UNIVERSITY OF WOLVERHAMPTON

6CS005 High Performance Computing

Lecture 1 – Part 2

Pointers and Dynamic Memory Allocation in C

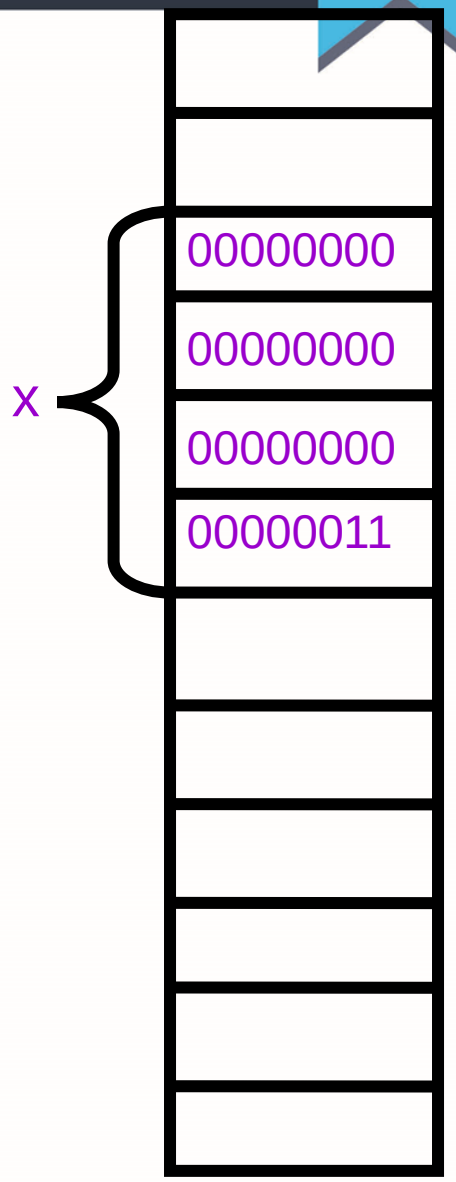
Jnaneshwar Bohara

Topics Covered

- **Pointer Fundamentals**
- **Initializing Pointers**
- **Using Pointers**
- **Pointers as Function Parameters**
- **Pointer Arithmetic**
- **Pointers and Arrays**
- **Using Pointers to Access Array Elements**
- **Dynamic memory allocation**

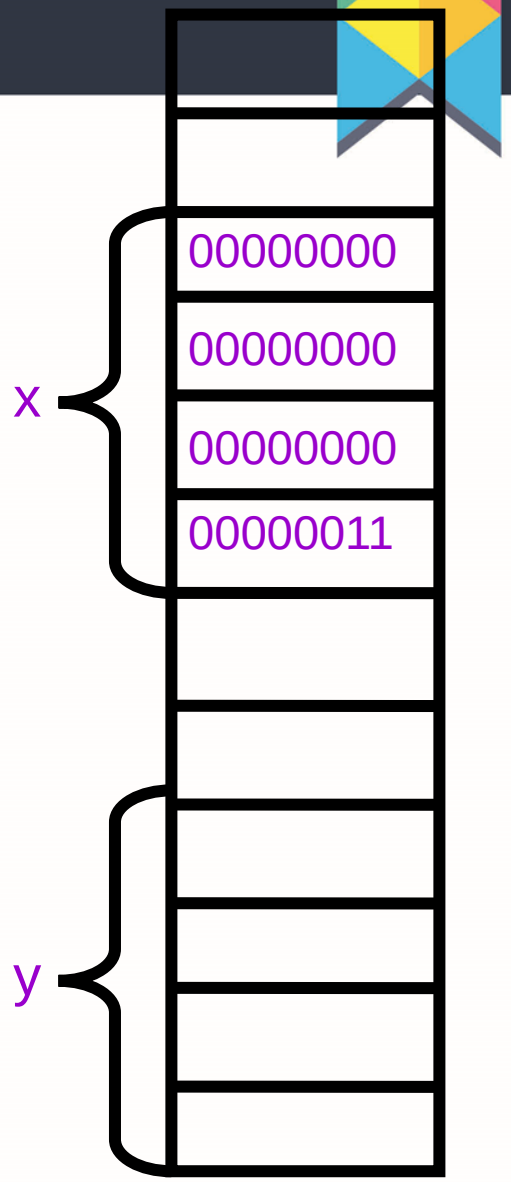
Pointer Fundamentals

- When a variable is defined the compiler (linker/loader actually) allocates a real memory address for the variable.
  - int x; will allocate 4 bytes in the main memory, which will be used to store an integer value.
- When a value is assigned to a variable, the value is actually placed to the memory that was allocated.
  - x=3; will store integer 3 in the 4 bytes of memory.



Pointers

- When the value of a variable is used, the contents in the memory are used.
  - y=x; will read the contents in the 4 bytes of memory, and then assign it to variable y.
- &x can get the address of x. (referencing operator &)
- The address can be passed to a function:
  - scanf("%d", &x);
- The address can also be stored in a variable ……



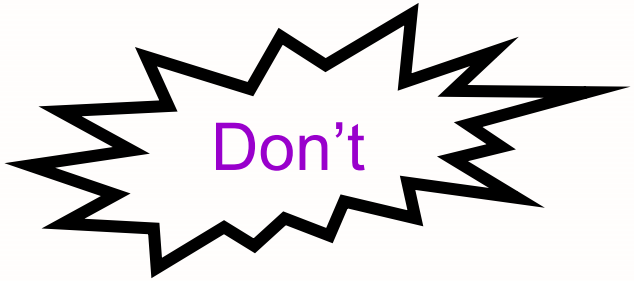
Pointers

- To declare a pointer variable
  - type * pointername;
- For example:
  - int * p1; p1 is a variable that tends to point to an integer, (or p1 is a int pointer)
  - char *p2;
  - unsigned int * p3;
- p1 = &x; /* Store the address in p1 */
- scanf("%d", p1); /* i.e. scanf("%d",&x); */
- p2 = &x; /* Will get warning message */

Initializing Pointers

- Like other variables, always initialize pointers before using them!!!
- For example:

```
int main(){
    int x;
    int *p;
    scanf("%d",p);   /*
    p = &x;
    scanf("%d",p);  /* Correct */
}
```




```
*/
```

Using Pointers

- You can use pointers to access the values of other variables, *i.e.* the contents of the memory for other variables.
- To do this, use the * operator (dereferencing operator).
  - Depending on different context, * has different meanings.
- For example:
  - int n, m=3, *p;
  - p=&m;
  - n=*p;
  - printf("%d\n", n);
  - printf("%d\n",*p);

An Example

```
int m=3, n=100, *p;
p=&m;
printf("m is %d\n",*p);
m++;
printf("now m is %d\n",*p);
p=&n;
printf("n is %d\n",*p);
*p=500;     /* *p is at the left of "="  */
printf("now n is %d\n", n);
```

Pointers as Function Parameters

- Sometimes, you want a function to assign a value to a variable.
  - e.g. scanf()
- E.g. you want a function that computes the minimum AND maximum numbers in 2 integers.
- Method 1, use two global variables.
  - In the function, assign the minimum and maximum numbers to the two global variables.
  - When the function returns, the calling function can read the minimum and maximum numbers from the two global variables.
- This is bad because the function is not reusable.

Pointers as Function Parameters

- Instead, we use the following function

```
void min_max(int a, int b,
        int *min, int *max){
        if(a>b){
                *max=a;
                *min=b;
        }
    else{
                *max=b;
                *min=a;
        }
}
```

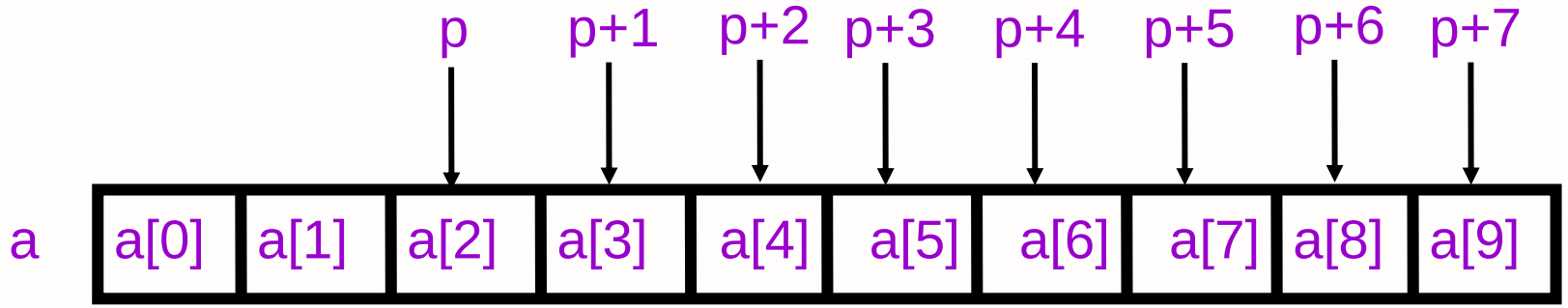
```
int main()
{
        int x,y;
        int small,big;
        printf("Two integers: ");
        scanf("%d %d", &x, &y);
        min_max(x,y,&small,&big);
        printf("%d <= %d", small, big);
        return 0;
}
```

Pointer Arithmetic

When a pointer variable points to an array element, there is a notion of adding or subtracting an integer to/from the pointer elements

```
int a[ 10 ], *p;

p = &a[2];

*p = 10;

*(p+1) = 10;

printf("%d", *(p+3));
```

```
int a[ 10 ], *p;


a[2] = 10;

a[3] = 10;

printf("%d", a[5]);
```



Pointer Arithmetic

- More examples:

```
int a[10], *p, *q;
p = &a[2];
q = p + 3;          /* q points to a[5] now */
p = q – 1;          /*  p points to a[4] now */
p++;                /*  p points to a[5] now */
p--;                /* p points to a[4] now */
*p = 123;           /*  a[4] = 123 */
*q = *p;            /*   a[5] = a[4]   */
q = p;              /*   q points to a[4] now */
scanf("%d", q)     /* scanf("%d", &a[4])  */
```

Pointers and Arrays

- Recall that the value of an array name is also an address.
- In fact, pointers and array names can be used interchangeably in many (but not all) cases.

E.g. int n, *p; p=&n;

n=1; *p = 1; p[0] = 1;

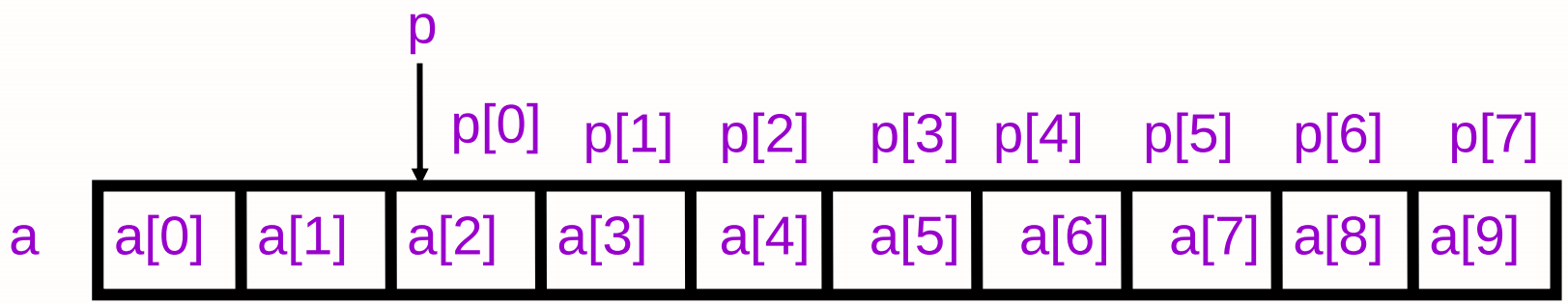
- The major differences are:
  - Array names come with valid spaces where they “point” to. And you cannot “point” the names to other places.
  - Pointers do not point to valid space when they are created. You have to point them to some valid space (initialization).

Using Pointers to Access Array Elements

- A pointer can be used just like an array

```
int a[ 10 ], *p;
p = &a[2];
p[0] = 10;
p[1] = 10;
printf("%d", p[3]);
```

```
int a[ 10 ], *p;

a[2] = 10;
a[3] = 10;
printf("%d", a[5]);
```



An Array Name is Like a Constant Pointer

- Array name is like a constant pointer which points to the first element of the array.

```
int a[10], *p, *q;
p = a;          /* p = &a[0] */
q = a + 3;      /* q = &a[0] + 3 */
a ++;           /*  illegal !!! */
```

- Therefore, you can “pass an array” to a function. Actually, the address of the first element is passed.

```
int a[ ] = { 5, 7, 8 , 2, 3 };
sum( a, 5 );  /* Equal to sum(&a[0],5)  */
……
```

UNIVERSITY OF WOLVERHAMPTON

Dynamic Memory Allocation

Problem with Arrays

- Sometimes
  - Amount of data cannot be predicted beforehand
  - Number of data items keeps changing during program execution

- Example: Search for an element in an array of N elements

- One solution: find the maximum possible value of N and allocate an array of N elements
  - Wasteful of memory space, as N may be much smaller in some executions
  - Example: maximum value of N may be 10,000, but a particular run may need to search only among 100 elements
    - Using array of size 10,000 always wastes memory in most cases

Better Solution

- Dynamic memory allocation
  - Know how much memory is needed after the program is run
    - Example: ask the user to enter from keyboard
  - Dynamically allocate only the amount of memory needed
- C provides functions to dynamically allocate memory
  - malloc, calloc, realloc

Memory Allocation Functions

- malloc
  - Allocates requested number of bytes and returns a pointer to the first byte of the allocated space
- calloc
  - Allocates space for an array of elements, initializes them to zero and then returns a pointer to the memory.
- free
  - Frees previously allocated space.
- realloc
  - Modifies the size of previously allocated space.
- We will only do malloc and free

Allocating a Block of Memory

- A block of memory can be allocated using the function malloc
  - Reserves a block of memory of specified size and returns a pointer of type void
  - The return pointer can be type-casted to any pointer type

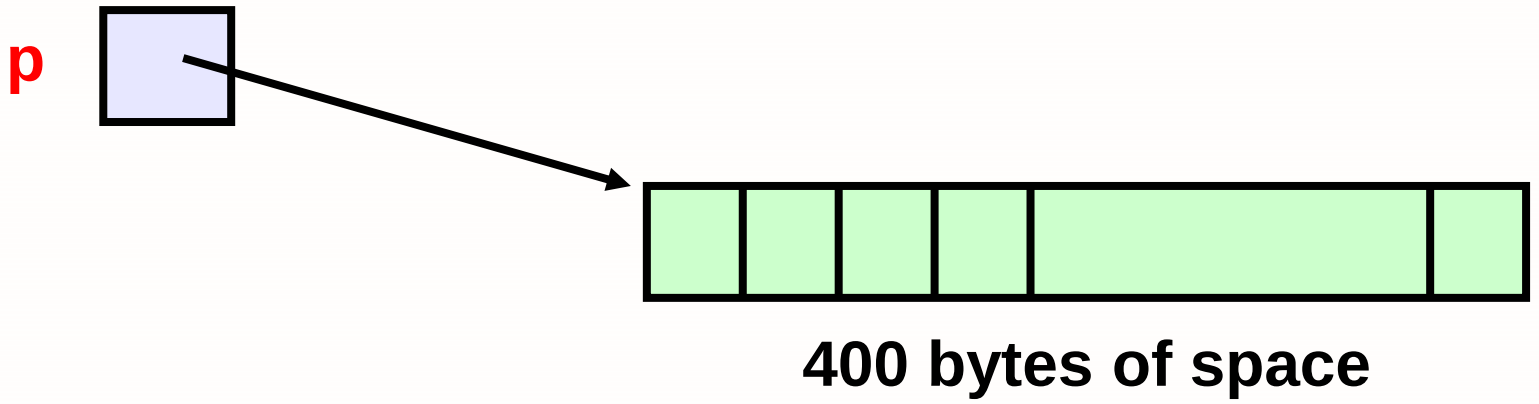
- General format:

```
type *p;
p =  (type *) malloc (byte_size);
```

Example

```
p = (int *) malloc(100 * sizeof(int));
```

- A memory space equivalent to 100 times the size of an int bytes is reserved
- The address of the first byte of the allocated memory is assigned to the pointer p of type int



Example

- cptr = (char *) malloc (20);

  Allocates 20 bytes of space for the pointer cptr of type char

- sptr = (struct stud *) malloc(10*sizeof(struct stud));

  Allocates space for a structure array of 10 elements. sptr points to a structure element of type struct stud

Always use sizeof operator to find number of bytes for a data type, as it can vary from machine to machine

Points to Note

- malloc always allocates a block of contiguous bytes
  - The allocation can fail if sufficient contiguous memory space is not available
  - If it fails, malloc returns NULL

```
if  ((p = (int *) malloc(100 * sizeof(int))) == NULL)
  {
      printf ("\n Memory cannot be allocated");
      exit();
   }
```

Using the malloc'd Array

- Once the memory is allocated, it can be used with pointers, or with array notation
- Example:

```
int *p, n, i;
scanf("%d", &n);
p = (int *) malloc (n * sizeof(int));
 for (i=0; i<n; ++i)
     scanf("%d", &p[i]);
```

The n integers allocated can be accessed as *p, *(p+1), *(p+2),…, *(p+n-1) or just as p[0], p[1], p[2], …,p[n-1]

UNIVERSITY OF WOLVERHAMPTON

Example

```
int main()
{
  int i,N;
  float *height;
  float sum=0,avg;

  printf("Input no. of students\n");
  scanf("%d", &N);

  height = (float *)
       malloc(N * sizeof(float));
```

```
printf("Input heights for %d
students \n",N);
  for (i=0; i<N; i++)
   scanf ("%f", &height[i]);

  for(i=0;i<N;i++)
    sum += height[i];

  avg = sum / (float) N;

  printf("Average height = %f \n",
                avg);
  free (height);
  return 0;
}
```

Releasing the Allocated Space: free

- An allocated block can be returned to the system for future use by using the free function
- General syntax:
  - free (ptr);
- where ptr is a pointer to a memory block which has been previously created using malloc
- Note that no size needs to be mentioned for the allocated block, the system remembers it for each pointer returned

Can we allocate only arrays?

- malloc can be used to allocate memory for single variables also
  - p = (int *) malloc (sizeof(int));
  - Allocates space for a single int, which can be accessed as *p
- Single variable allocations are just special case of array allocations

UNIVERSITY OF WOLVERHAMPTON

End of Lecture 1 Part 2