CL1002 – Programming Fundamentals Lab

Lab # 12

Pointers

Instructor: Muhammad Saood Sarwar

Email: saood.sarwar@nu.edu.pk

Department of Computer Science,

National University of Computer and Emerging Sciences FAST Peshawar

C Pointers

Pointers are powerful features of C and C++ programming. Before we learn pointers, let's learn about addresses in C programming.

Address in C

If you have a variable var in your program, &var will give you its address in the memory.

We have used address numerous times while using the scanf() function.

```
scanf("%d", &var);
```

Here, the value entered by the user is stored in the address of var variable. Let's take a working example.

Example 1 | Address in C

```
#include <stdio.h>
int main()
{
 int var = 5;
 printf("var: %d\n", var);

 // Notice the use of & before var
 printf("address of var: %p", &var);
 return 0;
}
```

Output

```
var: 5
address of var: 0x7ffeb9656ca4
```

Note: You will probably get a different address when you run the above code.

C Pointers

Pointers (pointer variables) are special variables that are used to store addresses rather than values.

Pointer Syntax

Here is how we can declare pointers.

```
int* p;
```

Here, we have declared a pointer *p* of *int* type.

Assigning addresses to Pointers

Let's take an example.

```
int* pc, c;
c = 5;
pc = &c;
```

Here, 5 is assigned to the *c* variable. And, the address of *c* is assigned to the *pc* pointer.

Get Value of Thing Pointed by Pointers

To get the value of the thing pointed by the pointers, we use the * operator. For example:

```
int* pc, c;
c = 5;
pc = &c;
printf("%d", *pc);   // Output: 5
```

Here, the address of c is assigned to the *pc* pointer. To get the value stored in that address, we used **pc*.

Changing Value Pointed by Pointers

Let's take an example.

```
int* pc, c;
c = 5;
pc = &c;
c = 1;
printf("%d", c);    // Output: 1
printf("%d", *pc);  // Output: 1
```

We have assigned the address of *c* to the *pc* pointer.

Then, we changed the value of *c* to 1. Since *pc* and the address of *c* is the same, **pc* gives us 1.

Let's take another example.

```
int* pc, c;
c = 5;
pc = &c;
```

```
*pc = 1;
printf("%d", *pc);   // Output: 1
printf("%d", c);     // Output: 1
```

We have assigned the address of *c* to the pc pointer.

Then, we changed **pc* to 1 using **pc = 1;*. Since *pc* and the address of *c* is the same, *c* will be equal to 1.

Example 2 | Working of Pointers

```
#include <stdio.h>
int main()
{
  int* pc, c;

  c = 22;
  printf("Address of c: %p\n", &c);
  printf("Value of c: %d\n\n", c);  // 22

  pc = &c;
  printf("Address of pointer pc: %p\n", pc);
  printf("Content of pointer pc: %d\n\n", *pc); // 22

  c = 11;
  printf("Address of pointer pc: %p\n", pc);
  printf("Content of pointer pc: %d\n\n", *pc); // 11

  *pc = 2;
  printf("Address of c: %p\n", &c);
  printf("Value of c: %d\n\n", c); // 2
  return 0;
}
```

Output

```
Address of c: 0x7fff17a620fc
Value of c: 22

Address of pointer pc: 0x7fff17a620fc
```

```
Content of pointer pc: 22

Address of pointer pc: 0x7fff17a620fc
Content of pointer pc: 11

Address of c: 0x7fff17a620fc
Value of c: 2
```

Relationship Between Arrays and Pointers

An array is a block of sequential data. Let's write a program to print addresses of array elements.

Example 3 | Pass Arrays to Functions

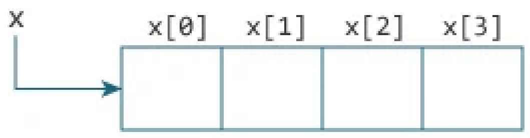
```
#include <stdio.h>
int main() {
  int x[4];
  int i;

  for(i = 0; i < 4; ++i) {
     printf("&x[%d] = %p\n", i, &x[i]);
  }

  printf("Address of array x: %p", x);
  return 0;
}
```

Output

```
&x[0] = 0x7fffca918820
&x[1] = 0x7fffca918824
&x[2] = 0x7fffca918828
&x[3] = 0x7fffca91882c
Address of array x: 0x7fffca918820
```

There is a difference of 4 bytes between two consecutive elements of array x. It is because the size of *int* is 4 bytes (on our compiler).

Notice that, the address of *&x[0]* and *x* is the same. It's because the variable name x points to the first element of the array.



From the above example, it is clear that &x[0] is equivalent to x. And, x[0] is equivalent to *x.

Similarly,

- &x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1).
- &x[2] is equivalent to x+2 and x[2] is equivalent to *(x+2).
- ...
- Basically, &x[i] is equivalent to x+i and x[i] is equivalent to *(x+i).

Example 4 | Pointers and Arrays

```
#include <stdio.h>
int main() {
 int i, x[6], sum = 0;
 printf("Enter 6 numbers:\n");

 for(i = 0; i < 6; ++i) {
 // Equivalent to scanf("%d", &x[i]);
     scanf("%d", x+i);
 // Equivalent to sum += x[i]
     sum += *(x+i);
 }
 printf("Sum = %d", sum);

 return 0;
}
```

Output

```
Enter 6 numbers:
2
3
4
5
```

```
6
7
Sum = 27
```

Example 5 | Arrays and Pointers

```
#include <stdio.h>
int main() {

 int x[5] = {1, 2, 3, 4, 5};
 int* ptr;

 // ptr is assigned the address of the third element
 ptr = &x[2];

 printf("*ptr = %d \n", *ptr);   // 3
 printf("*(ptr+1) = %d \n", *(ptr+1)); // 4
 printf("*(ptr-1) = %d", *(ptr-1));  // 2

 return 0;
}
```

Output

```
*ptr = 3
*(ptr+1) = 4
*(ptr-1) = 2
```

References:

https://www.programiz.com/c-programming/c-pointers

https://www.programiz.com/c-programming/c-pointers-arrays