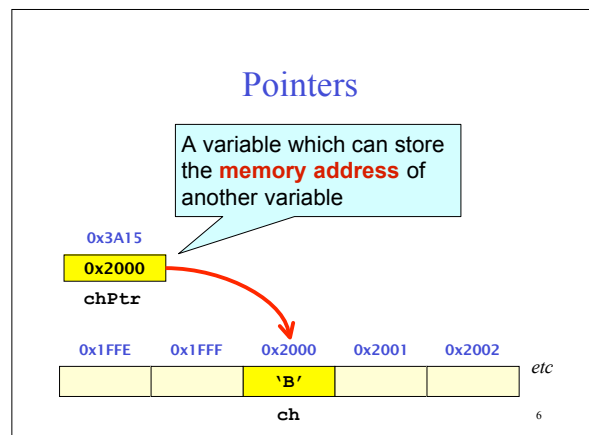
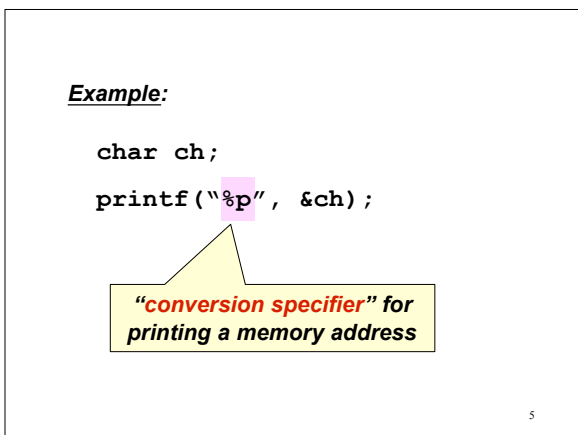
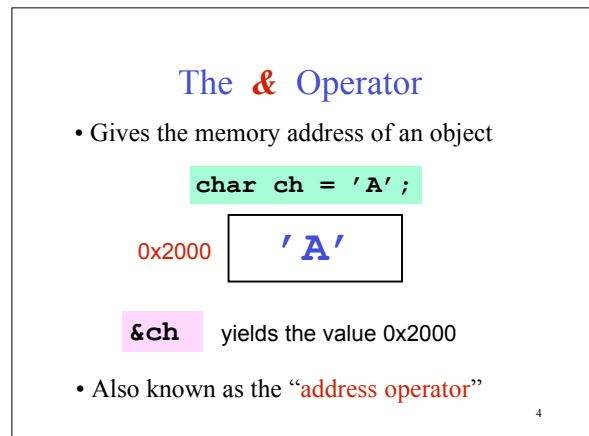
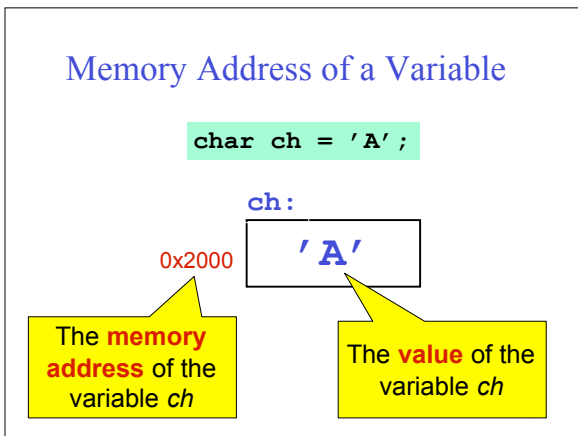


# CSE1301 Computer Programming Lecture 16 Pointers

1

- ## Topics
- Introduction to pointers
  - Pointers and function parameters
- 2



## Pointers

- A pointer is a **variable**
- Contains a **memory address**
- Points to a specific **data type**
- Pointer variables are usually named **varPtr**

7

### Example:

```
char* cPtr;
```

cPtr:

0x2004



Can store an **address** of variables of type **char**

- We say *cPtr* is a **pointer** to char

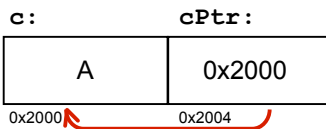
8

## Pointers and the & Operator

### Example:

```
char c = 'A';
char *cPtr;
cPtr = &c;
```

Assigns the address of *c* to *cPtr*



9

## Notes on Pointers

- We can have pointers to any data type

### Example:

```
int* numPtr;
float* xPtr;
```

- The **\*** can be anywhere between the type and the variable

### Example:

```
int *numPtr;
float * xPtr;
```

10

## Notes on Pointers (cont)

- You can assign the address of a variable to a “compatible” pointer using the **&** operator

### Example:

```
int aNumber;
int *numPtr;
numPtr = &aNumber;
```

- You can print the address stored in a pointer using the **%p** conversion specifier

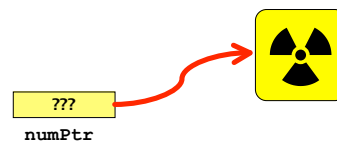
Example: `printf("%p", numPtr);`

11

## Notes on Pointers (cont)

```
int *numPtr;
```

Beware of pointers which are not initialized!



12

### Notes on Pointers (cont)

- When declaring a pointer, it is a good idea to always initialize it to **NULL** (a special pointer constant)

```
int *numPtr = NULL;
```

Diagram showing a variable `numPtr` containing the value `NULL`.

13

### The \* Operator

- Allows pointers to access variables they point to
- Also known as “dereferencing operator”
- Should not be confused with the `*` in the pointer declaration

14

### Pointers and the \* Operator

**Example:**

```
char c = 'A';
char *cPtr = NULL;
cPtr = &c;
*cPtr = 'B';
```

Changes the value of the variable which cPtr points to

Diagram showing memory layout:

<code>c:</code>	<code>cPtr:</code>
B	0x2000
0x2000	0x2004

15

### Easy Steps to Pointers

- Step 1:** Declare the variable to be pointed to

```
int num;
char ch = 'A';
float x;
```

Diagram showing memory layout:

<code>num:</code>	
<code>ch:</code>	'A'
<code>x:</code>	

16

### Easy Steps to Pointers (cont)

- Step 2:** Declare the pointer variable

```
int num;
char ch = 'A';
float x;

int* numPtr = NULL;
char *chPtr = NULL;
float * xPtr = NULL;
```

Diagram showing pointer variables and their values:

<code>numPtr:</code>	NULL
<code>chPtr:</code>	NULL
<code>xPtr:</code>	NULL

Diagram showing memory layout:

<code>num:</code>	
<code>ch:</code>	'A'
<code>x:</code>	

17

### Easy Steps to Pointers (cont)

- Step 3:** Assign address of variable to pointer

```
int num;
char ch = 'A';
float x;

int* numPtr = &num;
char *chPtr = &ch;
float * xPtr = &x;
```

Diagram showing pointer variables and their values:

<code>numPtr:</code>	addr of num
<code>chPtr:</code>	addr of ch
<code>xPtr:</code>	addr of x

Diagram showing memory layout:

<code>num:</code>	
<code>ch:</code>	'A'
<code>x:</code>	

A pointer's type has to correspond to the type of the variable it points to

18

### Easy Steps to Pointers (cont)

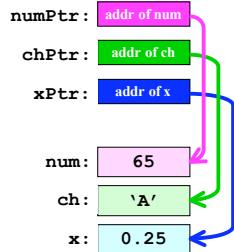
- **Step 4:** De-reference the pointers

```
int num;
char ch = 'A';
float x;

int* numPtr = NULL;
char *chPtr = NULL;
float * xPtr = NULL;

numPtr = &num;
chPtr = &ch;
xPtr = &x;

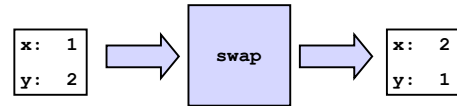
*xPtr = 0.25;
*numPtr = *chPtr;
```



19

### Pointers and Function Parameters

- **Example:** Function to swap the values of two variables



20

```
#include <stdio.h>
void swap1(int a, int b)
{
    int tmp;

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

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

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

Bad swap

21

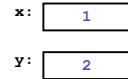
```
#include <stdio.h>
void swap1(int a, int b)
{
    int tmp;

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

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

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

Bad swap



22

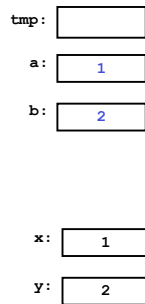
```
#include <stdio.h>
void swap1(int a, int b)
{
    int tmp;

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

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

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

Bad swap



23

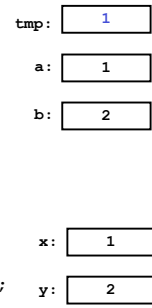
```
#include <stdio.h>
void swap1(int a, int b)
{
    int tmp;

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

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

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

Bad swap



24

Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

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

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

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

tmp:	<input type="text" value="1"/>
a:	<input type="text" value="2"/>
b:	<input type="text" value="2"/>
x:	<input type="text" value="1"/>
y:	<input type="text" value="2"/>

25

Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

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

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

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

tmp:	<input type="text" value="1"/>
a:	<input type="text" value="2"/>
b:	<input type="text" value="1"/>
x:	<input type="text" value="1"/>
y:	<input type="text" value="2"/>

26

Bad swap

```
#include <stdio.h>


void swap1(int a, int b)
{
    int tmp;

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

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

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

tmp:	<input type="text" value="1"/>
a:	<input type="text" value="2"/>
b:	<input type="text" value="1"/>
x:	<input type="text" value="1"/>
y:	<input type="text" value="2"/>



27

Good swap

```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

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

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

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```

28

Good swap

```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

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

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

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```

x:	<input type="text" value="1"/>
y:	<input type="text" value="2"/>

29

Good swap

```
#include <stdio.h>

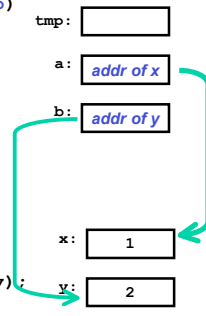
void swap2(int* a, int* b)
{
    int tmp;

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

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

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```

tmp:	<input type="text"/>
a:	<input type="text" value="addr of x"/>
b:	<input type="text" value="addr of y"/>
x:	<input type="text" value="1"/>
y:	<input type="text" value="2"/>



30

Good swap

```
#include <stdio.h>
void swap2(int* a, int* b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}
int main()
{
    int x = 1, y = 2;
    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```

31

Good swap

```
#include <stdio.h>
void swap2(int* a, int* b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}
int main()
{
    int x = 1, y = 2;
    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```

32

Good swap

```
#include <stdio.h>
void swap2(int* a, int* b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}
int main()
{
    int x = 1, y = 2;
    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```

33

Good swap

```
#include <stdio.h>
void swap2(int* a, int* b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}
int main()
{
    int x = 1, y = 2;
    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```

34

### Pointers and Function Arguments

- Change the value of an actual parameter variable
- `scanf` demystified

```
char ch;
int numx;
float numy;
scanf("%c %d %f", &ch, &numx, &numy);
```

35

### Reading

- King
  - Chapter 11
- Deitel and Deitel
  - Chapter 7 (7.1-7.4)

36