IS12 - Introduction to Programming
Lecture 21: Pointers

Peter Brusilovsky
http://www2.sis.pitt.edu/~peterb/0012-072/

Outline

- Computer memory
- Pointers
- Pointers and function parameters
- Pointers and arrays
- Address Arithmetic
Computer memory

- Computer memory is a huge array of consecutively numbered \textit{memory cells}
- On most modern computers a cell can store one \textit{byte} of data
- An address of a memory cell is its number in an array
- Data of different type occupies one or more continuous cells (bytes)

```
short a;
short width;
int i;
char ch;
char line[6];
```
Pointers

- A pointer is a variable that can store an address of another variable or data object located in memory (i.e., 112304)
- We say that a pointer points to a variable that is stored at that address
- A pointer itself usually occupies 4 bytes of memory (then it can address cells from 0 to $2^{32}-1$)
Declaring pointers

- In C you can specify the type of variable a pointer can “points to”
  - `int *ad; /* pointer to int */`
  - `char *s; /* pointer to char */`
  - `float *fp; /* pointer to float */`
  - `char **s; /* pointer to variable that is a pointer to char */`

Operations with pointers

- We can assign an address to a pointer
  - `int *p1, *p2; int a, b; p1 = &a; /* &a - an address of a */`
- We can assign pointers to each other
  - `p2 = p1;`
- We can dereference pointers
  - `*p1 = 3; /* same as a = 3 */`
  - `b = *p1; /* same as b = a */`
Operations & and *

\[ a = 1999; \]

\[ \text{int } a; \]
\[ \text{int } b; \]
\[ \text{int } *p1; \]
\[ \text{int } *p2; \]

\[ a \]
\[ b \]
\[ p1 \]
\[ p2 \]
Operations & and *

```c
int a;
int b;
int *p1;
int *p2;

*p1 = 2000;
b = *p1;
```
Operations & and *

```c
int a;
int b;
int *p1;
```

(*p1)++ ;

```c
int *p2;
```

Operations & and *

```c
p2 = &b ;
```

```c
int a;
int b;
int *p1;
int *p2;
```
Operations & and *

112300
112301
112302
112303
112304
112305
112306
112307
112308
112309
112310
112311
112312
112313
112314
112315

```c
int a;
int b;
int *p1;
int *p2;
p1 = p2;
```

Operations & and *

112300
112301
112302
112303
112304
112305
112306
112307
112308
112309
112310
112311
112312
112313
112314
112315

```c
int a;
int b;
int *p1;
int *p2;
*p1 = 0;
```
Operations & and *

```c
int a;
int b;
int *p1;
int *p2;
```

```c
a = *p2 + 10;
```

Pointers and function arguments

/* bad example of swapping
a function can't change parameters */
```c
void bad_swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

/* good example of swapping
-a function can't change parameters but if a parameter is a pointer it can change the value it points to */
```c
void good_swap(int *px, int *py)
{
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}
```
Pointers and function arguments

```c
#include <stdio.h>
void bad_swap(int x, int y);
void good_swap(int *p1, int *p2);

void main() {
    int a = 1, b = 999;

    printf("a = %d, b = %d\n", a, b);
    bad_swap(a, b);
    printf("a = %d, b = %d\n", a, b);
    good_swap(&a, &b);
    printf("a = %d, b = %d\n", a, b);
}
```

Inside story

```
main

    int a;
    int b;

    a 1
    b 999
```
Inside story

```c
int a;
int b;

int *px;
int *py;

swap(&a, &b);
```

```c
temp = *px;
```

Inside story

```c
int a;
int b;

int *px;
int *py;

swap(&a, &b);
```

```c
temp = *px;
```
Inside story

```c
int a;
int b;

*px = *py;
```

```c
main

int *px;

swap

int *py;
```

```c
*py = temp;
```

```c
int a;
int b;
```

```c
a 999
b 999
```

```c
1
```

```c
1
```

```c
px
```
Inside story

```c
int a;
int b;
main

The magical & in scanf()

- `scanf` is able to put the input into a variable because we pass its address
  ```c
  int capital = 0;
  scanf("%d", &capital);
  ```
- What happens without &?
  ```c
  scanf("%d", capital);
  ```
- `scanf` thinks that the value of capital is an address where it has to place data!
**Pointers and Arrays**

- We can address array elements

```c
int ar[10];
int *pa, *pb, *pc;
pa = &ar[0] /* same as pa = ar */
pb = pa + 1; pc = pa + 9;
```

**Address Arithmetic**

- We can add and subtract integers to/from pointers - the result is a pointer to another element of this type

```c
int *pa; char *s;
s-1 ⇒ points to char before s (1 subtracted)
pa+1 ⇒ points to next int (4 added!)
s+9 ⇒ points to 9th char after s (9 added)
++pa ⇒ increments pa to point to next int
```
Example 2: String Comparison

```c
#define MAXLEN 100
int strcmp(char *p1, char *p2);
int getline(char *line, int max);
void main() {
    char a[MAXLEN], b[MAXLEN];
    int comp;
    getline(a, MAXLEN);
    getline(b, MAXLEN);
    comp = strcmp(a, b);
    if(comp > 0)
        printf("First line is greater than second\n");
    else if (comp < 0)
        printf("First line is less than second\n");
    else printf("These lines are equal!\n");
}
```
Example 2: String Comparison

Example 2: String Comparison
Compare string comparisons

/* pointer version */
/* strcmp: return <0 if s<t, 
  0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
  for( ; *s == *t; s++, t++)
    if(*s == '\0')
      return 0;
  return *s - *t;
}

/* array version */
/* strcmp: return <0 if s<t, 
  0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
  int i;
  for( ; s[i] == t[i]; i++)
    if(s[i] == '\0')
      return 0;
  return s[i] - t[i];
}

Before Next Lecture:

- Do reading assignment
- Perry: Chapter 24; First reading of Chapter 25, also external materials
- Run Classroom Examples
- Knowledge Sea pair homework: find 4 pages that are good to read for a specific lecture. Assign praise comment
- Read other relevant material on pointers in Knowledge Sea