ESc 101: Fundamentals of Computing

Lecture 23

Feb 24, 2010

Outline

*A AND **A

- Consider `int A[SIZE][SIZE]` declaration.
- As observed, this declares `SIZE+1` pointers: `A`, `A[0]`, ..., `A[SIZE-1]`.
- `A[i]` points to the element `A[i][0]`.
- `A` also points to the element `A[0][0]`.
- `A[0]`, ..., `A[SIZE-1]` can be viewed as an array of pointers.
- In that case, `A` should point to `A[0]`!

*A AND **A

- Consider `int A[SIZE][SIZE]` declaration.
- As observed, this declares `SIZE+1` pointers: `A`, `A[0]`, ..., `A[SIZE-1]`.
- `A[i]` points to the element `A[i][0]`.
- `A` also points to the element `A[0][0]`.
- `A[0]`, ..., `A[SIZE-1]` can be viewed as an array of pointers.
- In that case, `A` should point to `A[0]`!

*A AND **A

- Consider `int A[SIZE][SIZE]` declaration.
- As observed, this declares `SIZE+1` pointers: `A`, `A[0]`, ..., `A[SIZE-1]`.
- `A[i]` points to the element `A[i][0]`.
- `A` also points to the element `A[0][0]`.
- `A[0]`, ..., `A[SIZE-1]` can be viewed as an array of pointers.
- In that case, `A` should point to `A[0]`!

*A AND **A

- Consider `int A[SIZE][SIZE]` declaration.
- As observed, this declares `SIZE+1` pointers: `A`, `A[0]`, ..., `A[SIZE-1]`.
- `A[i]` points to the element `A[i][0]`.
- `A` also points to the element `A[0][0]`.
- `A[0]`, ..., `A[SIZE-1]` can be viewed as an array of pointers.
- In that case, `A` should point to `A[0]`!

*A AND **A

- Consider `int A[SIZE][SIZE]` declaration.
- As observed, this declares `SIZE+1` pointers: `A`, `A[0]`, ..., `A[SIZE-1]`.
- `A[i]` points to the element `A[i][0]`.
- `A` also points to the element `A[0][0]`.
- `A[0]`, ..., `A[SIZE-1]` can be viewed as an array of pointers.
- In that case, `A` should point to `A[0]`!

*A AND **A

- However, there is no need to store the address of `A[0]` as we never change the contents of `A[0]`.
- So the treatment of `A` is a little inconsistent:
  - It points to `A[0][0]`.
  - It also "behaves" as pointer to `A[0]`, in that `*A` is the same as `A[0]`.
  - Which, of course, means that both `A` and `*A` are addresses of `A[0][0]`!
- Since `*(A[0])` is the location `A[0][0]` and `*A` is same as `A[0]`, `**A` is also the location `A[0][0]`.
- Similarly, `*(A+1)` is same as `A[1]`, `*(A+2)` is same as `A[2]` as so on.

*A AND **A

- However, there is no need to store the address of `A[0]` as we never change the contents of `A[0]`.
- So the treatment of `A` is a little inconsistent:
  - It points to `A[0][0]`.
  - It also "behaves" as pointer to `A[0]`, in that `*A` is the same as `A[0]`.
  - Which, of course, means that both `A` and `*A` are addresses of `A[0][0]`!
- Since `*(A[0])` is the location `A[0][0]` and `*A` is same as `A[0]`, `**A` is also the location `A[0][0]`.
- Similarly, `*(A+1)` is same as `A[1]`, `*(A+2)` is same as `A[2]` as so on.

*A AND **A

- However, there is no need to store the address of `A[0]` as we never change the contents of `A[0]`.
- So the treatment of `A` is a little inconsistent:
  - It points to `A[0][0]`.
  - It also "behaves" as pointer to `A[0]`, in that `*A` is the same as `A[0]`.
  - Which, of course, means that both `A` and `*A` are addresses of `A[0][0]`!
- Since `*(A[0])` is the location `A[0][0]` and `*A` is same as `A[0]`, `**A` is also the location `A[0][0]`.
- Similarly, `*(A+1)` is same as `A[1]`, `*(A+2)` is same as `A[2]` as so on.

*A AND **A

- However, there is no need to store the address of `A[0]` as we never change the contents of `A[0]`.
- So the treatment of `A` is a little inconsistent:
  - It points to `A[0][0]`.
  - It also "behaves" as pointer to `A[0]`, in that `*A` is the same as `A[0]`.
  - Which, of course, means that both `A` and `*A` are addresses of `A[0][0]`!
- Since `*(A[0])` is the location `A[0][0]` and `*A` is same as `A[0]`, `**A` is also the location `A[0][0]`.
- Similarly, `*(A+1)` is same as `A[1]`, `*(A+2)` is same as `A[2]` as so on.

*A AND **A

- However, there is no need to store the address of `A[0]` as we never change the contents of `A[0]`.
- So the treatment of `A` is a little inconsistent:
  - It points to `A[0][0]`.
  - It also "behaves" as pointer to `A[0]`, in that `*A` is the same as `A[0]`.
  - Which, of course, means that both `A` and `*A` are addresses of `A[0][0]`!
- Since `*(A[0])` is the location `A[0][0]` and `*A` is same as `A[0]`, `**A` is also the location `A[0][0]`.
- Similarly, `*(A+1)` is same as `A[1]`, `*(A+2)` is same as `A[2]` as so on.

*A AND **A

- However, there is no need to store the address of `A[0]` as we never change the contents of `A[0]`.
- So the treatment of `A` is a little inconsistent:
  - It points to `A[0][0]`.
  - It also "behaves" as pointer to `A[0]`, in that `*A` is the same as `A[0]`.
  - Which, of course, means that both `A` and `*A` are addresses of `A[0][0]`!
- Since `*(A[0])` is the location `A[0][0]` and `*A` is same as `A[0]`, `**A` is also the location `A[0][0]`.
- Similarly, `*(A+1)` is same as `A[1]`, `*(A+2)` is same as `A[2]` as so on.

Making Sense of the Pointers

A Useful Assumption

Assume that `A` points to `A[0]` instead of `A[0][0]`.

- Then pointer arithmetic as above has no inconsistency.
- Therefore, even though not true, it is a useful assumption to make.

Making Sense of the Pointers

A Useful Assumption

Assume that `A` points to `A[0]` instead of `A[0][0]`.

- Then pointer arithmetic as above has no inconsistency.
- Therefore, even though not true, it is a useful assumption to make.

Making Sense of the Pointers

A Useful Assumption

Assume that `A` points to `A[0]` instead of `A[0][0]`.

- Then pointer arithmetic as above has no inconsistency.
- Therefore, even though not true, it is a useful assumption to make.