Pointers – Basics-1

Atul Gupta

Simple Variables

Consider the declaration in C

int i = 3 ;

This declaration tells the C compiler to:

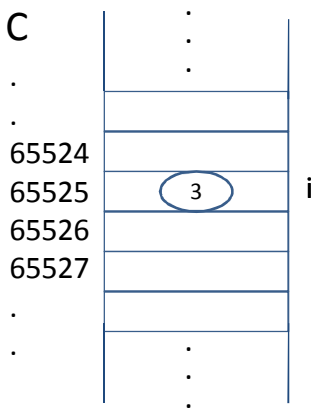
- Reserve space in memory to hold the integer value.
- Associate the name i with this memory location.
- Store the value 3 at this location.

Simple Variables

Consider the declaration in C

```
int i = 3 ;
```

Name of a simple variable, which will store a value

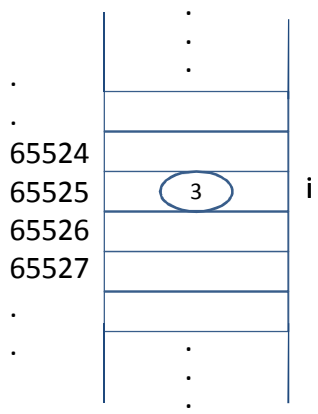


Simple Variables

Operator '&'

```
main( )
{
    int i = 3 ;
    printf ( "\nAddress of i = %u", &i ) ;
    printf ( "\nValue of i = %d", i ) ;
}
```

Output?

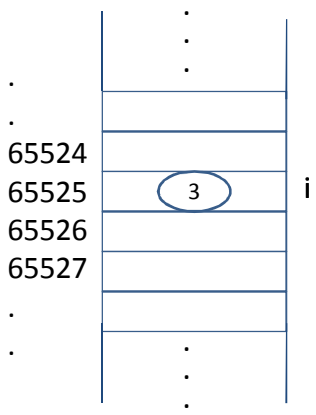


Simple Variables

Operator '*'

```
main( )
{
    int i = 3 ;
    printf ( "\nAddress of i = %u", &i) ;
    printf ( "\nValue of i = %d", i ) ;
     printf ( "\nValue of of i = %u", *(&i)) ;
}
```

Output?



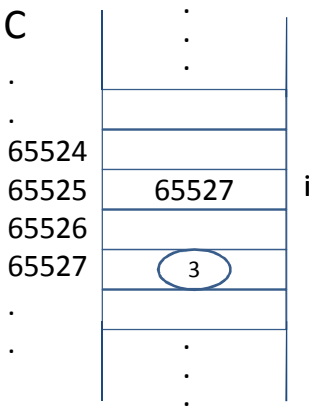
Pointer Variables

Consider the declaration in C

int *i = 3 ;

Name of another variable, but this time it's a pointer variable

This variable will store address rather than value



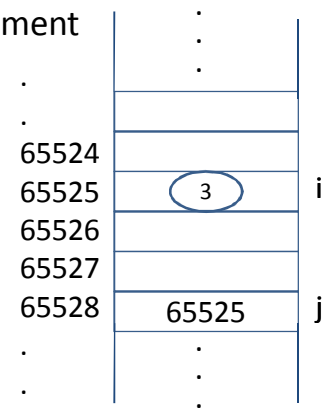
An Example

Consider the following C code fragment

```
main( )
{
    int i = 3 ;
    int *j ;

    j = &i ;

    printf ( "\nAddress of i = %u", &i ) ;
    printf ( "\nAddress of i = %u", j ) ;
    printf ( "\nAddress of j = %u", &j ) ;
    printf ( "\nValue of j = %u", j ) ;
    printf ( "\nValue of i = %d", i ) ;
    printf ( "\nValue of i = %d", *( &i ) ) ;
    printf ( "\nValue of i = %d", *j ) ;
}
```

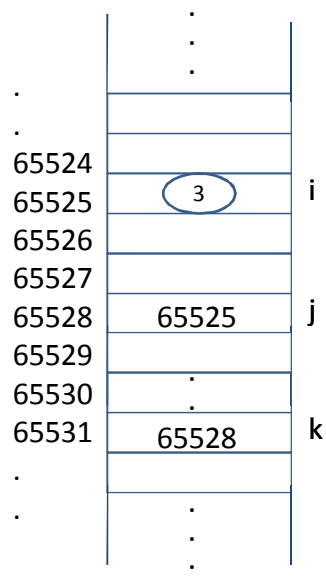


Pointer to a Pointer!

```
main( )
{
    int i = 3, *j, **k ;

    j = &i ;
    k = &j ;

    printf ( "\nAddress of i = %u", *k ) ;
    printf ( "\nAddress of j = %u", &j ) ;
    printf ( "\nAddress of j = %u", k ) ;
    printf ( "\nAddress of k = %u", &k ) ;
    printf ( "\nValue of j   = %u", j ) ;
    printf ( "\nValue of k   = %u", k ) ;
    printf ( "\nValue of i   = %d", i ) ;
    printf ( "\nValue of i   = %d", * ( &i ) ) ;
    printf ( "\nValue of i   = %d", *j ) ;
    printf ( "\nValue of i   = %d", **k ) ;
}
```



Parameter Passing

In function calls, parameters are passed by

- Value (of the arguments)
- Reference (addresses of the arguments)

Parameter Passing

- Pass by value

```
main( )
{
    int a = 10, b = 20 ;
    swapv ( a, b ) ;
    printf ( "\na = %d b = %d", a, b ) ;
}
```

Output?

```
swapv ( int  x, int  y )
{
    int t ;

    t = x ;
    x = y ;
    y = t ;

    printf ( "\nx = %d y = %d", x, y ) ;
}
```

Parameter Passing

- Pass by Reference

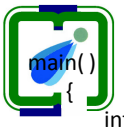
```
main( )
{
    int a = 10, b = 20 ;

    swapr ( &a, &b ) ;
    printf ( "\na = %d b = %d", a, b ) ;
}
```

```
swapr( int   *x, int  *y )
{
    int t ;

    t = *x ;
    *x = *y ;
    *y = t ;
}
```

Output?



Pointer Arithmetic

```
main( )
{
    int i = 3, *x ;
    float j = 1.5, *y ;
    char   k = 'c', *z ;

    printf ( "\nValue of i = %d", i ) ;
    printf ( "\nValue of j = %f", j ) ;
    printf ( "\nValue of k = %c", k ) ;
    x = &i ;
    y = &j ;
    z = &k ;
    printf ( "\nOriginal address in x = %u", x ) ;
    printf ( "\nOriginal address in y = %u", y ) ;
    printf ( "\nOriginal address in z = %u", z ) ;
    x++ ;
    y++ ;
    z++ ;
    printf ( "\nNew address in x = %u", x ) ;
    printf ( "\nNew address in y = %u", y ) ;
    printf ( "\nNew address in z = %u", z ) ;
}
```

Output?

Pointer Arithmetic

1. Addition of a number to a pointer. For example,

```
int  i = 4, *j, *k ;
     j = &i ;
     j = j + 1 ;
     j = j + 9 ;
     k = j + 3 ;
```

2. Subtraction of a number from a pointer. For example,

```
int  i = 4, *j, *k ;
     j = &i ;
     j = j - 2 ;
     j = j - 5 ;
     k = j - 6 ;
```

Pointer Arithmetic

3. Subtraction of one pointer from another

```
main( )
    {
       int arr[ ] = { 10, 20, 30, 45, 67, 56, 74 } ;
       int *i, *j ;

       i = &arr[1] ;
       j = &arr[5] ;
       printf ( "%d %d", j - i, *j - *i ) ;
    }
```

Pointer Arithmetic

4. Comparison of two pointer variables

```
main( )
    {
       int arr[ ] = { 10, 20, 36, 72, 45, 36 } ;
       int *j, *k ;

       j = &arr [ 4 ] ;
       k = ( arr + 4 ) ;

       if ( j == k )
          printf ( "The two pointers point to the same location" ) ;
       else
          printf ( "The two pointers do not point to the same location" ) ;
    }
```

Pointer Arithmetic

- *Do not attempt following operations on pointers !*
  - Addition of two pointers
  - Multiplication of a pointer with a constant
  - Division of a pointer with a constant