Pointers and dynamic allocation in C

Will Landau

Iowa State University

September 24, 2013

Outline

Computer memory

Pointers

Passing arguments by value and by reference

Arrays

Dynamic memory allocation

Outline

Computer memory

Pointers

Passing arguments by value and by reference

Arrays

Dynamic memory allocation

Computer memory

- Fundamentally, all data is encoded in *byte code*, strings of ones and zeros.

$$010010110010110010 1\cdots$$

- **Bit**: a 1 or 0 in byte code.
- **Byte**: a string of 8 bits. For example, 00110100.
- **Word**:
  - a natural unit of data, the length of which depends on the processor.
  - On "32-bit architectures", a word is a string of 32 bits (4 bytes).

Compute memory

- Computer memory is a linear array of bytes. Each byte has a word-sized index called an *address*, or *pointer*.

Address	Stored Value	Variable Name
24399440	3	a
24399441		
24399442		
24399443		
24399444	6.43451	b
24399445		
24399446		
24399447		
⋮	⋮	

- Note: we use the address, 24399440 (not 24399441 or 24399442) to refer to the storage space of variable a.

Computer memory

- I condense the previous table and write:

Address	Stored Value	Variable Name
24399440	3	a
24399444	6.43451	b
⋮	⋮	

- We say that:
  - 24399440 is the address of variable a.
  - 3 is the stored value at the address, 24399440.
  - a is the variable pointed to by 24399440.
  - 3 is the value pointed to by 24399440.

Outline

Declaring pointer variables

- Examples:
  - Write `int *pa;` to declare an int pointer variable: a variable whose value is the address of an integer.
  - Write `float *pa;` to declare a float pointer variable: a variable whose value is the address of a float.
  - Write `double *pa;` to declare a double pointer variable: a variable whose value is the address of a double.
- The type of a pointer variable depends on the data type pointed to because:
  - Different data types take up different amounts of memory.
  - The computer needs to know how to interpret the bytes of memory stored. Ints and floats, for example, are encoded differently.

Example

http://will-landau.com/gpu/Code/C/pointers/ex0.c

```c
#include <stdio.h>

int main(){
  int a = 17;

  printf("a = %d\n", a); // interpret as an int
  printf("a = %f\n", a); // interpret as a float
}
```

output

```
> gcc ex0.c -o ex0
> ./ex0
a = 17
a = 0.000000
```

http://will-landau.com/gpu/Code/C/pointers/ex1.c

```c
#include <stdio.h>

int main(){
  int a = 0;

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

output

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

Variable	Address	Stored value
a	1355533180	3

- Let a be an int and pa be a pointer to an int. Then:
    - &a returns the address of a (referencing).
    - *pa returns the value pointed to by a (dereferencing).

http://will-landau.com/gpu/Code/C/pointers/ex2.c

```
#include <stdio.h>

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

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

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

output

```
> gcc ex2.c -o ex2
> ./ex2
a=1
&a = 1420507900
?pa = 1
pa = 1420507900
&pa = 1420507888
```

Variable	Address	Stored value
a	1420507900	1
pa	1420507888	1420507900

http://will-landau.com/gpu/Code/C/pointers/ex3.c

```c
#include <stdio.h>

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

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

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

output

```
> gcc ex3.c ?o ex3
> ./ex3
 a=0
&a = 1537735420 b=1
&b = 1537735416 ?pa = 1
 pa = 1537735416
&pa = 1537735408
```

Variable	Address	Stored value
a	1537735420	0
b	1537735416	1
pa	1537735408	1537735416

Outline

Computer memory

Pointers

Passing arguments by value and by reference

Arrays

Dynamic memory allocation

Passing by value

http://will-landau.com/gpu/Code/C/pointers/ex4.c

```c
#include <stdio.h>

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

int main(){
  int a = 0;

  fun(a);

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

Passing by value

output

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

- `a` was passed to `fun()` by *value*
- `fun()` received a local copy of `a` and then lost it when the function call terminated.
- The copy of `a` in `int main()` remained unchanged.

Passing by reference

http://will-landau.com/gpu/Code/C/pointers/ex5.c

```c
#include <stdio.h>

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

int main(){
  int a = 0;

  fun(&a);

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

Passing by reference

output

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

- a was passed to `fun()` by *reference*
- `fun()` received a local copy of a *pointer* to a in `int main()`.
- When `fun()` terminated, it lost its copy of the address of a, but it did not have an actual copy of a to lose.

http://will-landau.com/gpu/Code/C/pointers/ex6.c

```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

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

- pa id not contain the address of a, so a was not passed at all.

http://will-landau.com/gpu/Code/C/pointers/ex7.c

```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

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

- Since pa points to a and pa was passed by value, a was passed by reference.

Caution

- Assign values to pointers before dereferencing them.

http://will-landau.com/gpu/Code/C/pointers/ex7.c

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

output

```
> gcc caution1.c ?o caution1
> ./caution1
Bus error: 10
```

- The value of a is some garbage number that isn't a real address! It points to nowhere!

Outline

Computer memory

Pointers

Passing arguments by value and by reference

Arrays

Dynamic memory allocation

Arrays

http://will-landau.com/gpu/Code/C/pointers/ar1.c

```c
#include <stdio.h>

int main(){
  int pa[] = {1,23,17}; // declare and initialize an array with 3 elements

  printf("%d\n", pa[0]); // prints the value 1
  printf("%d\n", pa[1]); // prints the value 23
  printf("%d\n", pa[2]); // prints the value 17
}
```

output

```
> gcc ar1.c -o ar1
> ./ar1
1
23
17
```

Arrays

http://will-landau.com/gpu/Code/C/pointers/ar2.c

```c
#include <stdio.h>

int main(){
  int i;
  int pa[4]; // declares an array with 4 elements

  pa[0] = 9; // assign values
  pa[1] = 17;
  pa[2] = 25;
  pa[3] = 7;

  printf("%d\n", pa[0]); // prints the value 9
  printf("%d\n", pa[1]); // prints the value 17
  printf("%d\n", pa[2]); // prints the value 25
  printf("%d\n", pa[3]); // prints the value 7
}
```

Arrays

output

```
> gcc ar2.c -o ar2
> ./ar2
9
17
25
7
```

Arrays

http://will-landau.com/gpu/Code/C/pointers/ar3.c

```c
#include <stdio.h>

int main(){
  int i;
  int pa[4]; // declares an array with 4 elements

  *pa = 9;          // same as pa[0] = 9
  *(pa + 1) = 17; // same as pa[1] = 17
  *(pa + 2) = 25; // same as pa[2] = 25
  *(pa + 3) = 7;   // same as pa[3] = 7

  printf("%d\n", *pa); // prints the value 9
  printf("%d\n", *(pa + 1)); // prints the value 17
  printf("%d\n", *(pa + 2)); // prints the value 25
  printf("%d\n", *(pa + 3)); // prints the value 7
  printf("pa = %d\n", pa);
}
```

Arrays

output

```
> gcc ar3.c -o ar3
> ./ar3
9
17
25
7
1518070576
```

- pa is actually pointer to the first element of the array.

Variable name(s)	Address	Stored value
pa		
pa[0], *pa	1518070576	9
pa[1], *(pa + 1)	1518070580	17
pa[2], *(pa + 2)	1518070584	25
pa[3], *(pa + 3)	1518070588	7

Caution

- Every (statically allocated) array has a set length. Do not dereference beyond this length.
- C lets you, but you risk a:
- bus error: dereferencing an address that points to nothing.
- segmentation fault: dereferencing an address that exists but that the program does not have permission to dereference (out of bounds).

http://will-landau.com/gpu/Code/C/pointers/caution2.c

```c
#include <stdio.h>

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

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

Caution

output

```
> gcc caution2.c -o caution2
> ./caution2
Segmentation fault: 11
```

Outline

Computer memory

Pointers

Passing arguments by value and by reference

Arrays

Dynamic memory allocation

Dynamic memory allocation

- **Static memory allocation**: acquiring a fixed-sized piece of memory for a variable at compile time.
- **Dynamic memory allocation**: acquiring a variable-length piece of memory at runtime.
- To use dynamic memory,
  1. use `malloc()`, defined in `stdlib.h`, to allocate memory.
  2. use the variable like an ordinary array.
  3. use `free()` to release the memory.

Dynamic memory allocation

http://will-landau.com/gpu/Code/C/pointers/dy1.c

```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);
}
```

Dynamic memory allocation

output

```
> gcc dy1.c -o dy
> ./dy
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
```

Dynamic memory allocation

http://will-landau.com/gpu/Code/C/pointers/dy2.c

```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;
  }
}
```

Dynamic memory allocation

http://will-landau.com/gpu/Code/C/pointers/dy2.c

```c
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);
}
```

Dynamic memory allocation

output

```
> gcc dy2.c -o dy2
> ./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
```

Dynamic memory allocation

output

```

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
```

Outline

Computer memory

Pointers

Passing arguments by value and by reference

Arrays

Dynamic memory allocation

Resources

1. Kernighan, B. W., and Ritchie, D. M. The ANSI C Programming Language. 2nd Ed.
2. Savitch, W. Absolute C++. 3rd Ed.
3. Jensen, T. A Tutorial on Pointers and Arrays in C. http://pw1.netcom.com/ tjensen/ptr/pointers.htm
4. GPU series materials: http://will-landau.com/gpu.