CSCI3150 – Tutorial 1

C REFRESHER: POINTERS

Calvin KAM, Ho Chuen (hckam@cse)
SHB 913

Agenda

1. Pointer Recap
   - Umm.. What is pointer?
2. Pointer Arithmetic
   - Yes! Pointer can be manipulated by addition and subtraction
3. Array of Pointers?

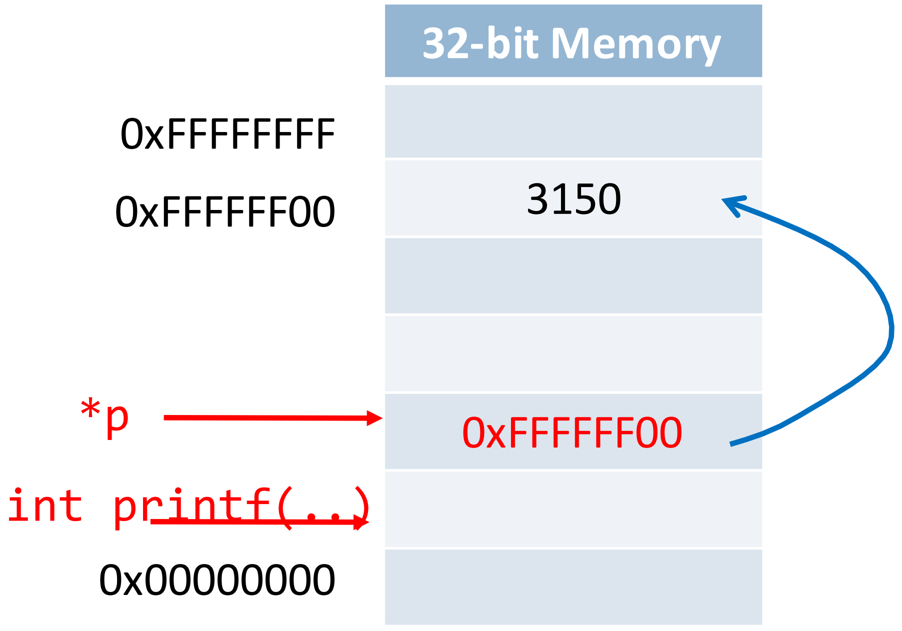
Hope you remember this :0

What is this?

```
#include <stdio.h>
 int main(int argc,char* argv[])
{
 char* msg = "Hello World";
 printf("%s\n",msg);
 return 0;
}
```

Pointer Recap

- In simple, pointer is an **address.**
- Everything in C has an address.
- The actual value is obtained by referring to another address by using its content.
- EXTRA: in 32-bit architecture, why is the range of memory from 0x00000000 to 0xFFFFFFFF?



Pointer Operators

```c
#include <stdio.h>

int main(int argc,char *argv[]){
int i = 0;
int *p = &i;

printf("Address of *p is : %p\n",p);
printf("Peek..The value is %d\n",*p);
// Changing i, What will happen?
i = 20;
printf("Peek Again, Value of *p: %d\n",*p);
// Changing *p will affect i?
*p = 3150;       printf("value of i: %d\n",i);
return 0;
}
```

Pointer Operators *

```
#include <stdio.h>

int main(int argc,char *argv[]){
int i = 0;
int *p = &i;

printf("Address of *p is : %p\n",p);
printf("Peek..The value is %d\n",*p);
// Changing i, What will happen?
i = 20;
printf("Peek Again, Value of *p: %d\n",*p);
// Changing *p will affect i?
*p = 3150;        printf("value of i: %d\n",i);
return 0;
}
```

Operator *
Declaring a pointer

Pointer Operators *

```
#include <stdio.h>

int main(int argc,char *argv[]){
int i = 0;
int *p = &i;

printf("Address of *p is : %p\n",p);
printf("Peek..The value is %d\n",*p);
// Changing i, What will happen?
i = 20;
printf("Peek Again, Value of *p: %d\n",*p);
// Changing *p will affect i?
*p = 3150;       printf("value of i: %d\n",i);
return 0;
}
```

Operator *
Declaring a pointer

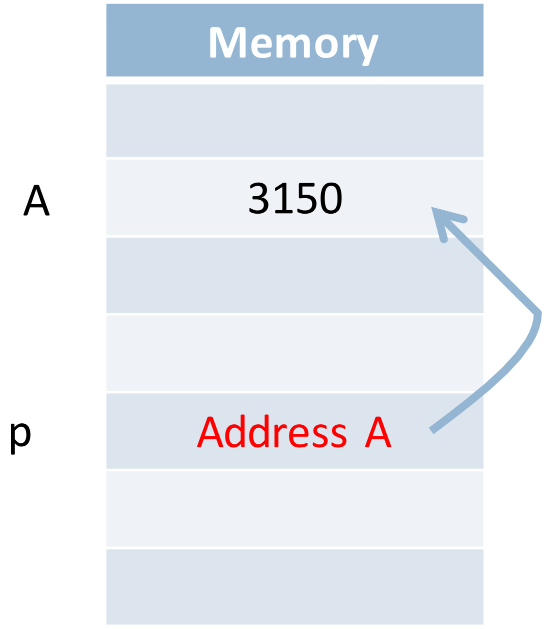
Operator *
Dereferencing

Dereferencing

```
#include <stdio.h>

int main(int argc,char *argv[]){
int i = 0;
int *p = &i;

printf("Address of *p is : %p\n",p);
printf("Peek..The value is %d\n",*p);
// Changing i, What will happen?
i = 20;
printf("Peek Again, Value of *p: %d\n",*p);
// Changing *p will affect i?
*p = 3150;       printf("value of i: %d\n",i);
return 0;
}
```



Dereferencing means to access the variable pointed by the pointer.

Pointer Operators &

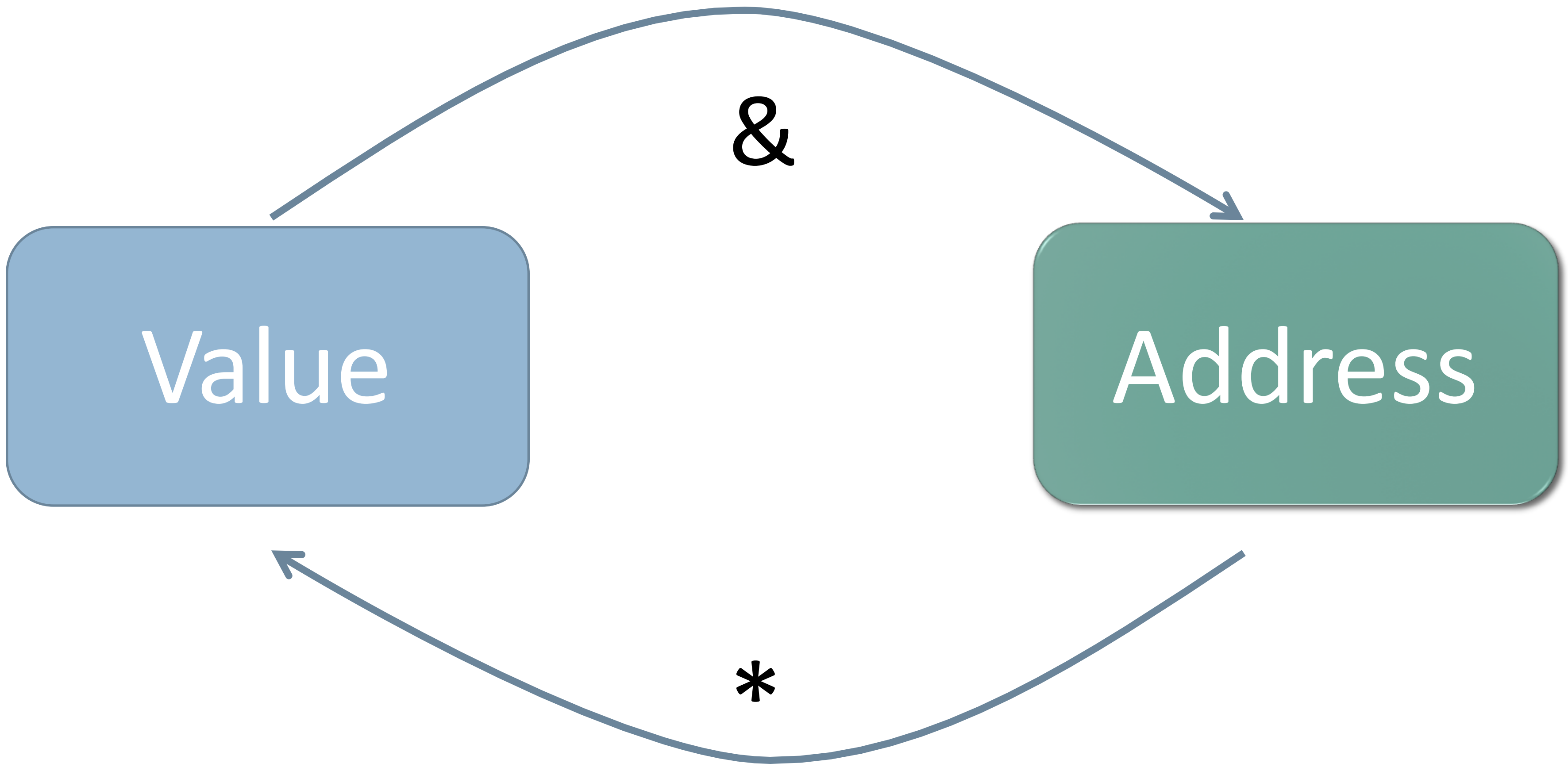
```c
#include <stdio.h>

int main(int argc,char *argv[]){
int i = 0;
int *p = &i;

printf("Address of *p is : %p\n",p);
printf("Peek..The value is %d\n",*p);
// Changing i, What will happen?
i = 20;
printf("Peek Again, Value of *p: %d\n",*p);
// Changing *p will affect i?
*p = 3150;        printf("value of i: %d\n",i);
return 0;
}
```

Operator &
Getting the address

Pointer Recap



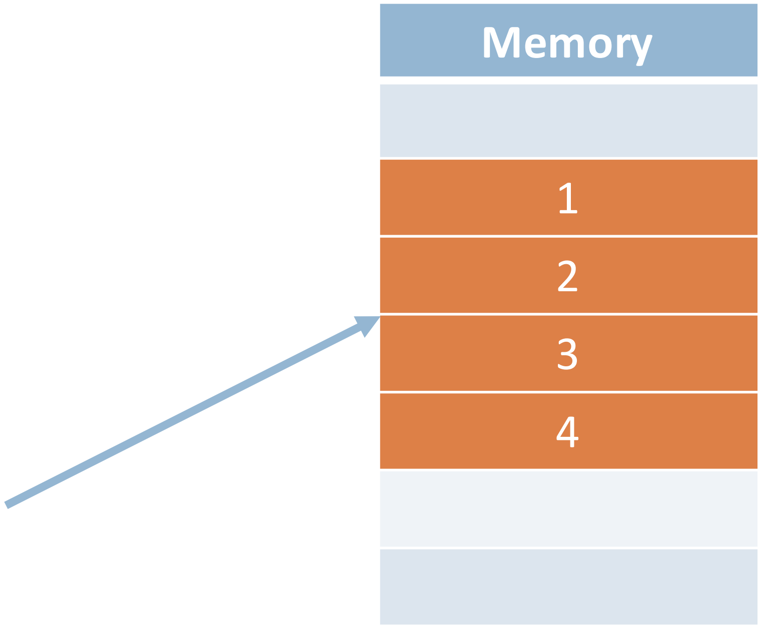
Pointer and Array

- In term of memory, a continuous range of addresses is allocated to an array.

```
int array[4];
```

- The name of array is actually a constant pointer to the first element.

*array is same as array[0]

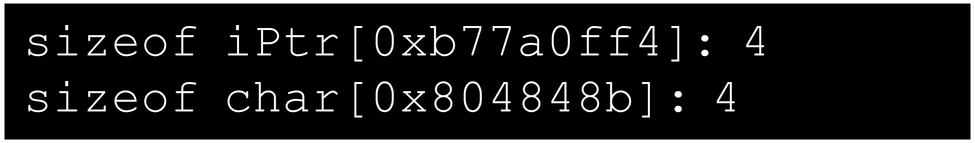


- Later we can know the usage A_A

Pointer Size?

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc,char *argv[])
{
        int *iPtr;
        char *cPtr;
        printf("sizeof iPtr[%p]: %d\n",iPtr,sizeof(iPtr));
        printf("sizeof char[%p]: %d\n",cPtr,sizeof(cPtr));
return 0;
}
```

Pointer Size?



```
sizeof iPtr[0xb77a0ff4]: 4
sizeof char[0x804848b]: 4
```

Every pointer are of the **<u>same size</u>**. Because they are only addresses :P

32-bit architecture: 4 Bytes.

64-bit architecture: 8 Bytes.

Pointer Arithmetic

We can do pointer arithmetic to manipulate pointers!!!

Generally there are two types: **Addition**, and **subtraction.**

Let's say if we add an integer to the pointer

Eg: `*(ptr + 4)`

What will happen?

Pointer Addition

```c
#include <stdio.h>
int main(int argc,char *argv[])
{
        char charArray[] = {'C','S','C','I'};
        int numArray[] = {3,1,5,0};
        int *nPtr = numArray;
        char *cPtr = charArray;
        // What is the current value of *nPtr?
        printf("Now CPtr(%p) : [%c]\n",cPtr,*cPtr);
        printf("Now nPtr(%p) : [%d]\n",nPtr,*nPtr);
        // Let's increment and look inside..
        printf("1 step forward..\n");
        nPtr++;cPtr++;
        printf("Now CPtr(%p) : [%c]\n",cPtr,*cPtr);
        printf("Now nPtr(%p) : [%d]\n",nPtr,*nPtr);
        return 0;
}
```

Advanced by 1 byte for **char** Pointer

Advanced by 4 bytes for **int** Pointer

```
Now CPtr(0xbfdb6264) : [C]
Now nPtr(0xbfdb6254) : [3]
1 step forward..
Now CPtr(0xbfdb6265) : [S]
Now nPtr(0xbfdb6258) : [1]
```

```c
#include <stdio.h>
int main(int argc,char *argv[])
{
        char charArray[] = {'C','S','C','I'};
        int numArray[] = {3,1,5,0};
        int *nPtr = numArray;
        char *cPtr = charArray;
        // What is the current value of *nPtr?
        printf("Now CPtr(%p) : [%c]\n",cPtr,*cPtr);
        printf("Now nPtr(%p) : [%d]\n",nPtr,*nPtr);
        // Let's increment and look inside..
        printf("1 step forward..\n");
        nPtr++;cPtr++;
        printf("Now CPtr(%p) : [%c]\n",cPtr,*cPtr);
        printf("Now nPtr(%p) : [%d]\n",nPtr,*nPtr);
        return 0;
}
```

Incrementing the pointer will advance it by the size of data type it points to.

Pointer Subtraction

```c
#include <stdio.h>
#define SIZE 4
int main(int argc,char *argv[]) {
        char charArray[] = {'C','S','C','I'};
        int numArray[] = {3,1,5,0};
        int *nPtr = numArray;
        char *cPtr = charArray;
        // What is the current value of *nPtr?
        printf("Now CPtr(%p) : [%c]\n",cPtr,*cPtr);
        printf("Now nPtr(%p) : [%d]\n",nPtr,*nPtr);
        // Move it Move it
        nPtr += SIZE;
        cPtr += SIZE;
        // What is the meaning of subtraction?
        printf("nPtr-numArray: [%d]\n",nPtr-numArray);
        printf("cPtr-charArray: [%d]\n",cPtr-charArray);
        return 0;
}
```

Pointer Subtraction

```c
#include <stdio.h>
#define SIZE 4
int main(int argc,char *argv[]) {
        char charArray[] = {'C','S','C','I'};
        int numArray[] = {3,1,5,0};
        int *nPtr = numArray;
        char *cPtr = charArray;
        // What is the current value of *nPtr?
        printf("Now CPtr(%p) : [%c]\n",cPtr,*cPtr);
        printf("Now nPtr(%p) : [%d]\n",nPtr,*nPtr);
        // Move it Move it
        nPtr += SIZE;
        cPtr += SIZE;
        // What is the meaning of subtraction?
        printf("nPtr-numArray: [%d]\n",nPtr-
numArray);
        printf("cPtr-charArray: [%d]\n",cPtr-
charArray);
        return 0;
}
```

```
Now CPtr(0xffb6c0d4) : [C]
Now nPtr(0xffb6c0c4) : [3]
nPtr-numArray: [4]
cPtr-charArray: [4]
```

Subtraction of two pointers in an array returns the **number of elements.**

IT IS NOT AN ABSOLUTE DISTANCE!!

Array of Pointers

Still remember this? :D

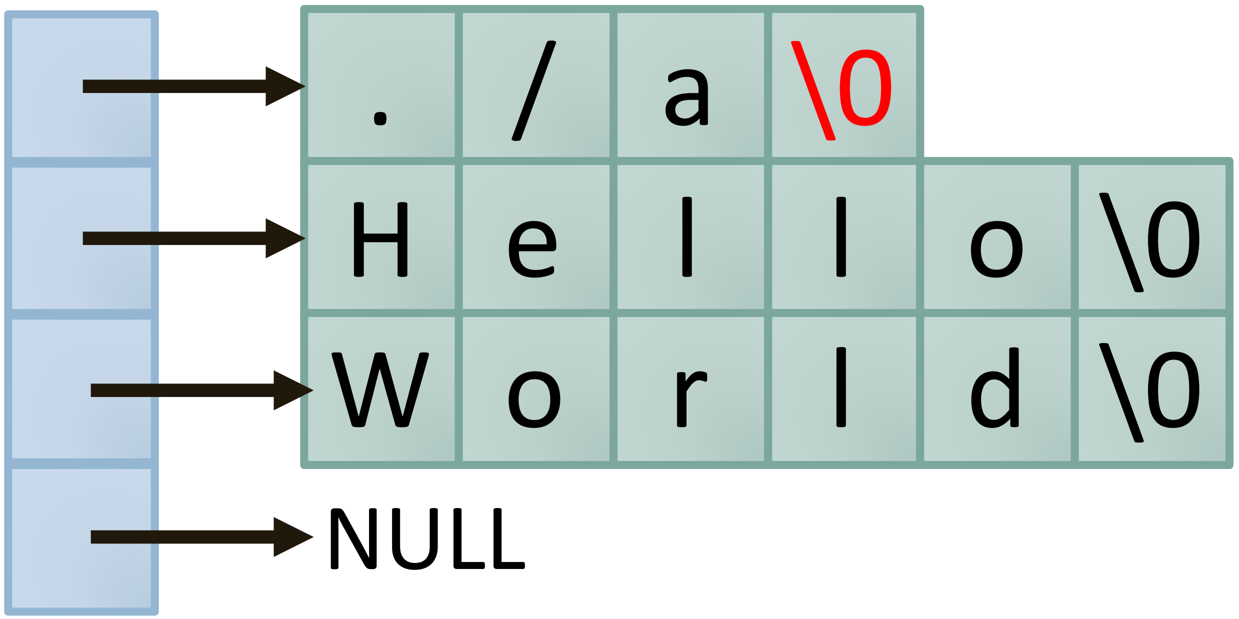
Command Line Argument

```
#include <stdio.h>

int main(int argc,char *argv[]){
   int i;
   for(i = 0;i < argc;i++)
   printf("[%d]:%s\n",i,argv[i]);
   return 0;
}
```

Array of Pointers

Actually char* argv[] is **Array of Character Pointers.**



```
$ ./a Hello World
```

Array vs Array of Pointer?

`char argv[]`

- Just a simple character array.

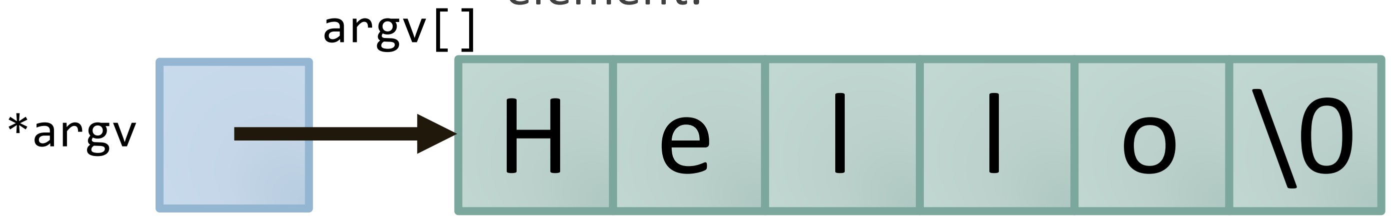
`char *argv[]`

- This declares **argv** as an array of char pointers.
- First Dereferencing :

```
*argv == arg[0]
```

What is inside arg[0]?

- Also pointer (address)! With type char* and pointing to the 1st element.



Summary

1. Pointer is an address referring to a particular place in memory.
2. Pointers has two related operators: * , &.
   1. * declares a pointer or performs dereferencing.
   2. & gets the address of the object.
3. All types of pointers are in same size, depending on CPU architecture.
4. Arithmetic can be done to pointers. By moving the pointers we can manipulate with the objects or getting the number of elements.
5. *argv[] declares an array **argv** containing character pointers which points to the first element of array.

MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.

Source: xkcd.com

Ready to play (or being played by) pointer?

See you!