

**CSE1303 Part A**  
**Data Structures and Algorithms**  
**Summer Semester 2003**

**Lecture A2 – Pointers (Revision)**

Kymerly Fergusson

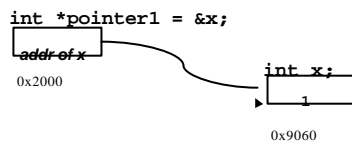
**Overview**

- Revision of Pointers
- Pointers and structs
- Basic Pointer Arithmetic
- Pointers and Arrays

2

**Pointers**

- A pointer is a **datatype**.
- You can create variables of this type as you would of other types.
- Contains a **memory address**.
- Points to a **specific data type**.



3

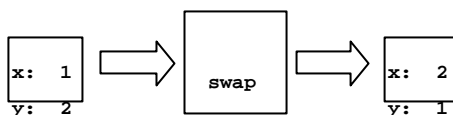
**Pointer Operations**

```
Type  object;
Type* aPtr;
```

- `aPtr = &object` assigns the address of `object` to `aPtr`.
- `*aPtr` allows access to the `object` through the `pointer`.
- If `aPtr` points to a structure then `(*aPtr).member` is equivalent to `aPtr-> member`

4

**Pointers and Function Arguments**



5

```
#include <stdio.h>

void fakeSwap(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1, y = 2;

    fakeSwap(x, y);
    printf("%d %d\n", x, y);
}
```

**Solution 1**

6

```
#include <stdio.h>

void fakeSwap(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1, y = 2;

    fakeSwap(x, y);
    printf("%d %d\n", x, y);
}
```

**Solution 1**

x:  0x2000  
 Y:  0x2010

7

```
#include <stdio.h>

void fakeSwap(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1, y = 2;

    fakeSwap(x, y);
    printf("%d %d\n", x, y);
}
```

**Solution 1**

tmp:  0x2060  
 a:  0x2038  
 b:  0x2040

x:  0x2000  
 Y:  0x2010

8

```
#include <stdio.h>

void fakeSwap(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1, y = 2;

    fakeSwap(x, y);
    printf("%d %d\n", x, y);
}
```

**Solution 1**

tmp:  0x2060  
 a:  0x2038  
 b:  0x2040

x:  0x2000  
 Y:  0x2010

9

```
#include <stdio.h>

void fakeSwap(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1, y = 2;

    fakeSwap(x, y);
    printf("%d %d\n", x, y);
}
```

**Solution 1**

tmp:  0x2060  
 a:  0x2038  
 b:  0x2040

x:  0x2000  
 Y:  0x2010

10

```
#include <stdio.h>

void fakeSwap(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int x = 1, y = 2;

    fakeSwap(x, y);
    printf("%d %d\n", x, y);
}
```

**Solution 1**

tmp:  0x2060  
 a:  0x2038  
 b:  0x2040

x:  0x2000  
 Y:  0x2010

11

```
#include <stdio.h>

void fakeSwap(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
}


int main()
{
    int x = 1, y = 2;

    fakeSwap(x, y);
    printf("%d %d\n", x, y);
}
```

**Solution 1**

x:  0x2000  
 Y:  0x2010

12



```

#include <stdio.h>

void trueSwap(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1, y = 2;

    trueSwap(&x, &y);
    printf("%d %d\n", x, y);
}

```

**Solution 2**

13

```

#include <stdio.h>

void trueSwap(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1, y = 2;

    trueSwap(&x, &y);
    printf("%d %d\n", x, y);
}

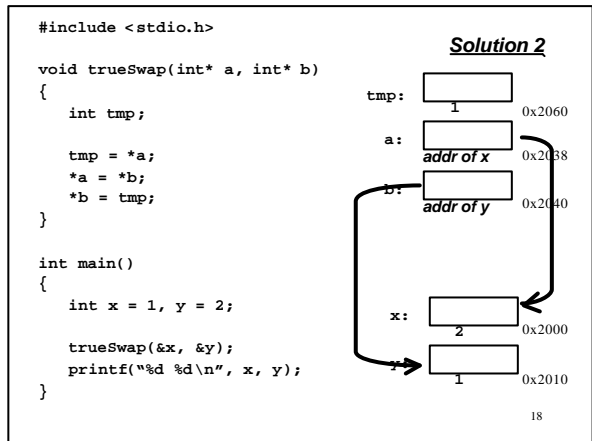
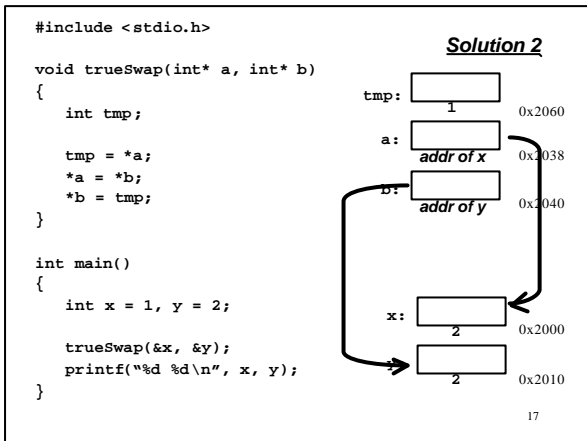
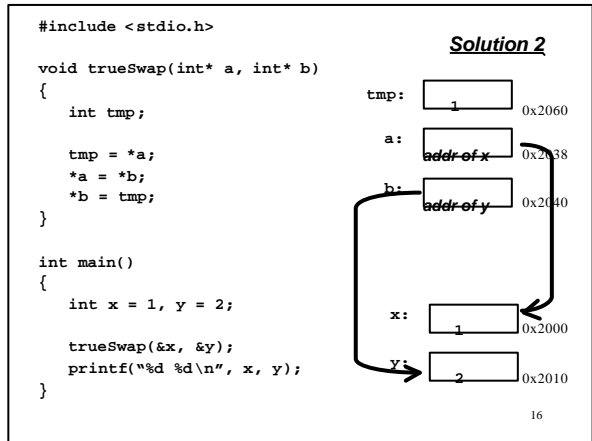
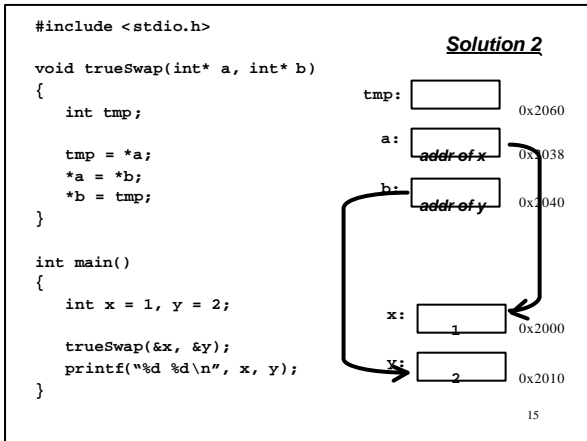
```

**Solution 2**

x:  0x2000

y:  0x2010

14



```
#include <stdio.h>

void trueSwap(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x = 1, y = 2;
    trueSwap(&x, &y);
    printf("%d %d\n", x, y);
}
```

**Solution 2**

19

## More on Pointers

- You can print the address stored in a pointer using the `%p` conversion specifier  
Example: `printf("%p", numPtr);`

`scanf()` needs to know where to put the value - it needs the address of the variable as it takes pointers as parameters.

Example: `int i;`  
`scanf("%d", &i);`

20

```
struct DataBaseRec
{
    /* Very Large Structure */
};

typedef struct DataBaseRec DataBase;
```

21

```
DataBase readNewEntry(DataBase theDataBase)
{
    /* Some code */
    return theDataBase;
}

void printDataBase(DataBase theDataBase)
{
    /* Some more code */
}
```

22

```
void readNewEntry(DataBase* dataBasePtr)
{
    /* Some code */
}

void printDataBase(const DataBase* dataBasePtr)
{
    /* Some more code */
}
```

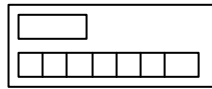
23

## Why Pointers?

- To modify a variable in a function that is not a global, or a local to that function
- To save space
- To save time
- To use dynamic memory (Lecture A6)
- Used extensively in linked structures

24

## Pointers and Structures



```

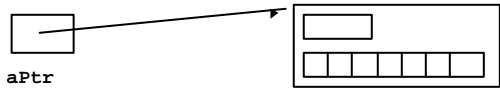
struct bookRec{
    float price;
    char name[7];
};
typedef struct bookRec Book

Book temp;

scanf("%d %s", &temp.price, temp.name);
    
```

25

## Pointers and Structures



```

aPtr
struct bookRec{
    float price;
    char name[7];
};
typedef struct bookRec Book

Book *aPtr;
Book temp;
aPtr = &temp;

scanf("%d %s", &(aPtr->price), aPtr->name);
    
```

Why do you need the brackets?

26

## Pointers and Functions

- To enable a function to access and change an object.
- For large structures it is more efficient.
- Use **const** specifier whenever a constant is intended.

27

## Pointers and Arrays

```

Type array[size];
Type* pPtr = array + i;
Type* qPtr = array + j;
    
```

- The name **array** is equivalent to **&array[0]**
- **pPtr++** increments **pPtr** to point to the next element of **array**.
- **pPtr += n** increments **pPtr** to point to **n** elements beyond where it currently points.
- **pPtr - qPtr** equals **i - j**.

28

## Pointers and Arrays (cont)

A normal 1 dimensional array:

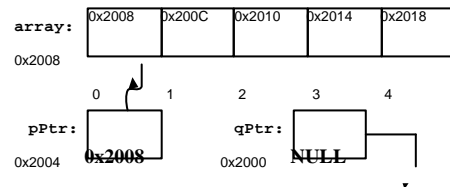
```

Type array[size];
    
```

- **array[0]** is equivalent to **\*array**
- **array[n]** is equivalent to **\*(array + n)**

29

## Basic Pointer Arithmetic



```

float array[5];
float* pPtr = array;
float* qPtr = NULL;
    
```

30

```

array: 0x2008 0x200C 0x2010 0x2014 0x2018
0x2008 [ ] [ ] [ ] [ ] [ ]
         0   1   2   3   4
pPtr:  0x2004 [0x200C]
qPtr:  0x2000 [NULL]
  
```

```

float array[5];
float* pPtr = array;
float* qPtr = NULL;

pPtr++; /* pPtr now holds the address: &array[1] */
  
```

31

```

array: 0x2008 0x200C 0x2010 0x2014 0x2018
0x2008 [ ] [ ] [ ] [ ] [ ]
         0   1   2   3   4
pPtr:  0x2004 [0x2018]
qPtr:  0x2000 [NULL]
  
```

```

float array[5];
float* pPtr = array;
float* qPtr = NULL;

pPtr++; /* pPtr = &array[1] */
pPtr += 3; /* pPtr now hold the address: &array[4] */
  
```

32

```

array: 0x2008 0x200C 0x2010 0x2014 0x2018
0x2008 [ ] [ ] [ ] [ ] [ ]
         0   1   2   3   4
pPtr:  0x2004 [0x2018]
qPtr:  0x2000 [0x2010]
  
```

```

float array[5];
float* pPtr = array;
float* qPtr = NULL;

pPtr++; /* pPtr = &array[1] */
pPtr += 3; /* pPtr = &array[4] */
qPtr = array + 2; /*qPtr now holds the address &array[2]*/
  
```

33

```

array: 0x2008 0x200C 0x2010 0x2014 0x2018
0x2008 [ ] [ ] [ ] [ ] [ ]
         0   1   2   3   4
pPtr:  0x2004 [0x2018]
qPtr:  0x2000 [0x2010]
  
```

```

float array[5];
float* pPtr = array;
float* qPtr = NULL;

pPtr++; /* pPtr = &array[1] */
pPtr += 3; /* pPtr = &array[4] */
qPtr = array + 2; /* qPtr = &array[2] */
printf("%d\n", pPtr-qPtr);
  
```

34

```

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

char*strcpy(char* s, char* t)
{
    char* p = s;
    while (*p != 0)
    {
        *p = *t;
        p++;
        t++;
    }
    return s;
}
  
```

35

### Revision

- Pointers, operations, structures, arguments, arithmetic and arrays
  - address operator (&), and dereferencing operator (\*)
  - structure pointer operator (->)
  - Look at "swap" example

### Revision: Reading

- Kruse - Appendix C.6
- Deitel & Deitel - Chapter 7
- King - Chapter 11, 12
- Langsam - Chapter 1.1
- Standish - Chapter 2.3
- Kernighan and Ritchie - Chapter 5.1 – 5.10, 6.2

### Preparation

Next lecture: **Stacks**

- Read 3.1.1 to 3.1.5 in Kruse et al.

36