CSE1102: Structured Programming Language

Pointer

Study Materials

- **The C Programming Language**, by Brian W. Kernighan & Dennis M. Ritchie, 2nd Ed., Prentice Hall.

- **Sams Teach Yourself C** in 21 Days, by Bradly L. Jones and Peter Aitken, 6th Ed., Sams Publishing.
- **Let Us C**, by Yashavant Kanetkar, 13th Ed., BPB Publications.

Contents

Study Materials

Textbook:

- **The C Programming Language**, by Brian W. Kernighan & Dennis M. Ritchie, 2nd Ed., Prentice Hall.

Reference books:

- **Sams Teach Yourself C** in 21 Days, by Bradly L. Jones and Peter Aitken, 6th Ed., Sams Publishing.
- **Let Us C**, by Yashavant Kanetkar, 13th Ed., BPB Publications.

1 Pointer

Pointer

A pointer is a variable that containts the address of a variable.

- Pointer "points to a variable"
- A pointer is always denoted by (*) asterisk symbol

```
int *p; // p is a pointer
int p; // p is a variable
```

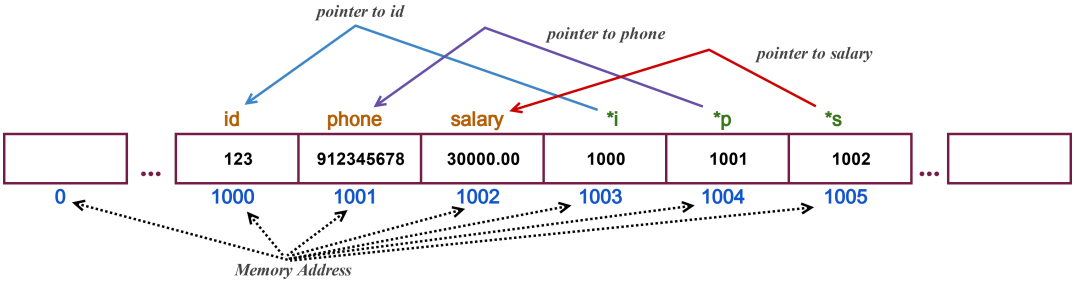
- A pointer always holds a computer memory address
- When a variable is declared, a memory location is allocated for that variable in the Randdom Access Memory (RAM)
- A memory location is called an address
- The ability of using a memory address (through pointers) usually leads to a more compact and efficient code

Pointer *(Cont'd)*

```
int id = 123;
int phone = 912345678;
float salary = 30000.00;

int *i, *p;
float *s;

i = &id;
p = &phone;
s = &salary;
```



2 Pointer Declaration and Initialization

Pointer Declaration and Initialization

Declaration:

```
<type> *<pointer_name>
```

```
int *p;
char *c;
double *d;
float *f;
```

Pointer Initialization:

```
int i = 1;
float num = 10.50;
p = &i;
f = &num;
```

Declaration and Initialization:

```
int y = 2;
double salary = 10000.00;
int *q = &y;
double *sal = &salary;
```

3 Reference and De-reference operator

Reference and De-reference operator

- & - Reference operator
- * - De-reference operator
- Reference operator (&) gives us the **address** of a variable
- De-Reference operator () gives us the **value** from the address

```c
#include<stdio.h>
void main(){
int x = 10;// x is a variable
int *i = &x; //pointer to x

printf ( "%u \n", &x); // u for unsigned
    integer address
printf ( "%u \n", i); //value of i
printf ( "%u \n", *i); //value that i is
    pointing to
}
```

Output:

```
1374441596
1374441596
10
```

4 Address of Pointer

Address of Pointer

- Poiner itself has it's own addreess

```c
#include<stdio.h>
void main(){
int x = 10;// x is a variable
int *i = &x; //pointer to x

printf ( "Address of x: %u \n", &x);
printf ( "Value of i: %u \n", i);
printf ( "Value that the pointer i is
    pointing to: %u \n", *i);
printf ( "Address of pointer i: %u \n", &i)
    ;
}
```

Address of Pointer

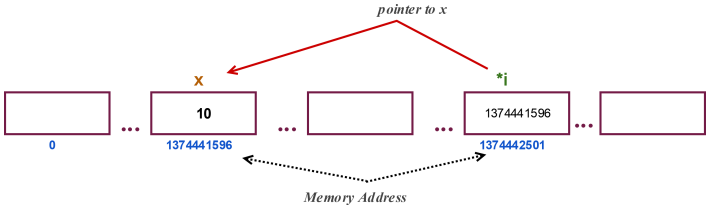
- Poiner itself has it's own addreess

```c
#include<stdio.h>
void main(){
int x = 10;// x is a variable
int *i = &x; //pointer to x

printf ( "Address of x: %u \n", &x);
printf ( "Value of i: %u \n", i);
printf ( "Value that the pointer i is
    pointing to: %u \n", *i);
printf ( "Address of pointer i: %u \n", &i)
    ;
}
```

Output:

```
Address of x: 1374441596
Value of i: 1374441596
Value that the pointer i is
    pointing to: 10
Address of pointer i:
    1374442501
```



5 Pointer to Pointer

Pointer to Pointer

- A poiner can point to other pointer

```c
#include <stdio.h>
void main () {
   int  var = 100;
   int  *ptr;
   int  **pptr;

   ptr = &var;
   pptr = &ptr;

   /* take the value using pptr */
   printf("var = %d\n", var );
   printf("*ptr = %d\n", *ptr );
   printf("**pptr = %d\n", **pptr);
}
```

Pointer to Pointer

- A poiner can point to other pointer

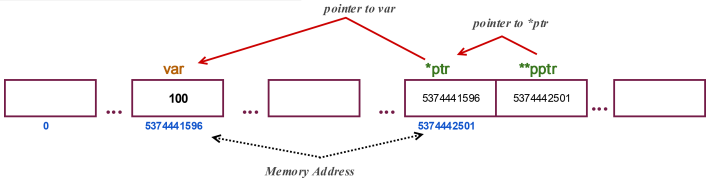
```c
#include <stdio.h>
void main () {
   int  var = 100;
   int  *ptr;
   int  **pptr;

   ptr = &var;
   pptr = &ptr;

   /* take the value using pptr */
   printf("var = %d\n", var );
   printf("*ptr = %d\n", *ptr );
   printf("**pptr = %d\n", **pptr);
}
```

Output:

```
var = 100
*ptr = 100
**pptr = 100
```



6 NULL Pointer

NULL Pointer

- A pointer that is assigned NULL is called a null pointer
- The NULL pointer is a constant with a value of zero (0)
- In most operating systems, memory address 0 is reserved
- If a pointer contains the NULL (0) value, it is assumed that the pointer is pointing to nothing

```c
#include <stdio.h>

int main () {
   int  *ptr = NULL;
   printf("Value of ptr =  %x\n", ptr  );

   return 0;
}
```

Output:

```
Value of ptr = 0
```

```c
if(ptr)     /* true, if p is not null */
if(!ptr)    /* true, if p is null */
```

7 Size of Pointers

Size of Pointers

- Size of a pointer is 8 byte in 64 bit C/C++ compiler
- Size of a pointer is 4 byte in 32 bit C/C++ compiler

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
  printf("Size of int pointer: %d \n",
      sizeof(int*));
  printf("Size of float pointer: %d \n",
      sizeof(float*));
  printf("Size of double pointer: %d \n",
      sizeof(double*));

 return 0;
}
```

Output:

```
Size of int pointer: 8
Size of float pointer: 8
Size of double pointer: 8
```

8 Passing Function Arguments as Reference

Passing Function Arguments as Reference

- Function call by reference

```
1  #include<stdio.h>
2  // function prototype or function declaration
3  void swap(int *a, int *b);
4
5  int main(){
6  int a = 10, b = 20;
7
8  printf("values before swap, a = %d , b = %d \n", a, b)
       ;
9  swap(&a, &b); //passing arguments as reference
10 printf("values after swap, a = %d , b = %d \n", a, b);
11 }
12
13 void swap(int *a, int *b){
14  int tmp;
15  tmp = *a;
16  *a = *b;
17  *b = tmp;
18 }
```

Passing Function Arguments as Reference

- Function call by reference

```c
#include<stdio.h>
// function prototype or function declaration
void swap(int *a, int *b);

int main(){
int a = 10, b = 20;

printf("values before swap, a = %d , b = %d \n", a, b)
    ;
swap(&a, &b); //passing arguments as reference
printf("values after swap, a = %d , b = %d \n", a, b);
}

void swap(int *a, int *b){
 int tmp;
 tmp = *a;
 *a = *b;
 *b = tmp;
}
```

Output:

```
values before swap,
    a = 10 , b = 20
values after swap,
    a = 20 , b = 10
```

9 Pointing to an Array

Pointing to an Array

- We can declare a pointer variable and initialize it to point at the array.

```
1  int array[100];
2  int *p_array;
3
4  p_array = array;      //valid
5  p_array = &array[0]; //valid
```

10 Passing Array Reference to a Function

Passing Array Reference to a Function

```c
#include<stdio.h>

void function(int, int[]);

int main(){
      int a = 20;
      int arr[5] = {11,22,33,44,55};

      function(a, &arr[0]);

      int i;
      for (i=0;i<5;i++){
         // Accessing each elements in the array
         printf("value of arr[%d] is %d\n",i,arr[i]);
      }

      return 0;
}

void function(int a, int *arr_ptr){
    arr_ptr[2] = a;
}
```

Passing Array Reference to a Function

```c
#include<stdio.h>

void function(int, int[]);

int main(){
      int a = 20;
      int arr[5] = {11,22,33,44,55};

      function(a, &arr[0]);

      int i;
      for (i=0;i<5;i++){
         // Accessing each elements in the array
         printf("value of arr[%d] is %d\n",i,arr[i]);
      }

      return 0;
}

void function(int a, int *arr_ptr){
    arr_ptr[2] = a;
}
```

Output:

```
value of arr[0] is
    11
value of arr[1] is
    22
value of arr[2] is
    20
value of arr[3] is
    44
value of arr[4] is
    55
```

11 Conclusion

Thanks

Thanks for your time and attention!