CS F111: Computer Programming

(Second Semester 2021-22)

Lect 25: Pointers contd.

Nikumani Choudhury

Asst. Professor, Dept. of Computer Sc. & Information System

BITS Pilani

Hyderabad Campus

Function Pointers

- We can have a concept of Pointer to a function known as **function pointer** in C

- function_return_type(*Pointer_name)(function argument list)

```
#include <stdio.h>
int sum (int num1, int num2) {
    return num1 + num2;
}

int main() {
    int (*f2p) (int, int);
    f2p = sum;

    int op1 = f2p (25, 50);
    int op2 = sum (25, 50);
```

```
printf("Output1: Call using function pointer: %d", op1);
printf("\nOutput2: Call using function name: %d", op2);
return 0;
}
```

```
Output1: Call using function pointer: 75
Output2: Call using function name: 75
```

```c
#include <stdio.h>
// A normal function with an int parameter
// and void return type
void fun(int a)
{
    printf("Value of a is %d\n", a);
}

int main()
{
    // fun_ptr is a pointer to function fun()
    void (*fun_ptr)(int) = &fun;

    /* The above line is equivalent of following two
    void (*fun_ptr)(int);
    fun_ptr = &fun;
    */

    // Invoking fun() using fun_ptr
    (*fun_ptr)(10);

    return 0;
}
```

```c
#include <stdio.h>
// A normal function with an int parameter
// and void return type
void fun(int a)
{
    printf("Value of a is %d\n", a);
}

int main()
{
    void (*fun_ptr)(int) = fun; // & removed

    fun_ptr(10); // * removed

    return 0;
}
```

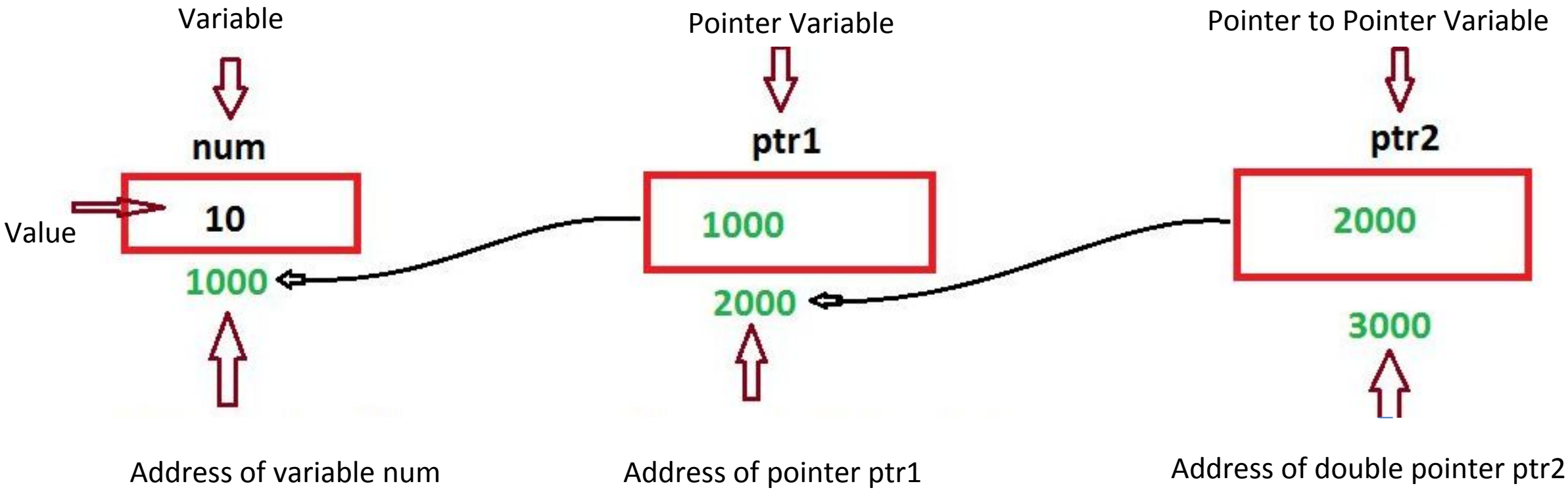
String sort example

```
#include <stdio.h>
#include <string.h>
void stringsort(char *strings[], int nstrings, int (*cmpfunc())   {
int i, j;  int flag;
do {
     flag = 0;
     for(i = 0; i < nstrings - 1; i++)  {
       j = i + 1;
       if ( (*cmpfunc)(strings[i], strings[j]) > 0)  {
          char *tmp = strings[i];
          strings[i] = strings[j];
          strings[j] = tmp;
          flag = 1; }
      }
  } while(flag);
}
```

```
int main()  {
  char *array [] = {"Raj", "Amit", "Mohan",
                                          "Vijay"};
  stringsort(array, 4, strcmp);

  for (int i=0; i < 4; i++)
     printf ("%s\n", array[i]);
return 0;
}
```

```
Amit    Mohan   Raj
Vijay
```

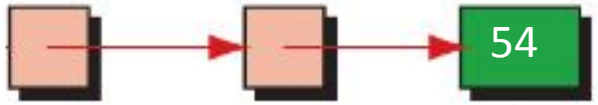
Pointers to Pointers



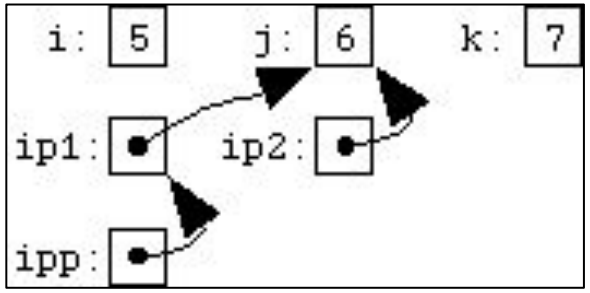
Double pointers example

```
#include <stdio.h>
int main(){
  int num = 54;
  int *ptr1 = &num;
  int **ptr2 = &ptr1;
  printf("%d", num);
  printf("\n%d", *ptr1);
  printf("\n%d", **ptr2);
  return 0;
}
```

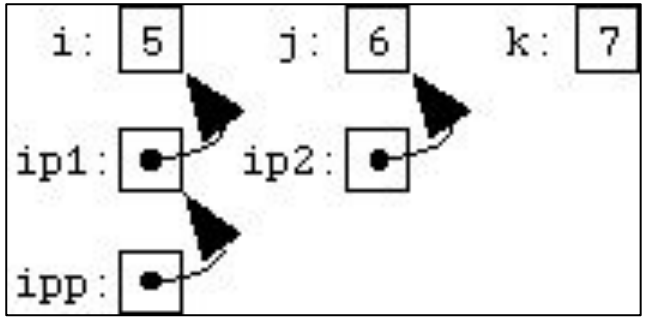
```
54    54
54
```



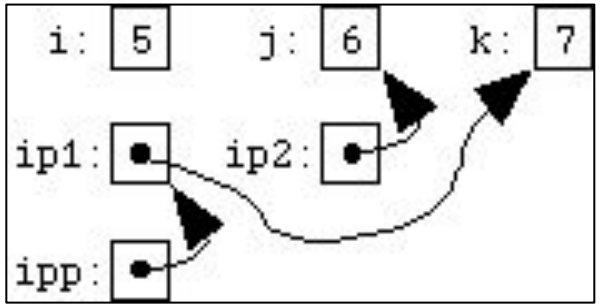
ptr2 ptr1 num



*ipp = ip2;



int i = 5, j = 6; k = 7; int *ip1 = &i, *ip2 = &j; ipp = &ip1;



*ipp = &k;

Pointer Compatibility

- Pointers are not just pointer types, these are pointers to specific types of data items. Hence, these exhibit specific characteristics based on the type of data.
- Size Compatibility: Size of all pointers is the same

```
#include <stdio.h>
int main(){
   int a;     char b;      float c;     double d;

   int *p;    char *q;    float *r;    double *s;

   printf("%ld \t %ld \t %ld \t %ld", sizeof(a), sizeof(b), sizeof(c), sizeof(d));
   printf("\n%ld \t %ld \t %ld \t %ld", sizeof(*p), sizeof(*q), sizeof(*r), sizeof(*s));

   printf("\n\n%ld \t %ld \t %ld \t %ld", sizeof(p), sizeof(q), sizeof(r), sizeof(s));
   return 0;
}
```

```
4       1       4       8
4       1       4       8

8       8       8       8
```

Continued…

- Dereference type Compatibility: Dereferencing type is the type of the variable that the pointer is referencing. A pointer to an int is only compatible with a pointer to an int. Likewise, a ptr to char.

```
#include <stdio.h>
int main(){
   int a;
   int *p = &a;
   char b;
   char *q = &b;
   q = &a;
   p = &b;
   printf("Hello World");
   return 0;
}
```

```
q = (char *) &a;
p = (int *) &b;
```

(Solution: Casting pointers)

```
int *p; int **q;
q = p;
```

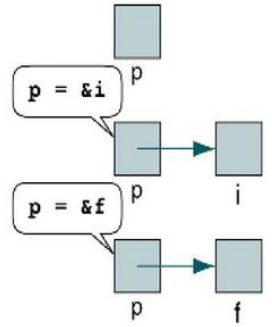
(Deref. level incompatibility)

```
int *p;    void *q;
q=p;
```

(No error with void) *q=p (No)

```
main.c:7:7: warning: assignment from incor
ompatible-pointertypes]
main.c:8:7: warning: assignment from incor
ompatible-pointertypes]
Hello World
```

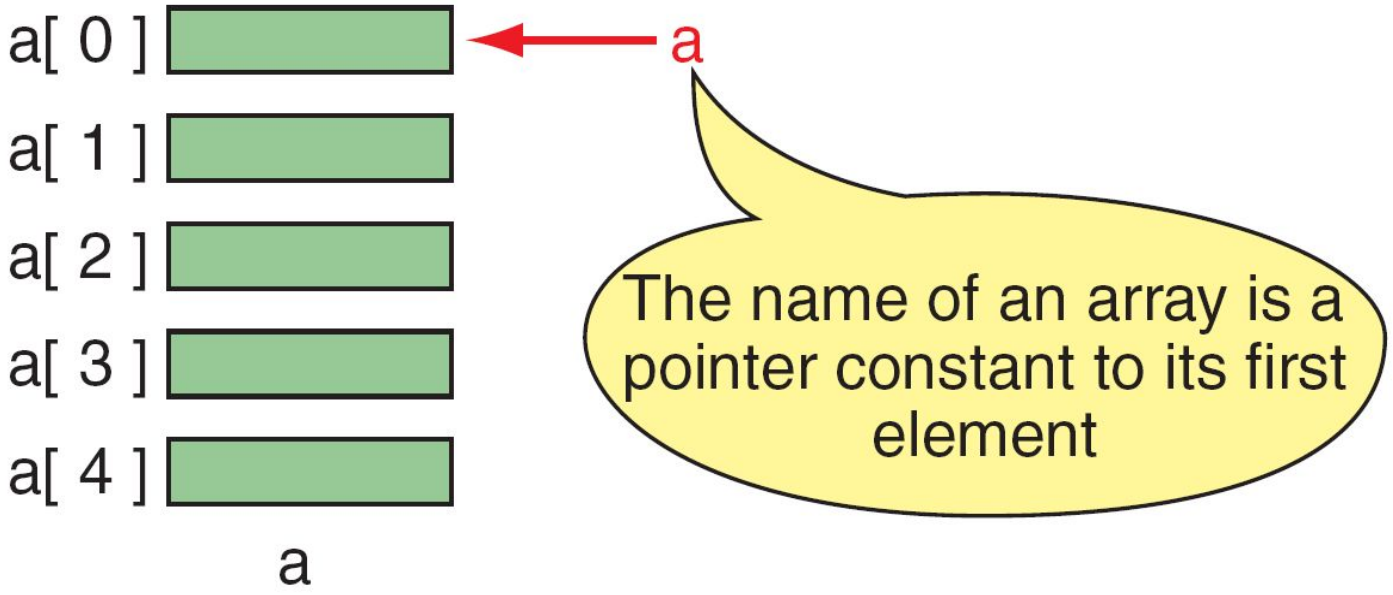
```
void* p;
int   i;
float f;

p = &i;
...
p = &f;
```



Pointer Applications: Arrays & Pointers

- Because the array's name (a) is a pointer constant, its value cannot be changed. The name 'a' is only a pointer to the first element and not the whole array.

&a[0] ⟷ a



An Example

```
#include <stdio.h>
int main() {
   int x [5];
   int i;
   for(i = 0; i < 5; ++i) {
      printf("Address of x[%d] = %p\n", i, &x[i]);
   }
   printf("Address of array x: %p", x);
return 0;
}
```

```
Address of x[0] = 0x7fff6119c5a0
Address of x[1] = 0x7fff6119c5a4
Address of x[2] = 0x7fff6119c5a8
Address of x[3] = 0x7fff6119c5ac
Address of x[4] = 0x7fff6119c5b0
Address of array x: 0x7fff6119c5a0
```

Output??

```c
#include <stdio.h>

int main()
{
    int *ptr;
    int x;

    ptr = &x;
    *ptr = 0;

    printf(" x = %dn", x);
    printf(" *ptr = %dn", *ptr);

    *ptr += 5;
    printf(" x  = %dn", x);
    printf(" *ptr = %dn", *ptr);

    (*ptr)++;
    printf(" x = %dn", x);
    printf(" *ptr = %dn", *ptr);

    return 0;
}
```

```c
#include <stdio.h>

int main()
{
    float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
    float *ptr1 = &arr[0];
    float *ptr2 = ptr1 + 3;

    printf("%f ", *ptr2);
    printf("%d", ptr2 - ptr1);
   return 0;
}
```

```c
#include<stdio.h>
int main()
{
 char *ptr = "BITS Pilani";
 printf("%c", *&*&*ptr);
 return 0;
}
```

```c
#include<stdio.h>
int main()
{
    int arr[] = {10, 20, 30, 40, 50, 60};
    int *ptr1 = arr;
    int *ptr2 = arr + 5;
    printf("Number of elements between two pointer are: %d.\n",(ptr2 - ptr1));
    printf("Number of bytes between two pointers are: %d",(char*)ptr2 - (char*) ptr1);
    return 0;
}
```