Pointers

A pointer is a variable that contains the address of some data object.

Major uses:
- direct access to specific mem location
- manipulation of array elements
- prog. management of dynamically alloc. mem.
- passing arguments by reference
In assembly lang., indirect addressing mode represents exactly same concept.

Size of pointer is machine dependent

(32 bits on x86 ISA used in class)
<table>
<thead>
<tr>
<th>short int var1</th>
<th>0x7fa8</th>
</tr>
</thead>
<tbody>
<tr>
<td>char var2</td>
<td>0x7faa</td>
</tr>
<tr>
<td>char var3</td>
<td>0x7fabc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>char * ptr2</th>
<th>0xaaa</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x7f</td>
</tr>
<tr>
<td></td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x00</td>
</tr>
</tbody>
</table>

← ptr2 points to var2
declaring pointers

\[ \text{[type]} \ast \text{[name]} \{=\text{[init \ val]}\}; \]

\[ \text{e.g.,} \]

\[ \text{char } c = \text{'A'}; \]

\[ \text{char } \ast p = \& c; \]

\& is operator that returns address of variable

\& c \text{ addr of } c

\& p \text{ addr of } p
* is the "indirection" or "reference" operator. When applied to a pointer it accesses the "pointed-to" memory.

* can go on either side of assignment.

Note: name of array is equivalent to

to pointer to first element

\[\text{int } \*x[10]; \quad x \equiv \&x[0]\]
int x = 1;
int y = 2;
int z <= 10;
int x > 0;
int x = 9;
<table>
<thead>
<tr>
<th>Addr</th>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>2004</td>
<td>Y</td>
<td>2</td>
</tr>
<tr>
<td>2008</td>
<td>X</td>
<td>13</td>
</tr>
<tr>
<td>2010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2024</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2028</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2030</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2034</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
i_p = \& x;  \quad /\star i_p \text{ points to } x \star/ \\
y = \& i_p;  \quad /\star \& i_p \text{ access contents of } x \star/ \\
i_p = 0;  \quad /\star \text{ sets } x \text{ to } 0 \star/ \\
i_p = \lfloor 0 \rfloor;  \quad /\star i_p \text{ now points to first element of } z \star/ \\
i_p = 13;  \quad /\star z(0) = 13 \star/ \\
i_p = i_p + 10;  \quad /\star \text{ adds } 10 \text{ to } z(0) \star/ \\
i_p = i_p - 1;  \quad /\star \text{ equiv. to } (\& i_p) - 1 \text{ and } - (\& i_p), \text{ but not } \& i_p - 1 \star/ \\
i = i_p;
\((\times iq) \rightarrow \) \\
\(i q + 4; \) \\
\(i q = 0\) \\
\(i q = i q + 4; \) \\
\(i q = 7; \) \\
\(i q = i q + 8; \) \\
\(i q = 16; \) \\
\(i q = \text{anything} \) \\
\(\times 4q = \text{BAD, BAD, BAD!} \)
physical addr example

unsigned int & status;

status = (unsigned int *)&0x8000040c;

for (i = 0; i < 24; i++) {
    mask = 1 << i;
    stat_bit(i) = (status) & mask;
}

function arguments

void swap (int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}

swap (a, b);

C passes parameters a, b by value
void swap (int *px, int *py)

int temp;

temp = *px;

*px = *py;

*py = temp;

swap (&a, &b);
The call is by reference
command line arguments in C

```c
int main (int argc, char * argv[]) {
    argc is number of arguments (incl. program name)
    argv is array of strings
    argv[0] : program name
    argv[1] : 1st argument
    argv[argc-1] : last arg.
```
```c
int main(int argc, char *argv[]) {
    int i;

    printf("\%d args in", argc);
    for (i = 0; i < argc; i++) {
        printf("\%d: \%s in", i, argv[i]);
    }

    return 0;
}

echo args.c
```
```
$> ./echoargs Hi there

3 args
0: ./echoargs
1: Hi
2: there

7 gcc -o echoargs
```