Operating Systems Lab (C+Unix)

Enrico Bini

University of Turin

Outline

Array of arrays ("matrices")

- C allows declaring **bi-dimensional arrays**

  `<type> v[<DIM1>][<DIM2>];`

  - allocates an array `v` of `DIM1` *elements*.
  - each *element* is an array of `DIM2` contiguous variables of type `<type>`
- Overall, it is allocated for `v` a contiguous amount of memory of `sizeof(<type>)*DIM1*DIM2` bytes
  - The first element in memory is `v[0][0]` then `v[0][1]` and so on
  - The element `v[i][DIM2-1]` (last element of "row" `i`) is followed by `v[i+1][0]` (first element of next "row")
  - The last element in memory is `v[DIM1-1][DIM2-1]`
- Example:

  `test-bi-array.c`

  ```
  int v[10][3];
  ```

  - `v[i][j]` is the `j`-th element of the `i`-th array in `v`
  - `v[i]` is the array of 3 `int` at position `i` in `v`
  - `v` is an array of 10 arrays of 3 `int` variables
- **WARNING**: elements are addressed by `v[i][j]` and not by `v[i,j]`

Array of pointers

- Pointers are variables
  - Arrays of pointers can be declared and used as arrays of any variable
- An array of pointers is declared by

```
<type> *v[<size>];
```

  which statically allocates an array of <size> pointers to <type>
- Example of initialization:

```
char * p[] = {
        "defghi",
        "jklmnopqrst",
        "abc"
        };
```

  initializes:
  - a vector p with three pointers p[0], p[1] and p[2] (read/write)
  - three strings pointed respectively by p[0], p[1] and p[2] (read-only)
- *test-array-ptr.c*

Usage of array of pointers: command-line arguments

- When commands are invoked at the shell, they may have a sequence of space-separated "*command-line arguments*"
- Example:

  `gcc -c my_file.c -o my_file`

  - the command is `gcc`
  - 4 command-line arguments follow
- Command-line arguments can be read and used within a program
- We have been writing the `main` as

```
int main() { /* body */}
```

  however, to read command-line arguments it must be written as

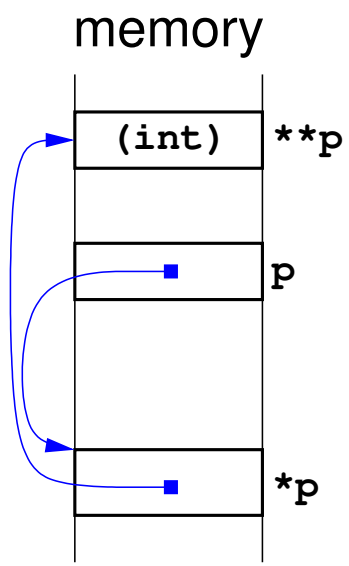
```
int main(int argc, char *argv[]) { /* body */}
```

  - `argc`: number of space-separated strings at command line
  - `argv`: array of pointers to each string
- *`test-command-line.c`*

Pointers to pointers

- C allows the declaration of pointers to pointers, for example by

```
int **p;
```



- in this case:
  - `p` is a pointer of type (`int **`) pointing to a memory address containing a variable of type `int *`
  - `*p` is a pointer of type (`int *`) pointing to a memory address containing a variable of type `int`
  - `**p` is an (`int`) variable
- Also variables of type

```
int **** p;
```

  are possible. Then `p` is a pointer to a pointer to a pointer to a pointer (4 times!!) to a variable of type `int`
- I never saw in the code more than 2 levels of dereferencing
- *test-ptr-ptr.c*

Pointers to functions

- The code of functions is in memory
- It is then possible to declare *pointers to functions*
- A pointer to a function is the address of the first instruction executed after the `CALL` assembly instruction
- A pointer to a function is declared by:

```
<type> (* var_name) (<param_types>);
```

  - `var_name`, name of the function pointer;
  - `type`, type returned by the function;
  - `<param_types>`, list of input types;

- Different than a function returning a pointer to `<type>`

```
<type> * var_name(<param_types>);
```

- Arrays of ponters to functions are also possible, by:

```
<type> (* var_name[LEN]) (<param_types>);
```

- *test-fun-ptr.c*