Pointer Fundamentals

- When a variable is defined, the compiler 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 value 3 in the 4 bytes of memory.

Pointers

- When a variable appears in a program, the content of the variable in the memory is used.
  - `y=x;` will read the contents in the 4 bytes of memory, and then assign it to variable `y`.
- To get the address (location) of `x` in the memory, use `&x` gets the address of `x`. (`&` is the referencing operator)
- The address can be passed to a function:
  - `printf("%d", &x);` /97
- The address can also be stored in a variable
  - `int *p1; //p1 is a int pointer
  - p1=&x; //p1=?
  - int *p2=p1; //p2=?

Initializing Pointers

- Like other variables, you have to initialize the pointers before using them!!!
  ```c
  int main(){
    int x;
    int *p;
    //read an int from console
    scanf("%d",p); // error
    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 variable (x) the pointer variable (p) points to.
• To do this, use the * operator (dereferencing operator):
  – int x=3;
  – printf("%d", &x); //97
  – int* p1; //p1 is a int pointer
  – p1=&x; //p1=? *p1=?
  – *(p1)+1; //x=?
  – ++*(p2); //x=?

Pointers and Arrays

• When a pointer variable points to an array element, the operation that can be achieved by array indexing can also be done with the pointer variable:
  – int a[10], *p;
  – p = &a[2];
  – *(p+1)=10;
  – *(++p)=10; // *(++p)=10

Array Names are Constant Pointers

• An array name is a constant pointer which points to the first element of the array.
  – int a[10]; // a points to a[0]
  – *a=5; // a[0]=5
  – *(a+1)=6; // a[1]=6
  – *(a+2)=7; // a[2]=7
  – *(a+2)+1; // a[2]=8

Constant Pointer vs. Pointer Variables

• You cannot assign a different value to a constant pointer:
  – int a[10];
  – int b[10];
  – a = b; //error
• You cannot ++ or - - a constant pointer:
  – *(++a)=6; //error, *(a+1)=6

• You have to initialize a pointer variable before you can use it:
  – int *p;
  – *p= 5; //error
  – int a[10];
  – *p=&a[0];
  – *(p++)=6; //ok
Array Name as Function Argument

- Passing array names as function argument is equivalent to passing pointer variables to the function.

```c
fun1(int a[])
{ //a is a pointer variable
  *(a++)=5; //a points a[1]
  a[0]=6; //a[1]
  (*a)++; //a[1]++
}
```

```
fun2(int* a)
{ //a is a pointer variable
  *a=8;
  *(a+1)=9;
  a[1]++; //a[1]++
}
```

Example

```c
/* strlen: return length of string s */
int strlen(char *s)
{
  char *p = s;
  while (*p != '\0')
    p++;
  return p - s;
}
```

Address Arithmetic

- If two pointer variables point to elements of the same array, there are notions of subtraction and comparisons between the two pointers.

  ```c
  p - q: the difference between array index that p and q point to.
  p < q: true if p points to earlier elements of the array than q does
  ```

Allocating Memory for a Pointer

- Allocate memory so the pointer can point to something:

  ```c
  int a[10], *p, *q, i;
  p = &a[2];
  q = &a[5];
  i = q - p; // i =?
  i = p - q; // i =?
  i = *p - *q; // i = ?
  p < q; // ?
  p == q; // ?
  p+3 == q; // ?
  ```

```c
int main()
{
  int *p;
  p = (int *) malloc(sizeof(int)); // Allocate 4 bytes
  printf("%d", p);
  printf("%d", *p);
  free(p); // This releases the memory to the system.
  }
```