Using Pointers

What is a Pointer?

- A pointer is “a constant or variable that contains an [memory] address” that we can use “to access data.”
- To extract the address for a variable, we use the address operator:

```
&variable_to_extract
```

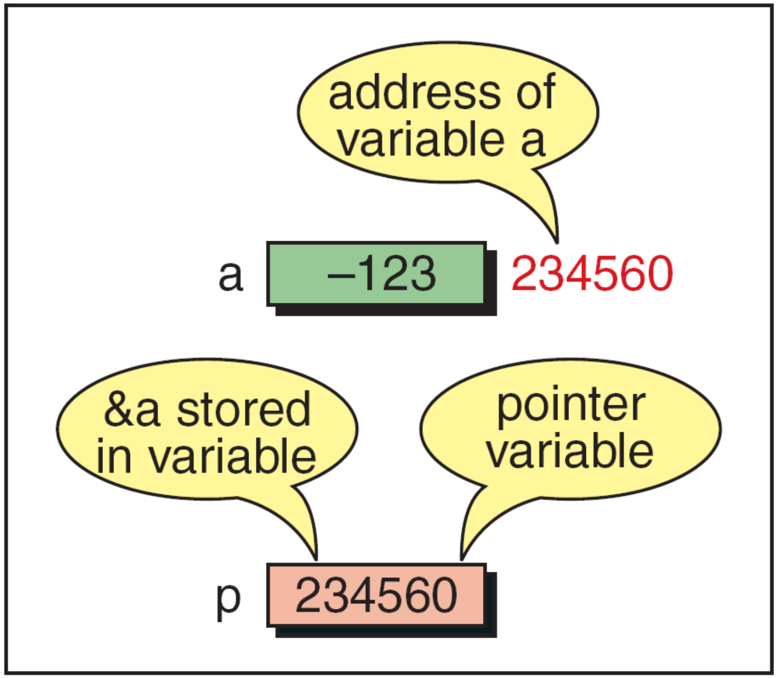
- To print an address we can use the following conversion code:

```
printf(“%p\n”, &variable_to_extract);
```

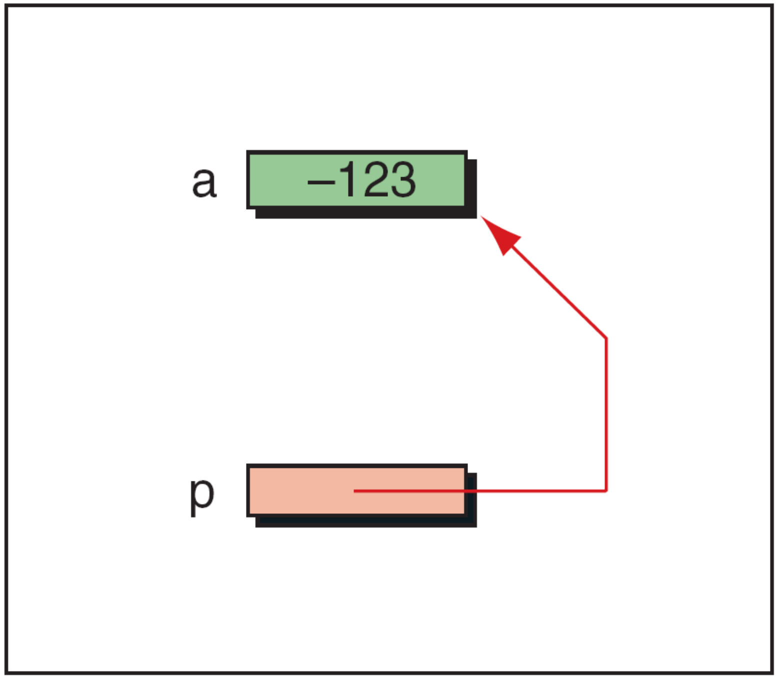
Pointer Variables

- A pointer variable stores the memory address of a variable, not its value!
- We can use a pointer variable, however, to manipulate the data a variable stores.
- Multiple pointer variables can point to the same variable; each pointer variable would point to *the same memory address*.
- If we create a pointer, but we don't want to assign it an address initially, we can assign it the pointer constant **`NULL`**.

Pointer Variable



Physical representation



Logical representation

Code Example of Reading Addresses

Example:

```
  #include <stdio.h>


int main(void)
{


  char a;
  char b;


  printf("The address of variable a is: %p\n",&a);
  printf("The address of variable b is: %p\n",&b);



  return 0;


}
```

The Indirection Operator

- How do we get the value at an address ??
  - A pointer x has an address, *x has the value.
  - ‘*’ stands for the de-referencing operator, which tell the system:
    - Go to the address which is stored in x, and return the value at that address.
  - *pointer → value at the address stored in variable called pointer.

Using the Indirection Operator

- Assume the following:

```
*p = &a;
```

- using a pointer:

```
*p = *p + 1;
(*p)++;
```

Declaring a Pointer

- When we declare a pointer, we use the indirection operator after defining the data type. The data must match the data type of the variable that we reference.
- General format:

```
type variableIdentifier;
type* pointerIdentifier;
```

Code Example of Pointer Declaration

Example:

```
 #include <stdio.h>

int main (void)
{

  int a;
  int* pa;

  a = 14;
  pa = &a;

  printf("\nValue of variable a: %d\n", a);
  printf("Address of variable a: %p\n", &a);
  printf("Value of pointer pa: %p\n", pa);
  printf("Dereferenced value of pointer pa: %d\n\n",*pa);

  return 0;

}
```

Pointer Initialization

- Like variables, C doesn’t automatically initialize pointers.
- An un-initialized pointer can cause run-time errors with unpredictable and hard to debug results.
- To prevent these errors, always initialize your pointers, either by assigning a variable address to them or by assigning the constant **`NULL`** to them.

Pointer Initialization Examples

- Pointer to a declared variable:

```
int a;
int* pa = &a;
```

- Pointer to a **`NULL`** constant:

```
int* pb = NULL;
```

Code Examples of Manipulating Data Using Pointers

```
    /*This program adds two numbers using pointers to demonstrate the concept of pointers.  */
#include <stdio.h>

int main (void)
{
    int  a;
    int  b;
    int  r;
    int* pa = &a;
    int* pb = &b;
    int* pr = &r;

    printf("\nEnter the first number : ");
    scanf ("%d", pa);
    printf("Enter the second number: ");
    scanf ("%d", pb);
    *pr = *pa + *pb;
    printf("\n%d + %d is %d\n\n", *pa, *pb, *pr);
    return 0;
}
```

```
/*          Results:
            Enter the first number : 15
            Enter the second number: 51

            15 + 51 is 66

*/
```

This program shows how the same pointer can point to different data variables in different statements.

```
#include <stdio.h>
void main (void)
{
    int  a;
    int  b;
    int  c;
    int* pMult;

    printf("Enter three numbers and key return: ");
    scanf ("%d %d %d", &a, &b, &c);
    pMult = &a;
    printf("%3d\n", *pMult);
    pMult = &b;
    printf("%3d\n", *pMult);
    pMult = &c;
    printf("%3d\n", *pMult);
}
/*  Results
Enter three numbers and key return: 10 20 30
 10
 20
 30 */
```

Arrays and Pointers

- Design considerations:
  - To access the n^{th} element of the array:
    - Address = starting address + n * size of element.
    - Where,
    - Starting address = name of the array → pointer to the first element.
    - Size of element = size of data type of array.
  - <array name>[n] de-references the value at the n^{th} location in the array.

Arrays and Pointers

- E.g:
  - int temp[10];
  - Assume temp = 100; // starting address.
  - Address of temp[5] = 100 + 2 * 5 = 110
    - Assuming size of int is : *2 bytes*.
  - Value at address 110 is returned when temp[5] is accessed.

Arrays and Pointers

- Passing an array passes a pointer:
  - Passing an array as an argument passes the address, hence arrays are always passed by reference !!!
  - int general (int size, int name []);
    - Expects a pointer to an int array.
  - int general (int size, int *name);
    - Expects a pointer to an int.

Arrays and Pointers

- Arrays as a data type are not pointers !!!
  - Array name just points to the first element of the array, but the properties are not the same as pointers.
- E.g:

```
float test[10];
float *fl;

fl = test; // valid, fl gets address of first element.

fl++; // valid as pointer can be incremented
test++; // not valid as this is an array.
```

Functions and Pointers

- Functions are declared to be of a certain type and must return a value of that type.
- Just like variables, functions can also be declared to be of type *pointer*.
    - float *calc_area (float radius);
    - Pointer functions must return a pointer of the same type as the function declaration.

Functions and Pointers

- E.g:

```
float *calc_area (float radius)
 {
   float *fl;
   .....

   return fl;
 }
```

Pointer Arithmetic

- Pointers can be added to and also subtracted from.
  - How come ??
    - Pointers contain addresses.
- When we add to a pointer, we go to the next specified location depending on the data type. (similarly with subtraction).

Pointer Arithmetic

- <data type> *ptr;
- ptr + d → ptr + d * sizeof (<data type>);
- E.g: (int = 2 bytes and char = 1)
  - int *ptr;
  - ptr + 2 → ptr + 2 * 2 → ptr + 4 !!!
  - char *ptr;
  - ptr + 2 → ptr + 2 * 1 → ptr + 2 !!!

Pointer Arithmetic

```
int i[7]; /* An array of 7 ints */
int *j;    /* A pointer to an int*/
j= i;     /* j points at the start of i*/
*j= 3;    /* Same as i[0]= 3 */
j= &i[0];  /* Same as j= i */
j= j+1;    /* Move j to point at i[1]*/
```

Pointer I/O

- Can actually print the value of pointers:
  - %p or %x conversion character in printf.

```
# include <stdio.h>
int main ()
{
  int *i;
  int j = 10;
  i = &j;
  printf ("address of j is : %p\n", i); /* display address of j */
  printf ("address of j + 1 is : %p\n", i + 1); /* address of j+1 */
}
```

Passing Addresses to Functions

- If a called function includes formal parameters that act as pointers, we need to pass the address to it.
- If we pass a regular variable, we preface the variable name with the address operator.
- If we pass a pointer, we pass only the pointer’s name.

Passing Addresses to Functions

- Function Prototype:

```
int foo(int* px, int* py);
```

- Passing regular variable to foo:

```
c = foo(&a, &b);
```

- Passing pointers to foo:

```
int* pa = &a;
int* pb = &b;
c = foo(pa, pb);
```

Manipulating Passed Pointers

- Treat as passed pointer as you would a pointer declared in the body of the called function:

```
int foo(int* px, int* py)
{
   /*Assigns 50 to a*/
   *px = 50;
}//end foo
```

Returning a Pointer

- We can return a pointer by adding the indirection operator to a functions return data type:

  ```
  int* foo(…);
  ```

- When we return a pointer, the pointer "must point to data in the calling function or a higher-level function." We cannot return a pointer to a local variable, declared inside a called function.

- Write a ‘C’ program to swap to numbers using pointers.

  [Note: use the function

  void swap(int *px, int *py)

Code Examples of Pointers & Functions

Example:Swap using pointers

```
#include <stdio.h>

void Swap(int* px, int* py);
int main (void)
{
  int a = 5;
  int b = 7;

  printf("\nValue of variable a before swap: %d\n", a);
  printf("Value of variable b before swap: %d\n", b);
  Swap(&a, &b);
  printf("\nValue of variable a after swap: %d\n", a);
  printf("Value of variable b after swap: %d\n", b);
  return 0;
}
void Swap(int* px, int* py)
{
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
  return;
}
```

- Write a ‘c’ program to find minimum of two numbers using pointers.

  [note: use the function:

  ```
  int* ReturnSmaller(int* p1, int *p2)
  ```

Write a ‘C’ program to find maximum of three numbers using pointers.

[use UDF to find maximum of 3 numbers]

Find the min. of two numbers using pointers

```
#include <stdio.h>
int* ReturnSmaller(int* p1, int* p2);
int main (void)
{
  int a;
  int b;
  int* pSmaller = NULL;

  printf("Please enter an integer: ");
  scanf(" %d", &a);
  printf("Please enter another integer: ");
  scanf(" %d", &b);
  pSmaller = ReturnSmaller(&a, &b);
  printf("\n%d is the smaller value.\n\n", *pSmaller);
  return 0;
}//end main

int* ReturnSmaller(int* p1, int* p2)
{

  return (*p1 < *p2 ? p1 : p2);

}
```

Pointers to Pointers

- Especially with more advanced data structures, we might need to create pointers that reference other pointers.
- To do this, we would declare the pointer-to-a-pointer using an additional indirection operator:

```
int a = 58;
int* p = &a;
int** q = &p;
```

Code Examples of a Pointer-to-Pointer

Example:

```
    #include <stdio.h>

int main (void)
{
  int a;
  int* pa = &a;
  int** pp = &pa;

  printf("\nPlease enter an integer value: ");
  scanf("%d",&a);  /*78*/

  printf("\nValue of variable a: %d", a); /*78*/
  printf("\nAddress of variable a: %p", &a); /*FFDA*/
  printf("\nValue of pointer pa: %p", pa); /*FFDA*/
  printf("\nDe-referenced value of pointer pa: %d", *pa); /*78*/
  printf("\nValue of pointer pp: %p", pp); /*FFDA*/
  printf("\nDe-referenced value of pointer pp: %d\n\n", **pp); /*78*/

  return 0;
}
```

Pointer Arithmetic

- You can add or subtract integers to/from pointers to modify the address of a pointer.
- If you add an integer to a pointer, it increases the pointer’s address.
- If you subtract an integer from a pointer, it decreases the pointer’s address.

Comparing Pointers

- We can use relational operators to compare pointers by the addresses they reference.
- We often use pointer comparisons when the pointers point to similar objects.
- Example:

```
if(pa < pb)
{
   printf(“pa points to lower memory.”);
}else{
   printf(“pb points to lower memory.”);
}//end if
```

Arrays as Pointers

- Remember, that an array name is a pointer to the first location in the array.
- Because of this syntax, we can use pointers instead of subscripts to read/write to array elements.

Array to Pointers - Example

```
int scores[] = {78, 84, 97, 58, 81};
//Prints 78 to the screen:
printf(“%d”, *scores);
//Prints 84 to the screen:
printf(“%d”, *(scores+1));
//Prints 97 to the screen:
printf(“%d”, *(scores+2));
```

Pointers & Character Arrays

- We can also use pointers to assign string variables to a character array:

  ```
  char courseName[]  = “CS103”;
  ```

  is the same as …

  ```
  char* courseNameP;
  courseNameP = “CS103”;
  ```

Arrays of Pointers

- We can create an array of pointers.
- In an array of pointers, each element holds a pointer to a memory location.
- The data type to which each element points and the data type of the array must match.
- Examples:

```
int* examScores[10];
char* ptrNames[10];
```

Code Examples of a Pointers & Arrays

Example:

```
  #include<stdio.h>

int main(void)
{
  char anyPhrase[]="Go Jaguars!";
  char *ptr=anyPhrase;
  int i;

  for(i=0; i<sizeof(anyPhrase)-1; i++)
    {
      putchar('\n');
      putchar(*ptr++);
    }

  putchar('\n');

  return 0;

}
```

Example: pointer for string.

```
#include<stdio.h>
int main(void)
{
  char* ptr0;

  ptr0="ptr0 points to this string.";
  puts(ptr0);
  ptr0="A shorter string.";
  puts(ptr0);
  ptr0="A new string for ptr0 that is longer than the previous.";
  puts(ptr0);
  return 0;
}
```

Pointer Array

```
#include <stdio.h>
void PrintErrorMsg(int errorNum);
int main (void)
{
  PrintErrorMsg(3);

  return 0;

}//end main
void PrintErrorMsg(int errorNum)
{
  static char *errorList[] =
  {
    "Cannot open file.\n",
    "Read error.\n",
    "Write error.\n",
    "Media failure.\n"
  };
  printf("%s", errorList[errorNum]);
  return;
}//end PrintErrorMsg
```