IS1101: Programming and Problem Solving

Pointers - 1

Upul Anuradha

Variables

int mark = 67;

What the declaration tells the C compiler?

Variables

```
int mark = 67;
```

What the declaration tells the C compiler?

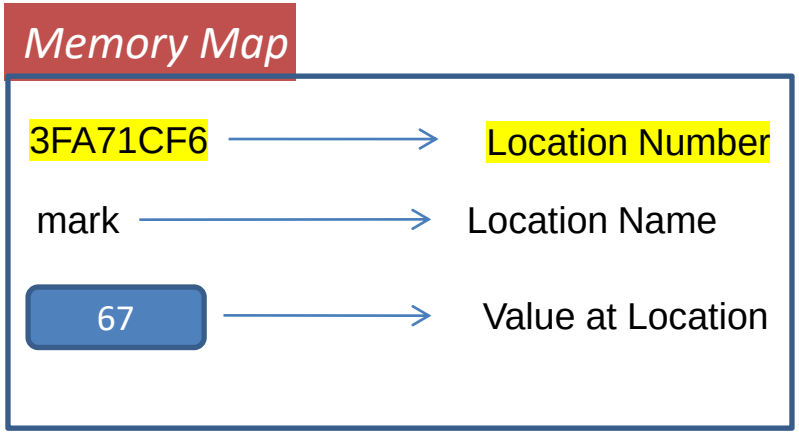
- Reserve space in memory to hold the integer value
- Associate the name “mark” with this memory location
- Store the value 67 at this location

Variables

int mark = 67;

What the declaration tells the C compiler?

- Reserve space in memory to hold the integer value
- Associate the name “mark” with this memory location
- Store the value 67 at this location



Variables

- int mark = 67;
- char grade = ‘A’;
- char[50] name = “Saman Kumara”;
- double gpa = 3.2;

Variables

- int mark = 67;
- char grade = ‘A’;
- char[50] name = “Saman Kumara”;
- double gpa = 3.2;

an Integer value

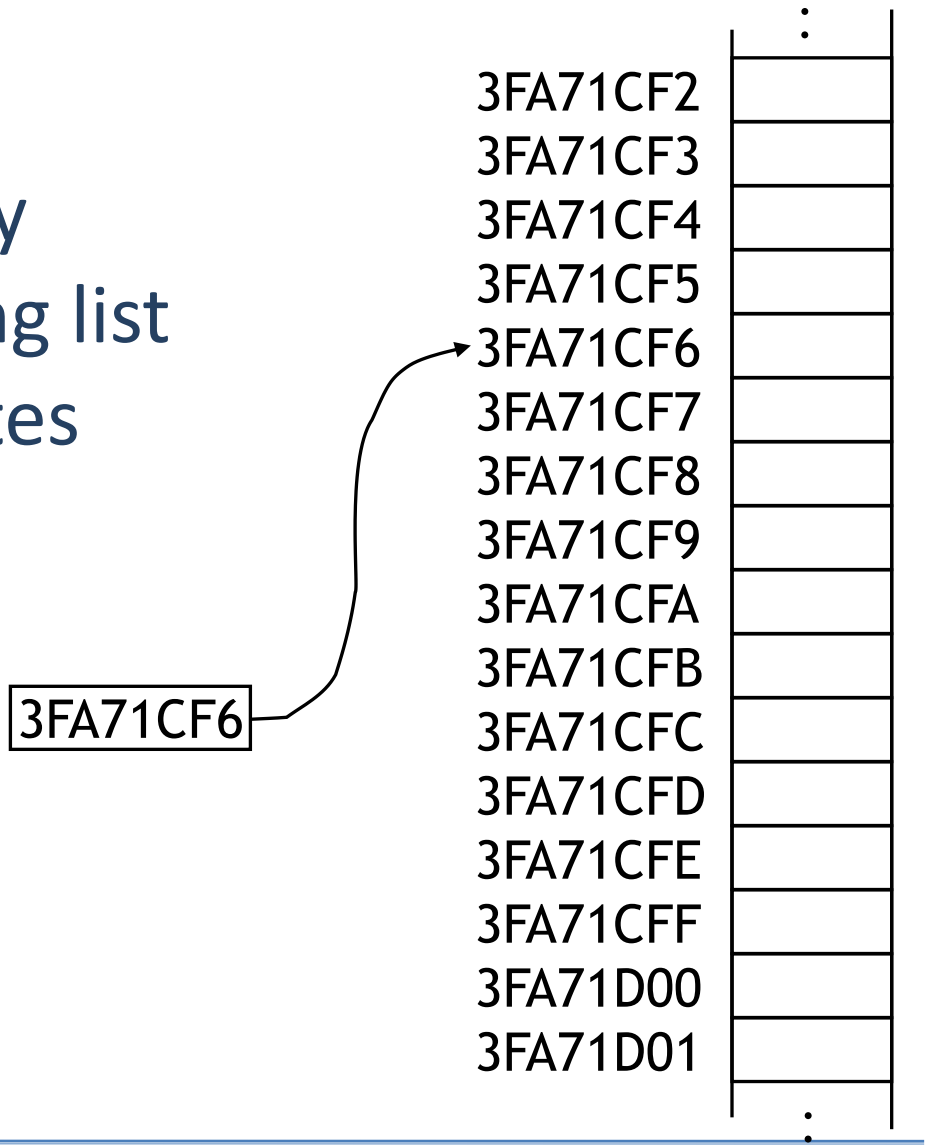
a character

a set of characters

a decimal value

Memory Address

- Computer memory consists of one long list of addressable bytes



Variables

- int mark = 67;
- char grade = 'A';
- char[50] name = "Saman Kumara";
- double gpa = 3.2;
- memory_address mem = 3FA71CF6

an Integer value

a character

a set of characters

a decimal value

a Memory Address!

What is a Pointer?

- Variable which contains the address of another variable
- Pointer variable used to hold an address of the memory
  - i.e., direct address of the memory location.
- Like any variable or constant, you must declare a pointer before using it to store any variable address.

What is a Pointer?

- The address which a pointer holds is the location of another entity (typically another variable) in memory.

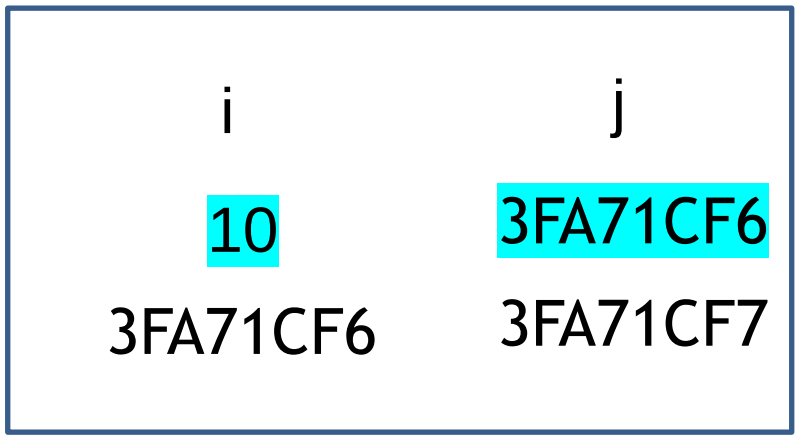
- For example, if one variable contains the address of another variable, the first variable is said to *point* to the second.

Address of an Address…

Code Snippet

```
void main()
{
   int i = 10;
   int *j;
   j = &i;
}
```

Memory Map



Why Pointers?

- To return more than one value from a function (using pass by reference)
- To create and process strings
- To pass arrays & strings more conveniently from one function to another
- To manipulate arrays more easily by moving pointers to them, Instead of moving the arrays themselves
- To allocate memory and access it (Dynamic Memory Allocation)
- To create complex data structures such as Linked List, Where one data structure must contain references to other data structures

Advantages of Pointers

- A pointer enables us to access a variable that is defined outside the function.
- Pointers are more efficient in handling the data tables.
- Pointers reduce the length and complexity of a program.
- They increase the execution speed.
- The use of a pointer array to character strings results in saving of data storage space in memory.
- The function pointer can be used to call a function
- Pointer arrays give a convenient method for storing strings
- Many of the 'C' Built-in functions that work with strings use Pointers
- lt provides a way of accessing a variable without referring to the variable directly

Declare a Pointer Variable

- * used with pointer variables

  int *numPtr;

  – Defines a pointer to an int (pointer of type int *)
- Multiple pointers require using a * before each variable definition

  int *numPtr1, *numPtr2;
- Can define pointers to any data type
- Initialize pointers to 0, NULL, or an address
- 0 or NULL – points to nothing (NULL preferred)

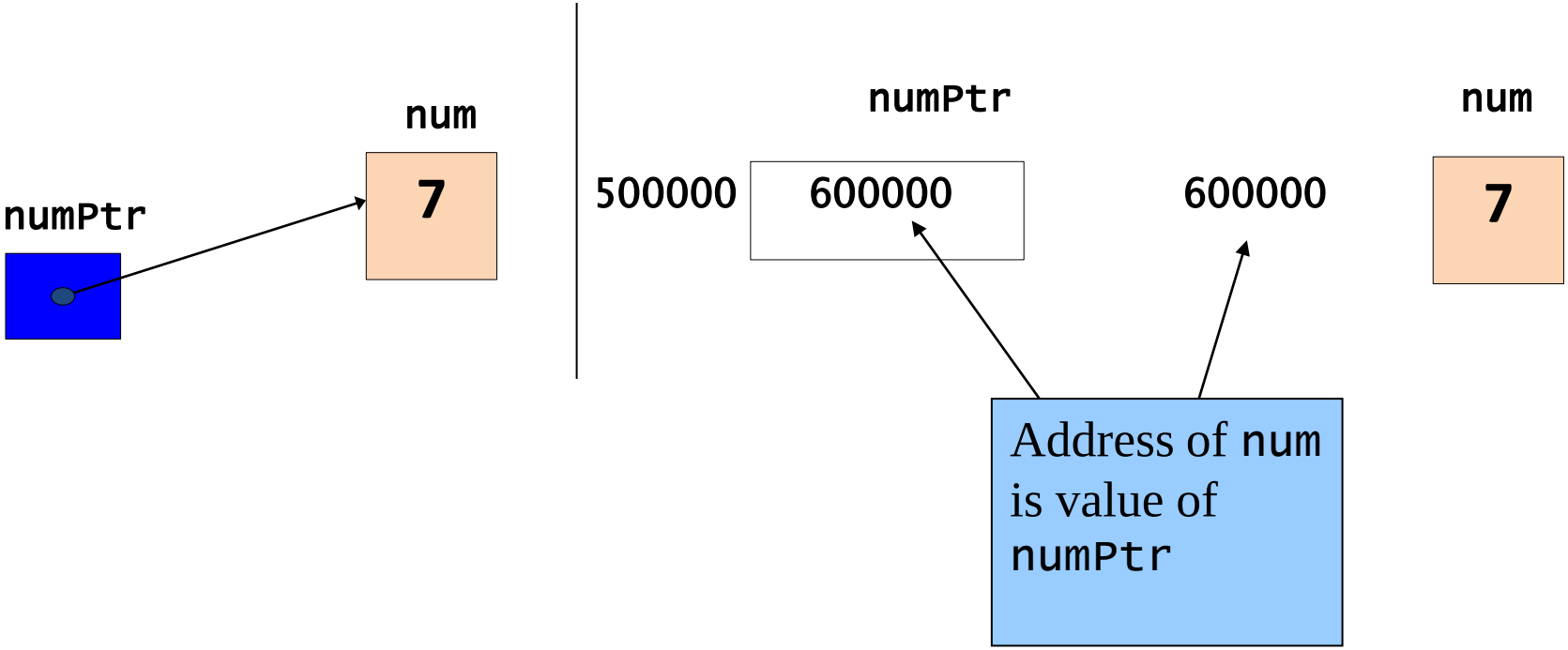
  int *numPtr = NULL; or int *numPtr = 0;

Pointer Operators

- Symbol '&' is called address operator
- Returns address of operand

```
int num = 7;
int *numPtr;
numPtr = &num;   /* numPtr gets address of num */
                 //numPtr "points to" num
```

Pointer Operators



Pointer Operators

- Symbol '*' is called indirection/dereferencing operator
- Returns a synonym/alias of what its operand points to

  *numPtr returns num (because numPtr points to num)

- * can also be used for assignment
  - Returns alias to an object
  - *numPtr = 10; /* changes num to 10 */
- For example, the statement

```
printf( "%d", *numPtr );
```

prints the value of variable num, namely 7.

Pointer Operators

- * and & operators are complements of one another
- They can be applied in consecutively in either order as the same result will be printed

  Example: &*numPtr or *&numPtr

Example

```
3  void main() {
4      int age = 20;
5      printf("value of the age = %d\n", age);
6      printf("Memory addrress of age: = %d\n", &age);
7      int *ageptr;
8      ageptr = &age;
9      printf("value of the ageptr: = %p\n", ageptr);
10     printf("value of the Memory addrress of age: = %d\n", *ageptr);
11 }
```

Example

```
3  void main() {
4      int age = 20;
5      printf("value of the age = %d\n", age);
6      printf("Memory addrress of age: = %d\n", &age);
7      int *ageptr;
8      ageptr = &age;
9      printf("value of the ageptr: = %p\n", ageptr);
10     printf("value of the Memory addrress of age: = %d\n", *ageptr);
11 }
```

printf()

"%d"	->	Integers
"%c"	->	Characters
"%s"	->	Strings (array of chars)
"%f"	->	Decimal values
"%p"	->	Memory addresses

Example

```
3  void main() {
4      int age = 20;
5      printf("value of the age = %d\n", age);
6      printf("Memory addrress of age: = %d\n", &age);
7      int *ageptr;
8      ageptr = &age;
9      printf("value of the ageptr: = %p\n", ageptr);
10     printf("value of the Memory addr ss of age: = %d\n", *ageptr);
11 }
```

Printing as a memory address

Example

```
3  void main() {
4      int age = 20;
5      printf("value of the age = %d\n", age);
6      printf("Memory addrress of age: = %d\n", &age);
7      int *ageptr;
8      ageptr = &age;
9      printf("value of the ageptr: = %p\n", ageptr);
10     printf("value of the Memory addrress of age: = %d\n", *ageptr);
11 }
```

Example

```
3  void main() {
4      int age = 20;
5      printf("value of the age = %d\n", age);
6      printf("Memory addrress of age: = %d\n", &age);
7      int *ageptr;
8      ageptr = &age;
9      printf("value of the ageptr: = %p\n", ageptr);
10     printf("value of the Memory addrress of age: = %d\n", *ageptr);
11 }
```

```
value of the age = 20
Memory addrress of age: = 6487572
value of the ageptr: = 000000000062FE14
value of the Memory addrress of age: = 20
```

scanf()

scanf("%d", &age);

Why do we need to use '&' when reading values?

Passing Arguments to Functions

- There are two ways to pass arguments to a function: call-by-value and call-by-reference.
- All arguments in C are passed by value.
- In C, you use pointers and the indirection operator to simulate call-by-reference.
- When calling a function with arguments that should be modified, the addresses of the arguments are passed.

Passing by Value: Example

```
3   int cubeByValue(int n); // prototype
4
5   void main() {
6
7       int number = 4; // inicialize the number
8
9       printf("original value of number: %d\n", number);
10
11      //pass the number by value to cubeByValue
12      number = cubeByValue(number);
13
14      printf("The new value of number: %d\n", number);
15
16  }//main()
17
18  //calculate the cube value of n
19  int cubeByValue(int n) {
20      return n*n*n;
21  }
```

Passing by Value: Example

```
3   int cubeByValue(int n); // prototype
4
5   void main() {
6
7       int number = 4; // inicialize the number
8
9       printf("original value of number: %d\n", number);
10
11      //pass the number by value to cubeByValue
12      number = cubeByValue(number);
13
14      printf("The new value of number: %d\n", number);
15
16  }//main()
17
18  //calculate the cube value of n
19  int cubeByValue(int n) {
20      return n*n*n;
21  }
```

```
original value of number: 4
The new value of number: 64
```

Passing by Value: Example

- in the above code, it passes the variable number to function cubeByValue using call-by-value (line 12).
- The cubeByValue function cubes its argument and passes the new value back to main using a return statement.
- The new value is assigned to number in main (line 12).

Passing Arguments to Functions by Reference

- This is normally accomplished by applying the address operator (&) to the variable (in the caller) whose value will be modified.
- Arrays are not passed using operator & because C automatically passes the starting location in memory of the array
  - the name of an array is equivalent to &arrayName[0]
- When the address of a variable is passed to a function, the indirection operator (*) may be used in the function to modify the value at that location in the caller's memory.

Passing Arguments to Functions by Reference

- * operator used as alias or nickname for variable inside of function

```
void fun1 (int *number) {
    *number = 2 * (*number);
}
```

- *number used as nickname for the variable passed

Passing by Reference: Example

```
3   void cubeByReference(int *pn); // prototype
4
5   void main() {
6
7       int number = 5; // inicialize the number
8
9       printf("original value of number: %d\n", number);
10
11      //pass the number by reference to cubeByReference
12      cubeByReference(&number);
13
14      printf("The new value of number: %d\n", number);
15
16  }//main()
17
18  //calculate the cube value of number by modifing it
19  void cubeByReference(int *pn) {
20      *pn = *pn * *pn * *pn;
21  }
```

Passing by Reference: Example

```
3   void cubeByReference(int *pn); // prototype
4
5   void main() {
6
7       int number = 5; // inicialize the number
8
9       printf("original value of number: %d\n", number);
10
11      //pass the number by reference to cubeByReference
12      cubeByReference(&number);
13
14      printf("The new value of number: %d\n", number);
15
16  }//main()
17
18  //calculate the cube value of number by modifing it
19  void cubeByReference(int *pn) {
20      *pn = *pn * *pn * *pn;
21  }
```

```
original value of number: 5
The new value of number: 125
```

Passing by Reference: Example

- in the above code, it passes the variable number using call-by-reference (line 12)
- the address of number is passed to the function cubeByReference.
- Function cubeByReference takes as a parameter a pointer to an int called pn (line 19).
- The function dereferences the pointer and cubes the value to which pn points (line 20), then assigns the result to *pn (which is really number in main), thus changing the value of number in main.

Example: Swapping Two Values

Example: Swapping Two Values

```
 3  void swap(int *x, int *y){
 4      int temp = *x;
 5      *x = *y;
 6      *y = temp;
 7  }//swap()
 8
 9  void main(){
10      int a,b;
11      printf("Enter number 1: ");
12      scanf("%d", &a);
13      printf("Enter number 2: ");
14      scanf("%d", &b);
15      printf("Before swapping: %d, %d\n", a, b);
16      swap(&a, &b);
17      printf("After swapping: %d, %d\n", a, b);
18  }//main()
```

Example: Swapping Two Values

```c
void swap(int *x, int *y){
    int temp = *x;
    *x = *y;
    *y = temp;
}//swap()

void main(){
    int a,b;
    printf("Enter number 1: ");
    scanf("%d", &a);
    printf("Enter number 2: ");
    scanf("%d", &b);
    printf("Before swapping: %d, %d\n", a, b);
    swap(&a, &b);
    printf("After swapping: %d, %d\n", a, b);
}//main()
```

```
Enter number 1: 3
Enter number 2: 12
Before swapping: 3, 12
After swapping: 12, 3
```

Recap…

Type /element	Variable	Pointer Variable
&	Address operator -return address of operand	Address operator -return address of operand
*	-	Indirection/dereferencing operator -return value of whatever operand pointed to
Declaration/Define	int a;	int *aPtr;

Initialization	int a = 5;	int *aPtr = &a;
		int *aPtr = NULL;
		int *aPtr = 0;

Verify address	&a	&aPtr //address of pointer
Verify content	a	aPtr//address of variable a
Value pointed	a	*aPtr//value of variable a