Pointer arithmetic

• Pointer addition:

```c
int p[2];
&p[0] is the same as p
p++ increments p by sizeof(*p);
p1 – p2 = number of steps of sizeof(*p)
```
# Pointer arithmetic

Based on ESC101 lectures 2010, Manindra Agarwal

```c
int main()
{
    int m = 10;
    int n = 5;
    int z[2];
    int i;
    for (i=0; i<2; i++)
        z[i] = 100 +i ;
}
```

<table>
<thead>
<tr>
<th>&amp;m</th>
<th>30016</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;n</td>
<td>30012</td>
<td>n</td>
</tr>
<tr>
<td>z+1</td>
<td>30008</td>
<td>z[1]</td>
</tr>
<tr>
<td>&amp;z[0]</td>
<td>30004</td>
<td>z[0]</td>
</tr>
<tr>
<td></td>
<td>30000</td>
<td></td>
</tr>
<tr>
<td>29996</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29992</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29988</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29984</td>
<td>30000</td>
<td></td>
</tr>
<tr>
<td>29980</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Based on ESC101 lectures 2010, Manindra Agarwal
Pointer arithmetic

• Pointer addition:

```c
int p[2];
&p[0] is the same as p
p++ increments p by sizeof(*p);
p1 – p2 = number of steps of sizeof(*p)
```

• But one must be careful!!
  • Can overshoot array bounds without warning
Pointer arithmetic

```
int main()
{
    int m = 10;
    int n = 5;
    int z[2];
    int i;

    for (i=0; i<4; i++)
        z[i] = 100 + i;
}
```
Pointer arithmetic

- Pointer addition:
  
  ```c
  int p[2];
  &p[0] \text{ is the same as } \ p
  p++ \text{ increments } p \text{ by } \text{sizeof}(*p);
  p1 - p2 = \text{number of steps of} \text{sizeof}(*p)
  ```

- But one must be careful!!
  - Can overshoot array bounds without warning
  - What is the value of \( p \) Itself?

  Try printing “p” (file: `pointer_overflow.c`):
  ```c
  [never use this value (an address) – can change from run to run]
  ```
#include <stdio.h>

int main()
{
    int m = 10;
    int n = 5;
    int z[2];
    int i;

    int SIZE=4;   /* overflows; compare with SIZE=2 */
    for (i=0; i<SIZE; i++)
        z[i] = 100 + i;

    printf("m, n: %d, %d ; z[0], z[1]: %d %d \n", m, n, z[0], z[1]);

    printf("m, n, z[0], z[1]: %d %d %d %d \n", m, n, z[0], z[1]);

    printf("z, &z[0], &z[1]: %d %d %d %d &m, &n: %d %d \n", z, &z[0], &z[1], &m, &n);
}
pointer_overflow.c

With
    int SIZE=2;

```c
>gcc pointer_overflow.c -o ptr
>ptr
  m, n: 10, 5 ; z[0], z[1]: 100 101
  z, &z[0], &z[1]: 2293512 2293512 2293516 ; &m, &n: : 2293524; 2293520
```

int SIZE=4;

```c
>ptr
  m, n: 103, 102 ; z[0], z[1]: 100 101
  z, &z[0], &z[1]: 2293512 2293512 2293516 ; &m, &n: : 2293524; 2293520
```
Multi-dimensional arrays

• All arrays in C are implemented as pointers
• In a multi-dimensional array e.g.
  
  ```
  float average[12][3]
  ```
  
  average points to a bank of 12 pointers, each pointing to an array of 3 floats.
• Each row `average[i]` is an 1-D array of 3 elements
• `**average` or `*(average)` is the [0][0] element
• `average+1` : what does this represent?
• `*(average+i)+j` is equivalent to `average[i][j]`
• `float marks[12][30][3];` what is `***marks`?
```c
#include <stdio.h>

int main()
{
    int i, j, SIZE1=2, SIZE2=3;
    int avg[SIZE1][SIZE2];
    int *x;

    for (i=0; i<SIZE1; i++)
        for (j=0; j<SIZE2; j++)
            *(avg+i)+j) = 50 +i*SIZE2 +j;

    printf(" \n addr avg,  *avg, **avg= %d %d %d; 
", avg, *avg, **avg);

    for (i=0; i<SIZE1; i++)
    {
        printf(" \n addr &avg[%d]= %d; values: ", i, &avg[i]);
        x = avg[i];
        for (j=0; j<SIZE2; j++)
            printf("[%d][%d]: %d ", i, j, *x++);
    }
}
```