

POINTERS AND DYNAMIC ALLOCATION IN C

Will Landau

OUTLINE

- Pointers
- Arrays
- Bus errors and segmentation faults
- Dynamic allocation

Pointer: An address in memory of a stored C object.

Address (in bytes)	Stored Value
24399440	3
24399444	5
24399448	28
24399452	7
24399456	-2
:	:

ex1.c:

```
#include <stdio.h>

int main() {
    int a = 0;

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

output:

```
~/Desktop> gcc ex1.c -o ex1
~/Desktop> ./ex1
a = 0
&a = 1355533180
~/Desktop>
```

Address (in bytes)	Stored Value
1355533176	3
1355533180	0
1355533184	28
1355533188	7
1355533192	-2
:	:

value of a

$\&a$, pointer to a.

ex2.c:

```
#include <stdio.h>

int main() {
    int a = 0;
    int *pa;

    *pa = a;
    *pa = *pa + 1;

    printf("a = %d\n", a);
    printf("*pa = %d\n", *pa);
    printf("pa = %d\n", pa);
}
```

output:

```
~/Desktop> gcc ex2.c -o ex2
~/Desktop> ./ex2
a = 0
*pa = 1
pa = 1458113424
~/Desktop>
```

$\&a$, pointer to a

value of a

Address (in bytes)	Stored Value
24399440	0
24399444	5
:	:
1458113424	1
1458113428	-2
:	:

value of pa

$*pa$, value pointed
to by pa

ex3.c:

```
#include <stdio.h>

int main() {
    int a = 0;
    int *pa;

    pa = &a;
    *pa = *pa + 1;

    printf("a = %d\n", a);
    printf("*pa = %d\n", *pa);
    printf("pa = %d\n", pa);
}
```

output:

```
~/Desktop> gcc ex3.c -o ex3
~/Desktop> ./ex3
a = 1
*pa = 1
pa = 1444219772
~/Desktop>
```

Address (in bytes)	Stored Value
1444219768	3
1444219772	1
1444219776	28
:	:

Value of pa,
also the address
of a

Value of a,
also pointed to
by pa

PASSING BY ARGUMENTS BY VALUE AND BY REFERENCE

ex4.c:

```
#include <stdio.h>

void fun( int a){
    a = a + 1;
}

int main(){
    int a = 0;

    fun( a );

    printf("a = %d\n", a);
}
```

output:

```
~/Desktop> gcc ex4.c -o ex4
~/Desktop> ./ex4
a = 0
~/Desktop>
```

a was passed to **fun()** by value.

PASSING ARGUMENTS BY REFERENCE

ex5.c:

```
#include <stdio.h>

void fun( int *a) {
    *a = *a + 1;
}

int main() {
    int a = 0;

    fun(&a);

    printf("a = %d\n", a);
}
```

output:

```
~/Desktop> gcc ex5.c -o ex5
~/Desktop> ./ex5
a = 1
~/Desktop>
```

a was passed to **fun()** by reference.

ex6.c:

```
#include <stdio.h>

void fun( int *a) {
    *a = *a + 1;
}

int main() {
    int a = 0, *pa;

    *pa = a;
    fun(pa);

    printf("a = %d\n", a);
    printf("*pa = %d\n", *pa);
}
```

output:

```
~/Desktop> gcc ex6.c -o ex6
~/Desktop> ./ex6
a = 0
*pa = 1
~/Desktop>
```

The value pointed to by **pa** was passed to **fun()** by reference, while **a** was left alone.

ex7.c:

```
#include <stdio.h>

void fun( int *a) {
    *a = *a + 1;
}

int main() {
    int a = 0, *pa;

    pa = &a;
    fun(pa);

    printf("a = %d\n", a);
    printf("*pa = %d\n", *pa);
}
```

output:

```
~/Desktop> gcc ex7.c -o ex7
~/Desktop> ./ex7
a = 1
*pa = 1
~/Desktop>
```

Since **pa** points to **a**, ***pa** and **a** are the same object. **a** was passed to **fun()** by reference.

ARRAYS

ar1.c:

```
#include <stdio.h>

void modify( int *a){
    int i;
    for( i = 0; i < 6; ++i){
        a[ i ] = i ;
    }
}

int main(){
    int i;
    int a[] = {1,23,17,4,-5,100};

    for( i = 0; i < 6; ++i){
        printf("a[%d] = %d\n", i , a[ i ] );
    }
    printf("\nModifying ... \n\n");

    modify( a );
}
```

```
for ( i = 0; i < 6; ++i ) {  
    printf( "a[%d] = %d\n" , i , a[ i ] ) ;  
}  
}
```

output:

```
~/Desktop> gcc ar1.c -o ar1  
~/Desktop> ./ar1  
a[0] = 1  
a[1] = 23  
a[2] = 17  
a[3] = 4  
a[4] = -5  
a[5] = 100
```

Modifying . . .

```
a[0] = 0  
a[1] = 1  
a[2] = 2  
a[3] = 3  
a[4] = 4  
a[5] = 5
```

ar2.c:

```
#include <stdio.h>

void modify( int *a){
    int i;
    for( i = 0; i < 6; ++i ){
        a[ i ] = i ;
    }
}

int main(){
    int i;
    int a[ 6 ];

    for( i = 0; i < 6; ++i ){
        a[ i ] = i * i + 1;
        printf("a[%d] = %d\n", i , a[ i ] );
    }
    printf("\nModifying . . . \n\n");

    modify( a );

    for( i = 0; i < 6; ++i ){
```

```
    printf( " a[%d] = %d\n" , i , a[ i ] ) ;  
}  
}
```

output:

```
~/Desktop> gcc ar2.c -o ar2  
~/Desktop> ./ar2  
a[0] = 1  
a[1] = 2  
a[2] = 5  
a[3] = 10  
a[4] = 17  
a[5] = 26
```

Modifying . . .

```
a[0] = 0  
a[1] = 1  
a[2] = 2  
a[3] = 3  
a[4] = 4  
a[5] = 5  
~/Desktop>
```

ar3.c:

```
#include <stdio.h>

void modify( int *a){
    int i;
    for( i = 0; i < 6; ++i ){
        *(a + i) = i;
    }
}

int main(){
    int i;
    int *a;

    for( i = 0; i < 6; ++i ){
        *(a + i) = i*i + 1;
        printf("*(a + %d) = %d\n", i, *(a + i));
    }
    printf("\nModifying...\\n\\n");

    modify(a);

    for( i = 0; i < 6; ++i ){
```

```
    printf( "*(a + %d) = %d\n", i, *(a + i));  
}  
}
```

output:

```
~/Desktop> gcc ar3.c -o ar3  
~/Desktop> ./ar3  
*(a + 0) = 1  
*(a + 1) = 2  
*(a + 2) = 5  
*(a + 3) = 10  
*(a + 4) = 17  
*(a + 5) = 26
```

Modifying . . .

```
*(a + 0) = 0  
*(a + 1) = 1  
*(a + 2) = 2  
*(a + 3) = 3  
*(a + 4) = 4  
*(a + 5) = 5  
~/Desktop>
```

ar4.c:

```
#include <stdio.h>

void modify( int *a){
    int i;
    for( i = 0; i < 6; ++i ){
        a[ i ] = i ;
    }
}

int main(){
    int i;
    int *a;

    for( i = 0; i < 6; ++i ){
        a[ i ] = i * i + 1;
        printf("a[%d] = %d\n", i , a[ i ] );
    }
    printf("\nModifying . . . \n\n");

    modify( a );

    for( i = 0; i < 6; ++i ){
```

```
    printf( " a[%d] = %d\n" , i , a[ i ] ) ;  
}  
}
```

output:

```
~/Desktop> gcc ar4.c -o ar4  
~/Desktop> ./ar4  
a[0] = 1  
a[1] = 2  
a[2] = 5  
a[3] = 10  
a[4] = 17  
a[5] = 26
```

Modifying . . .

```
a[0] = 0  
a[1] = 1  
a[2] = 2  
a[3] = 3  
a[4] = 4  
a[5] = 5  
~/Desktop>
```

CAUTION

Without dynamic allocation, assign `*a` by copying from named variables instead of using literals.

caution1.c:

```
int main() {
    int *a;
    *a = 0;
}
```

output:

```
~/Desktop> gcc caution1.c -o caution1
~/Desktop> ./caution1
Bus error: 10
~/Desktop>
```

The value of `a` is a memory address that does not exist!

CAUTION

Every (statically allocated) array has a set length. Do not reference beyond this length.

caution2.c:

```
#include <stdio.h>

int main() {
    int i = 0, *a;
    *a = i;
    printf("*a = %d\n", *a);

    *(a + 10000) = 1;
}
```

output:

```
~/Desktop> gcc caution2.c -o caution2
~/Desktop> ./caution2
*a = 0
Segmentation fault: 11
~/Desktop>
```

It is illegal to reference the object at memory address, (`a + 10000`).

DYNAMIC ALLOCATION

Static memory allocation: Acquiring a fixed, unchangeable piece of memory for a fixed-size data object at compile-time.

Dynamic memory allocation: Acquiring a variable-length piece of memory for an variable-size object during runtime.

For dynamic allocation, use the function `malloc()`, which is in `stdlib.h`.

dy1.c:

```
#include <stdio.h>
#include <stdlib.h>

void fill( int *a){
    int i;
    for( i = 0; i < 10; ++i ){
        a[ i ] = 10 + i*i;
    }
}

int main(){
    int i , *a;

    a = ( int * ) malloc( 10 * sizeof( int ) );
    fill( a );

    for( i = 0; i < 10; ++i ){
        printf( "a[%d] = %d\n" , i , a[ i ] );
    }

    free( a );
}
```

output:

```
~/Desktop> gcc dy1.c -o dy1
~/Desktop> ./dy1
a[0] = 10
a[1] = 11
a[2] = 14
a[3] = 19
a[4] = 26
a[5] = 35
a[6] = 46
a[7] = 59
a[8] = 74
a[9] = 91
~/Desktop>
```

dy2.c:

```
#include <stdio.h>
#include <stdlib.h>

#define M 10
#define N 15

void fill( float *x, int size){
    int i;
    for( i = 0; i < size; ++i){
        x[ i ] = 10.25 + i*i;
    }
}

int main(){
    int i;
    float *a, *b;

    a = ( float * ) malloc( M * sizeof( float ) );
    b = ( float * ) malloc( N * sizeof( float ) );

    fill( a, M );
    fill( b, N );
```

```
for ( i = 0; i < M; ++i ) {
    printf( "a[%d] = %f\n" , i , a[ i ] ) ;
}
printf( "\n" ) ;

for ( i = 0; i < N; ++i ) {
    printf( "b[%d] = %f\n" , i , b[ i ] ) ;
}

free( a );
free( b );
}
```

output:

```
~/Desktop> gcc dy2.c -o dy2
~/Desktop> ./dy2
a[0] = 10.250000
a[1] = 11.250000
a[2] = 14.250000
a[3] = 19.250000
a[4] = 26.250000
a[5] = 35.250000
a[6] = 46.250000
a[7] = 59.250000
a[8] = 74.250000
a[9] = 91.250000

b[0] = 10.250000
b[1] = 11.250000
b[2] = 14.250000
b[3] = 19.250000
b[4] = 26.250000
b[5] = 35.250000
b[6] = 46.250000
b[7] = 59.250000
b[8] = 74.250000
```

```
b[9] = 91.250000
b[10] = 110.250000
b[11] = 131.250000
b[12] = 154.250000
b[13] = 179.250000
b[14] = 206.250000
~/Desktop>
```

OUTLINE

- Pointers
- Arrays
- Bus errors and segmentation faults
- Dynamic allocation

LECTURE SERIES MATERIALS

These lecture slides, a tentative syllabus for the whole lecture series, and code are available at:

<https://github.com/wlandau/gpu>.

After logging into you home directory on impact1, type:

```
git clone https://github.com/wlandau/gpu
```

into the command line to download all the course materials.

REFERENCES

Ted Jensen. "A Tutorial on Pointers and Arrays in C".
<http://pw1.netcom.com/~tjensen/ptr/pointers.htm>

Brian W. Kernighan and Dennis M. Ritchie. "The ANSI C Programming Language". 2nd Ed.

Walter Savitch. "Absolute C++". 3rd Ed.