C under Linux

Dr. Naeem Odat

Department of Computer and Communications Engineering

C - Pointers - More on Pointers

C - Pointers - More on Pointers

C - Pointers - More on Pointers

Array of Pointers

- Number of Rows are fixed, whereas the number of columns is varying.
- Example:Imagine, the number of students is known beforehand and each student can select the subjects for the exam to be taken. In this type of system, each student can have a number of subjects.
- Declaring an Array of Pointer:

  ```
  int *arr[ROWSIZE]; // can be written as int *(arr[ROWSIZE]);
  ```
- Number of mallocs and frees is **ROWSIZE**.

C - Pointers - More on Pointers

Array of Pointers

```
#include<stdio.h>
#include<stdlib.h>
#define ROW 3
#define COL1 1
int main(void){
    int nCnt, *pArr[ROW];
    for(nCnt = 0; nCnt < ROW; nCnt++)
        pArr[nCnt] = (int*) malloc ( Col1 * sizeof(int));
    ...
    for(nCnt = 0; nCnt < ROW; nCnt++)
        free(pArr[nCnt]);/*Do an explicit free for each malloc *
    return(0);
}
```

C - Pointers - More on Pointers

Pointer to Array

- Number of rows is varying, and the number of columns is fixed.
- Example, imagine, the number of subjects to be taken by each students is fixed as in the case of schools, but the number of students appearing in the exam is not known and its varying.
- Declaring a pointer to array:

  ```
  int (*arr)[COLSIZE];
  ```
- The number of mallocs and frees is **One**.

C - Pointers - More on Pointers

Pointer to Array

```
#include<stdio.h>
#include<stdlib.h>
#define ROW 3
#define COL 3
int main(void){
    int (*pArr)[COL];
    int i, j;
    pArr = (int(*)[COL]) malloc (sizeof(int[COL]) * ROW);
    ...
    free(pArr);
    return(0);
}
```

C - Pointers - More on Pointers

Pointer to Pointer

- Both number of rows and number of columns are varying.
- Advantage of having jagged array, where the number of columns in each row will be different.
- Most flexible, both the row size and the column size are specified at run-time.
- The number of mallocs and frees is **number of columns + 1**.

C - Pointers - More on Pointers

Pointer to Pointer - Example

```
#include<stdio.h>
#include>stdlib.h>
int main(){
    int**ptr;
    ptr=(int**)malloc(5*sizeof(int*));
    for(int i=0;i<5;i++){
        ptr[i]=(int*)malloc(3*sizeof(int));
        ptr[i][0]=20+i;
        ptr[i][1]=40+i;
        ptr[i][2]=50+i;
    }
    //To free
    for(int i=0;i<5;i++)
        free(ptr[i]);

    free(ptr);
}
```

C - Pointers - More on Pointers

Memory Allocation for Structure

```
typedef struct{
    int a;
    char b;
    float f[3];
} some_struct;

some_struct *ssa2 = (some_struct*) malloc(25*sizeof(some_struct)

ssa2[13].b = 'g';
ssa2[9].f[2] = 24.6;
...
free(ssa2);
```

C - Pointers - More on Pointers

Pointer to Structure

- Example:

```
struct some_struct{
    int a;
    float f[3];
} ss;
struct some_struct *sp;
sp = &ss;
```

- To reference an element of sp:
  - $(*sp).a = 12;$
  - $sp - > a = 12;$

C - Pointers - More on Pointers

What can be passed to the following function?

```
#include<stdio.h>
void F(int* pA){
    int nCtr;
    for (nCtr = 0; nCtr< 5; nCtr++)
        *(pA+nCtr) = ...;

    ...
}
```

An array or pointer to an array:

```
int arr[] = {1,2,3,4,5,6};
int* p = arr;
F(arr);//or F(p);
```

C - Pointers - More on Pointers

Which Variables Can Be Returned by the Function?

```
#include<stdlib.h>
int nGlobal = 10;
int* f(int);
int main(void){
    int nMainVar = 10, * p = NULL;
    p = f(nMainVar);
    return(0);
}
int* f(int nVar1){
    int nVar2 [10];
    int *pVar3 = malloc(2 *sizeof(int));//*
    ...
    return (...);
}
```

C - Pointers - More on Pointers

Pass By Value

```
void Swap(int nP1, int nP2){
    int temp;
    temp = nP1;
    nP1 = nP2;
    nP2 = temp;
}
int main(){
    int a = 20, b = 30;
    Swap(a,b);
}
```

C - Pointers - More on Pointers

Pass By Reference (pointer)

```
void Swap(int* nP1, int* nP2){
    int temp;
    temp = *nP1;
    *nP1 = *nP2;
    *nP2 = temp;
}
int main(){
    int a = 20, b = 30;
    Swap(&a, &b);
}
```

C - Pointers - More on Pointers

Pointer to function

- C does not require that pointers point only to data; it is also possible to have pointers to functions.
- After all, functions occupy memory locations, so start of every function has an address, just like each variable has an address.
- Function pointers as Arguments:
  - We can use pointers to function in much the same way as pointers to data.
  - Passing a function pointer as an argument is fairly common in C.

C - Pointers - More on Pointers

Pointer to function

```
int Sum(int a, int b){
    return(a + b);
}
int Call(int a, int b, int (*f)(int, int)){
//int Call(int a, int b, int f(int, int)){
    return( (*f)(a, b));
}
int main(void){
    printf("%d\n", Call(10, 20, Sum));
    return(0);
}
```

C - Pointers - More on Pointers

Pointer to function - Example

```
#include<stdio.h>
int Sum(int, int);
int Subtract(int, int);
int main(){
    int nChoice, nA = 40, nB = 20;
    int (*Call[2])(int, int) = { Sum, Subtract };
    printf("\nEnter 0 for Sum or 1 for Sub :");
    scanf("%d", &nChoice);
    printf("%d\n" , (*Call[nChoice])(nA,nB));
    return 0;
}
int Sum(int a, int b){
    return (a+b);
}
int Subtract(int a, int b){
    return(a-b);
}
```