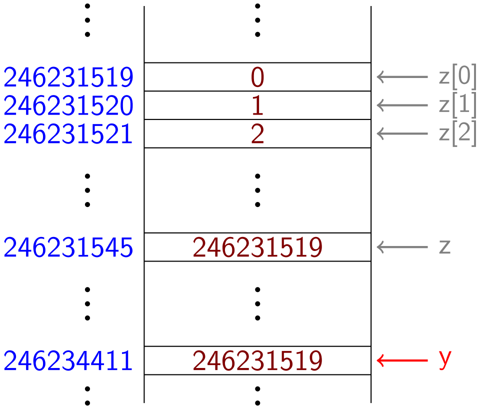
⋮	⋮	
246231519	0	← z[0]
246231520	1	← z[1]
246231521	0	← z[2]
⋮	⋮	
246231545	246231519	← z
⋮	⋮	
246234411	246231519	← y
⋮	⋮	

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example



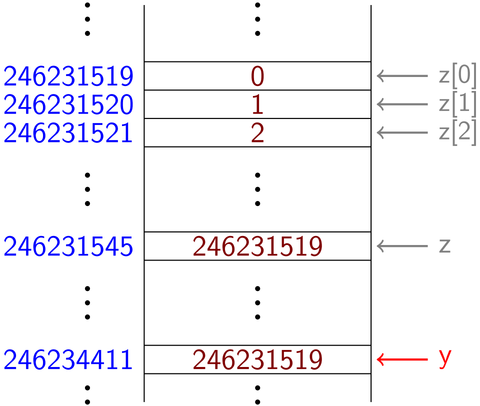
⋮	⋮	
246231519	0	← z[0]
246231520	1	← z[1]
246231521	2	← z[2]
⋮	⋮	
246231545	246231519	← z
⋮	⋮	
246234411	246231519	← y
⋮	⋮	

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example



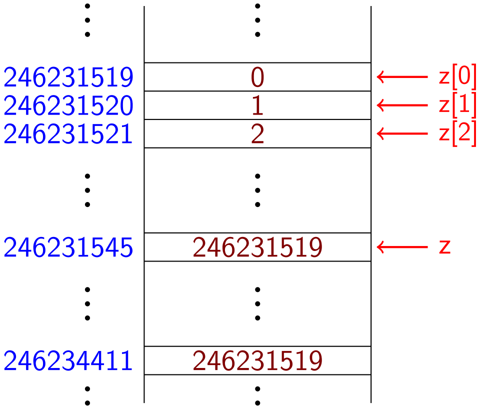
⋮	⋮	
246231519	0	← z[0]
246231520	1	← z[1]
246231521	2	← z[2]
⋮	⋮	
246231545	246231519	← z
⋮	⋮	
246234411	246231519	← y
⋮	⋮	

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Example



⋮	⋮	
246231519	0	← z[0]
246231520	1	← z[1]
246231521	2	← z[2]
⋮	⋮	
246231545	246231519	← z
⋮	⋮	
246234411	246231519	
⋮	⋮	

```
void foo( int y[] ) {
   for (int i=0;i<3;i++)
      y[i] = y[i] + i;
}
main() {
   int z[3];
   for (int i=0;i<3;i++)
      z[i] = 0;
   foo(z);
   /* do something */
}
```

MEMORY

PROGRAM

Passing Arrays as Parameters

- The declaration

  ```
  void foo( int y[] )
  ```

  passes the pointer `y` as parameter.
- Its type is specified to to mean that it is a pointer to not one integer, but to an array of integers.
- We can also write

  ```
  void foo( int *y )
  ```

  instead.

Passing Arrays as Parameters

- The declaration
  `void foo( int y[] )`
  passes the pointer `y` as parameter.
- Its type is specified to to mean that it is a pointer to not one integer, but to an array of integers.
- We can also write
  `void foo( int *y )`
  instead.

Pointer Arithmetic

- Since pointer variables store memory addresses, we can add and subtract from them to access other addresses!
- Caution: This must be done with extreme care!!
- For a pointer variable y, *(y1)+ refers to the next memory location.
- Depending on the type of variable, this can be one or more bytes away.

Pointer Arithmetic

- Since pointer variables store memory addresses, we can add and subtract from them to access other addresses!
- Caution: This must be done with extreme care!!
- For a pointer variable y, *(y1)+ refers to the next memory location.
- Depending on the type of variable, this can be one or more bytes away.

Pointer Arithmetic

- Since pointer variables store memory addresses, we can add and subtract from them to access other addresses!
- Caution: This must be done with extreme care!!
- For a pointer variable `y`, `*(y1)+` refers to the next memory location.
- Depending on the type of variable, this can be one or more bytes away.