Software Tools

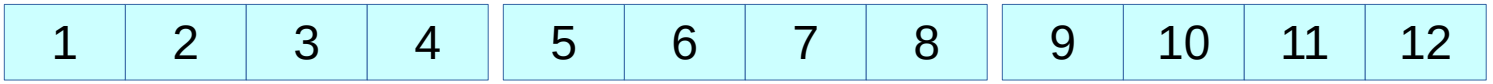
C, Unix (Linux), and tools

Initialization of Arrays

- We very often initialize static and external variables when defining them
  - This includes arrays
- Initializing automatic variables when defining them is a matter of style and convenience
  - Many programmers prefer to do it with explicit assignment

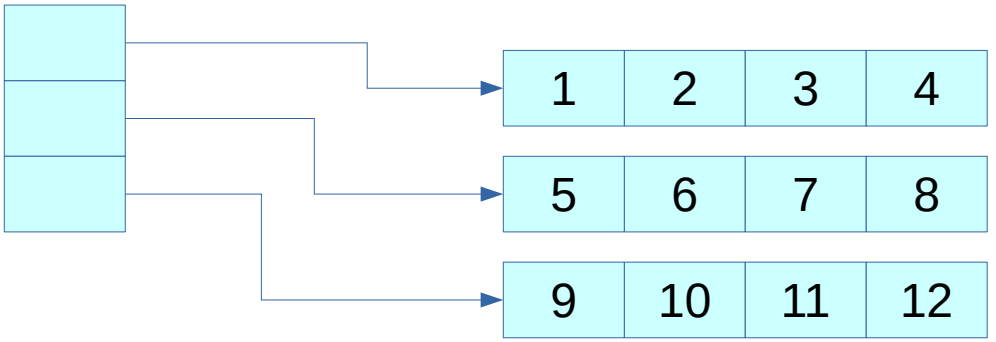
Initializing 2-D arrays

- A 2-D array is just a bunch of consecutive chunks of memory
  - Array int A[3][4] is 12 integers in a row (48 bytes)
  - We initialize with
    - static int A[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
    - The memory for this array will be initialized.



Initializing pointers to pointers

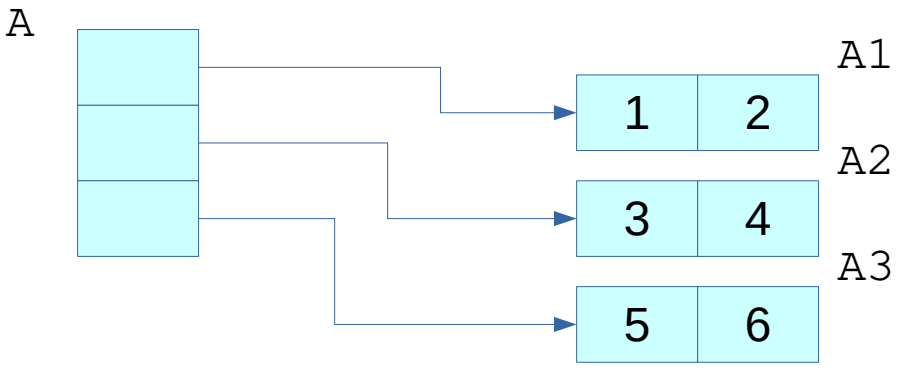
- An array using pointers to pointers looks different
- Needs more memory space, more memory references
- Far more flexible.



Initializing Pointers to Pointers

- We do it with similar syntax

```
static int A1[2] = {1, 2};
static int A2[2] = {3, 4};
static int A3[2] = {5, 6};
static int *(A[]) = {A1, A2, A3};
```



strcpy

- Four versions:

```
void strcpy(char *s, char *t)
{
  while ( (*s=*t)!='\0' ){
    s++;
    t++;
  }
}
```

```
void strcpy(char *s, char *t)
{
  int i;
  i=0;
  while ( (s[i]=t[i])!='\0' )
    i++;
}
```

```
void strcpy{char *s, char *t)
{
  while ( (*s++=*t++)!='\0' )
    ;
}
```

```
void strcpy{char *s, char *t)
{
  while (*s++=*t++)
    ;
}
```

Arrays of Pointers

- We have seen `argc` and `argv`
- `char s[]` means s is an array of characters
- `char *s[]` means the contents of address `s` is an array of characters

```
int main(int argc, char *argv[])
{             /* argc is argument count */
              /* argv is an array of strings */
  int i;

  for (i=0; i<argc; i++)
    printf("argv[%d]: %s\n", i, argv[i]);
  return 0;
}
```