Fundamentals of Programming
CCS1063/CSE1062
Lecture 11 –Pointers in C

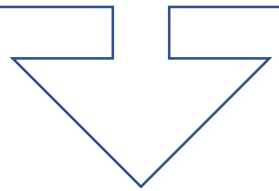
Professor Noel Fernando

UCSC

Variables

int mark = 67;

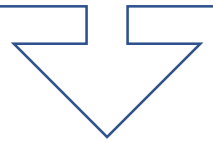
What the declaration tells the C compiler?



Variables

int mark = 67;

What the declaration tells the C compiler?



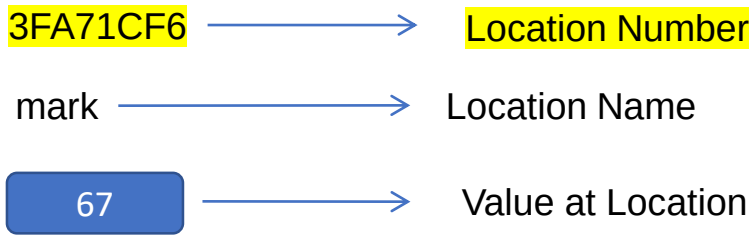
- Reserve space in memory to hold the integer value
- Associate the name “mark” with this memory location
- Store the value 67 at this location

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



Memory Map

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

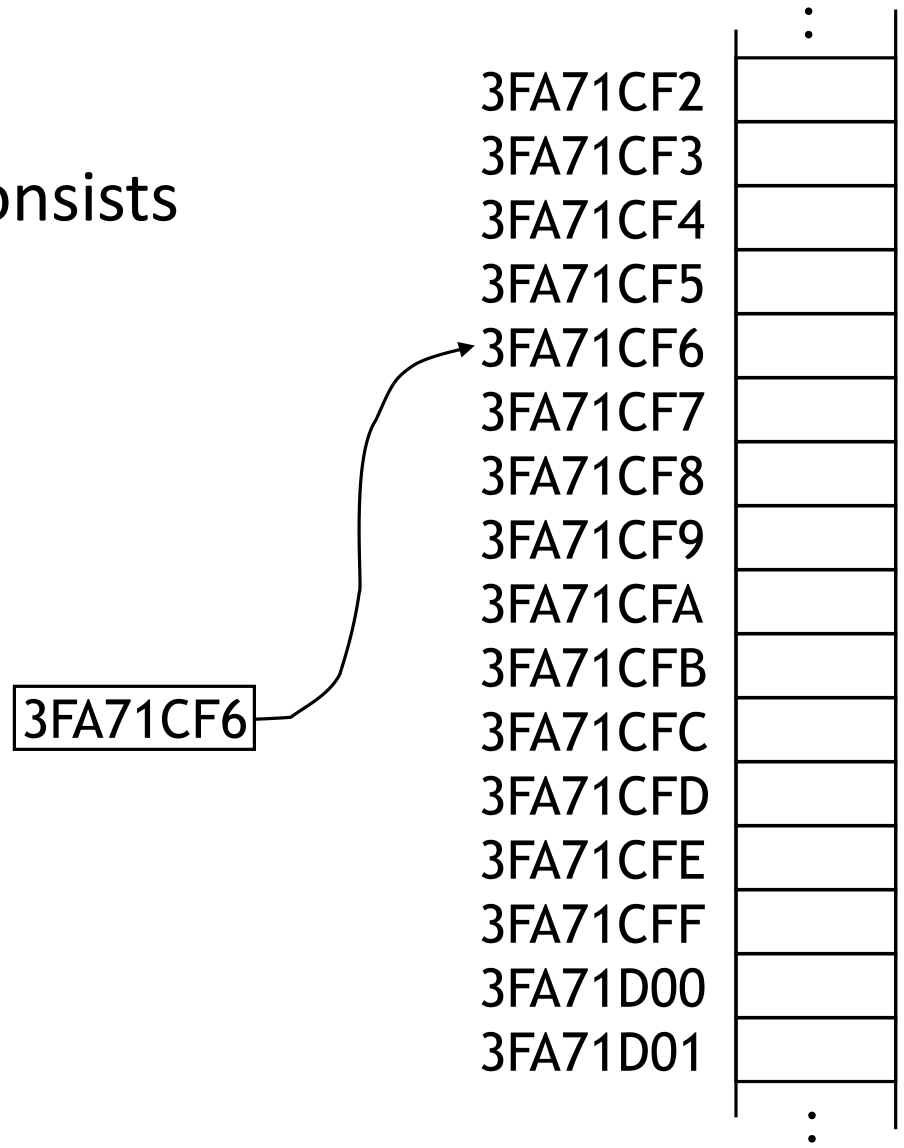
a character

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!

C – Variables and pointers

- Some C programming tasks are performed more easily with pointers.
- It includes dynamic memory allocation.
- Every variable is a memory location and every memory location has its address defined which can be accessed using and (&) operator, which denotes an address in memory.

E.g. Variables and address of the variable

```
#include <stdio.h>
int main () {
int var1; char var2[10];
 printf("Address of var1 variable: %x\n", &var1 );
 printf("Address of var2 variable: %x\n", &var2 );
return 0;
 }
```

- When the above code is compiled and executed, it produces the following result –
- Address of var1 variable: bff5a400
- Address of var2 variable: bff5a3f6

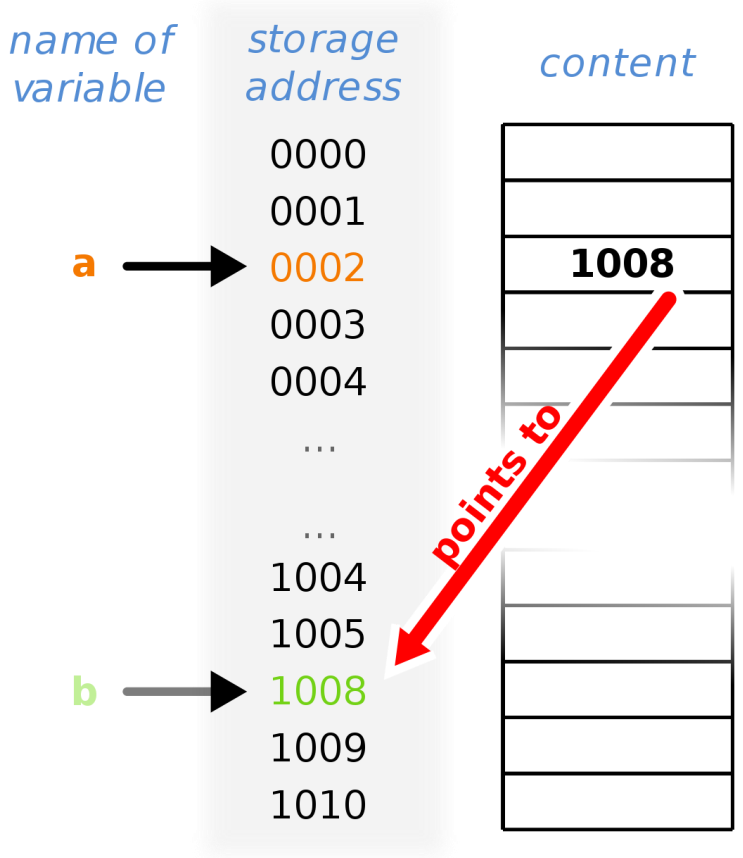
What is a Pointer?

- Variable which contains the address of another variable or
- Pointers (pointer variables) are special variables that are used to store addresses rather than values.
- 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.

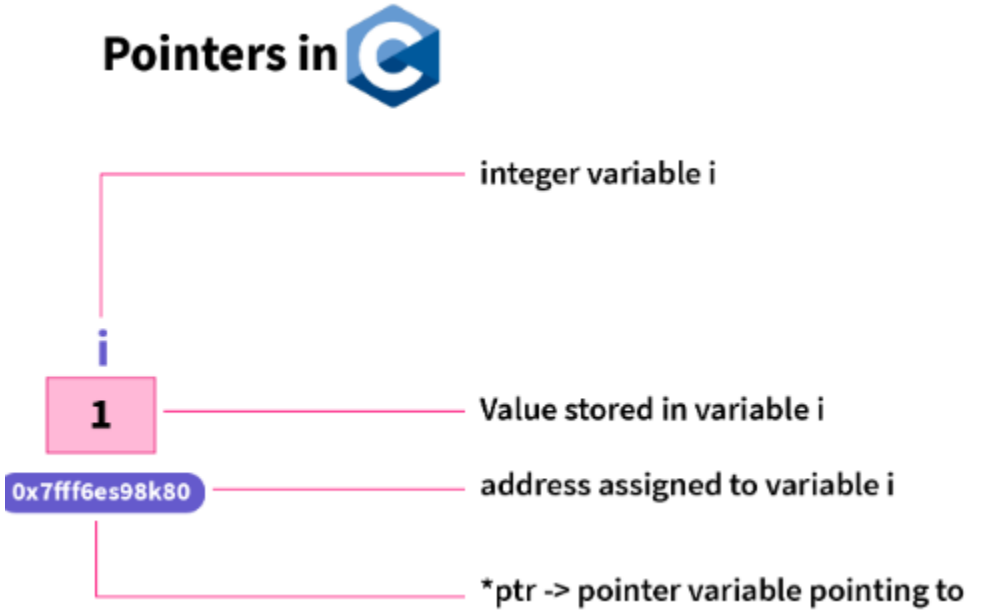


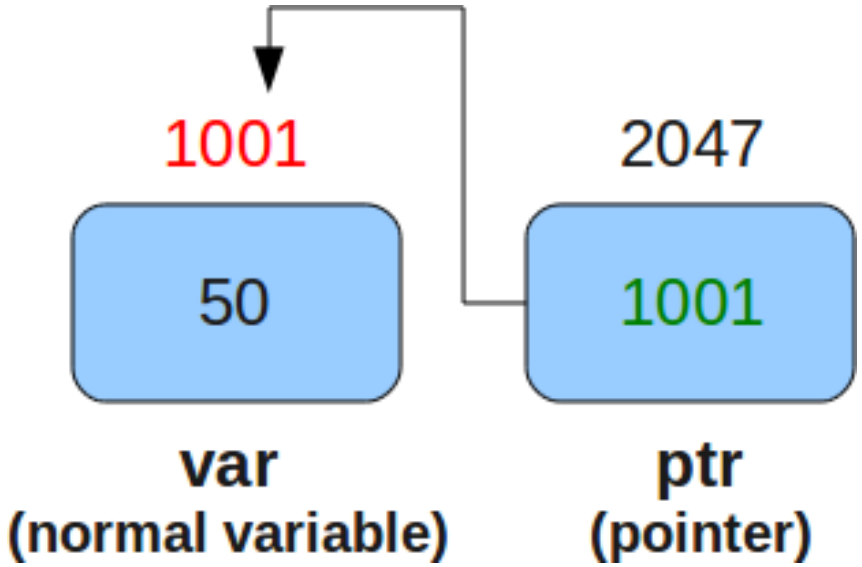
Why pointers are useful?

- Pointers save memory space. Describe how?
- Execution time with pointers is faster because data are manipulated with the address, that is, direct access to memory location.
- Memory is accessed efficiently with the pointers. The pointer assigns and releases the memory as well. Hence it can be said the Memory of pointers is dynamically allocated.
- Pointers are used with data structures. They are useful for representing two-dimensional and multi-dimensional arrays.
- An array, of any type, can be accessed with the help of pointers, without considering its subscript range.
- Pointers are used for file handling.
- Pointers are used to allocate memory dynamically.

What are Pointers?

- A **pointer** is a variable whose value is the address of another variable,
- i.e., direct address of the memory location.
- The general form of a pointer variable declaration is –
- type *var-name;





Examples of valid pointer variable declaration

- int *ip; /* pointer to an integer */
- double *dp; /* pointer to a double */
- float *fp; /* pointer to a float */
- char *ch /* pointer to a character */
- All the above are examples of pointer variable declarations.

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)

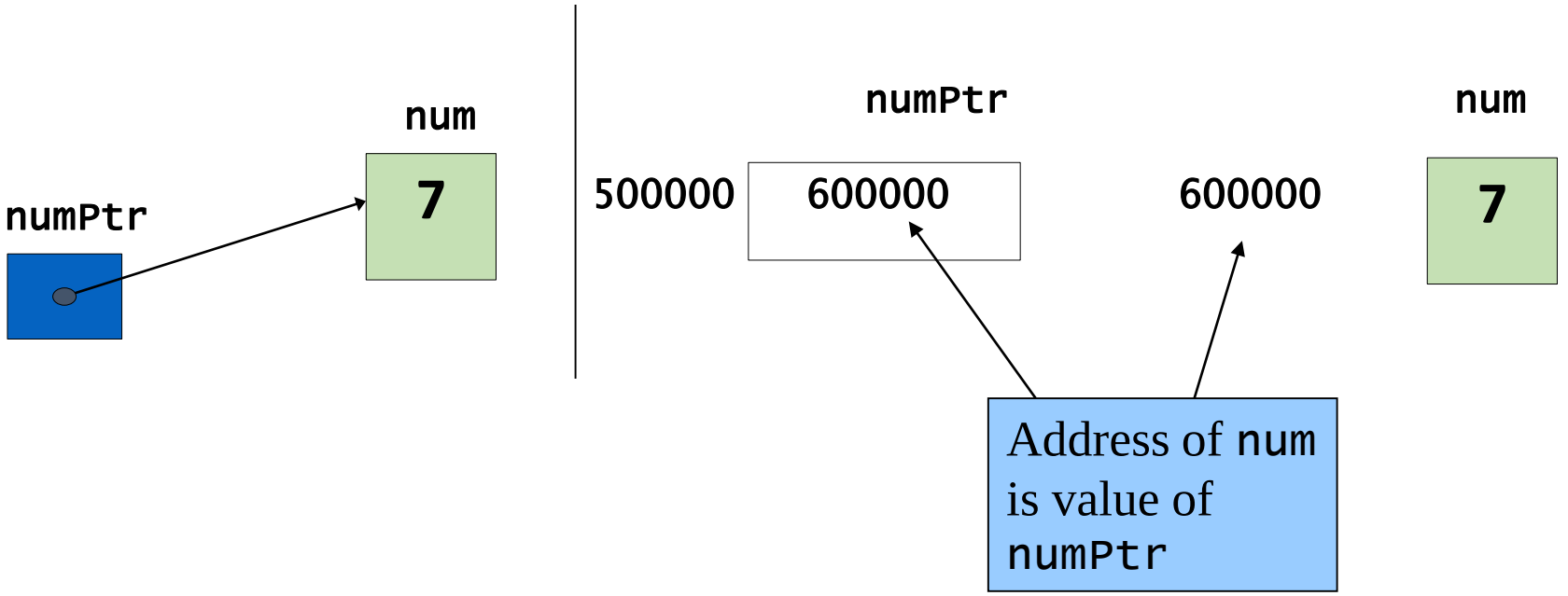
  int *numPtr = NULL; or int *numPtr = 0;

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




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

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.

- **What is Indirection operator?**
- Example

int x; int *p

In this case, the compiler is informed by the asterisk that "p is not an integer,

but rather a reference to a place in memory that stores an integer" (int x; int *p).

Here, it is a component of a pointer declaration rather than a dereference.

Creating an integer variable

- Consider the following:

```
int num = 10;
```

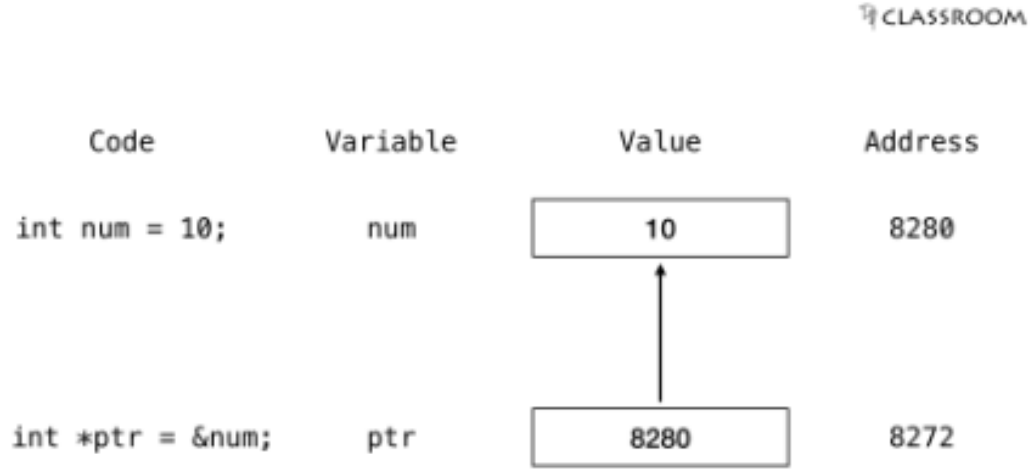
Three things will happen for the above line of code.

- A memory location is located to store integer value.
- The value 10 is saved in that memory location.
- We can refer the memory location using the variable name num

- Creating integer pointer variable
  - To get the address of a variable we use the **address of** & operator.

```
int *ptr = &num;
```

We can represent the integer variable `num` and pointer variable `ptr` as follows.



Updating the value of a variable via pointer

```
// updating the value of num via ptr
*ptr = 20;
```

Complete code:

```
#include <stdio.h>
int main(void) {
// num variable
int num = 10;
// ptr pointer variable
 int *ptr = NULL;
// assigning the address of num to ptr
ptr = &num;
// printing the value of num - Output: 10
printf("num: %d\n", num);
 printf("num via ptr: %d\n", *ptr);
 // updating the value of num via ptr
printf("Updating value of num via ptr...\n");
*ptr = 20;
// printing the new value of num - Output: 20
 printf("num: %d\n", num);
printf("num via ptr: %d\n", *ptr);
return 0;
}
```

- OUTPUT

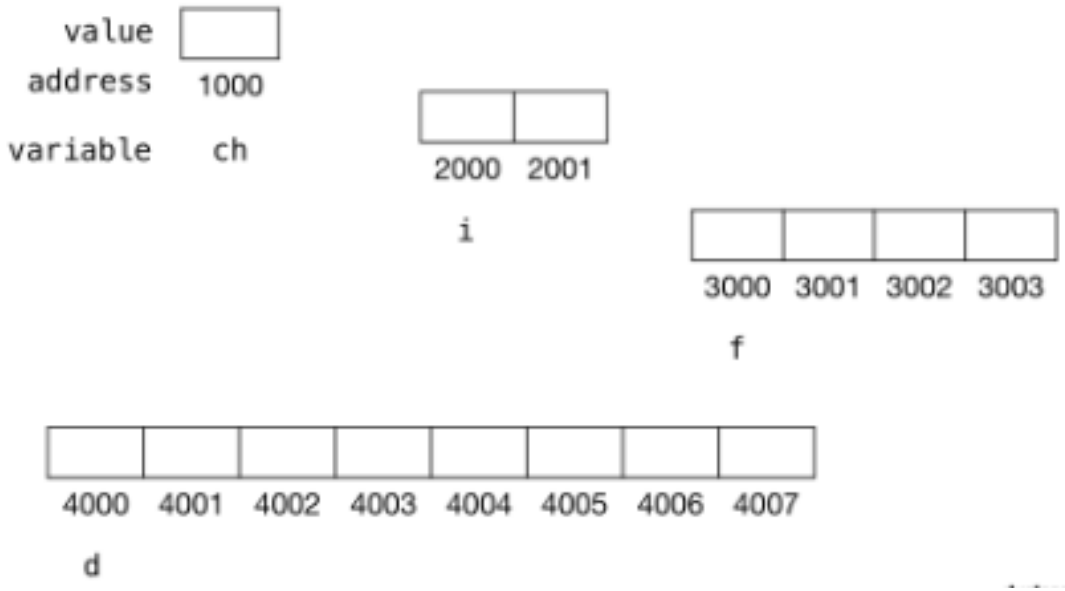
```
num: 10
num via ptr: 10
Updating value of num via ptr...
num: 20
num via ptr: 20
```

Creating variables

- We will consider the following four variables of data types char, int, float and double

Data Type	Variable	Memory Size
char	`char ch = 'a';`	1 byte
int	`int i = 10;`	2 bytes
float	`float f = 12.34;`	4 bytes
double	`double d = 12.3456;`	8 bytes

- We can represent the above variables as follows.



Size of pointer variables

Pointer variables stores the address of other variables.

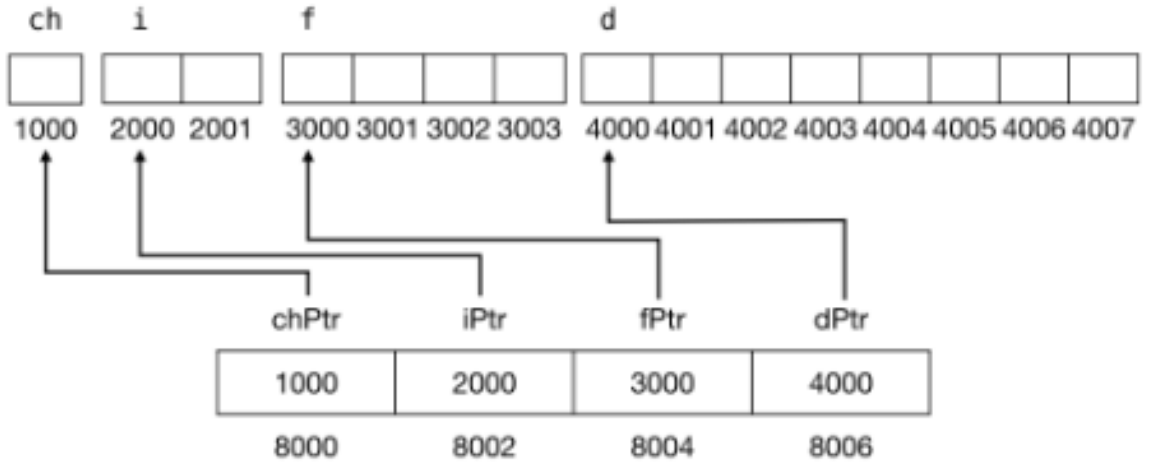
And these addresses are integer value.

We can use the sizeof() operator to find the size of the pointer variable

- Pointers for the variables

```
char *chPtr = &ch;
int *iPtr = &i;
float *fPtr = &f;
double *dPtr = &d;
```

- We can represent the above pointer variables as follows:

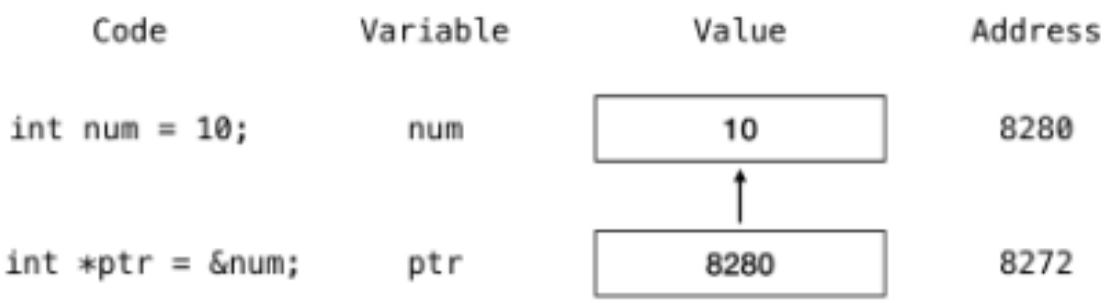


Creating integer pointer variable for the integer variable

- Consider the following program segment

```
// integer variable num
int num = 10;

// pointer variable pointing at num
int *ptr = &num;
```

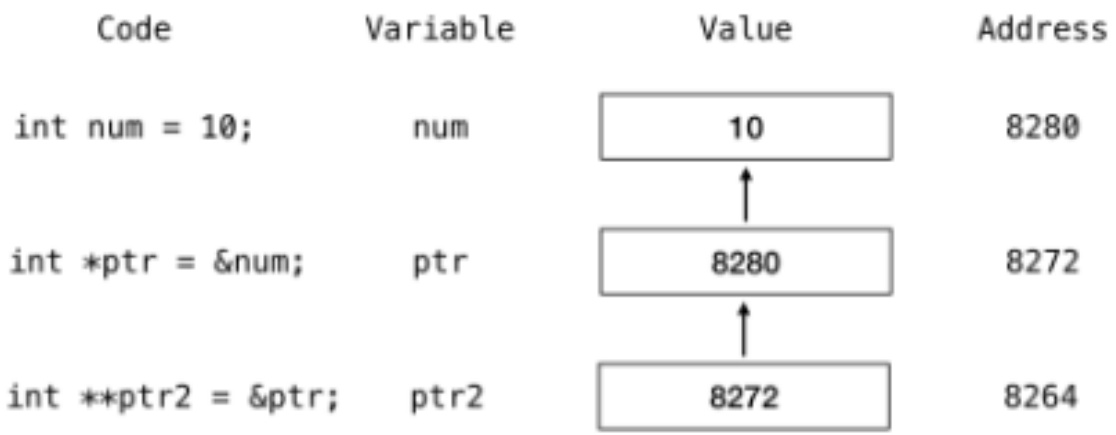
- The above variable creation and pointing can be represented as follows:



Creating the second integer pointer variable for the first integer pointer variable

- To create a second pointer variable to point at the first pointer variable we use the following syntax.
- dataType **secondPtr = &firstPtr
- Consider the following example
- Int num=10;
- Int *ptr = #
- int **ptr2 = &ptr;

- The above variable creation and pointing can be represented as follows:



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

Write-down the output of the following program:

```
#include <stdio.h>

int main() {

  int myAge = 43;  // An int variable

  int* ptr = &myAge;  // A pointer variable, with the name ptr, that stores the address of myAge

  // Output the value of myAge (43)

  printf("%d\n", myAge);

  // Output the memory address of myAge (0x7ffe5367e044)

  printf("%p\n", &myAge);

// Output the memory address of myAge with the pointer (0x7ffe5367e044)
  printf("%p\n", ptr);
  return 0; }
```

Answer :Write-down the output of the following program:

```
#include <stdio.h>

int main() {

  int myAge = 43;  // An int variable

  int* ptr = &myAge;  // A pointer variable, with the name ptr, that stores the address of myAge

  // Output the value of myAge (43)

  printf("%d\n", myAge);

  // Output the memory address of myAge (0x7ffe5367e044)

  printf("%p\n", &myAge);

  // Output the memory address of myAge with the pointer (0x7ffe5367e044)

  printf("%p\n", ptr);

  return 0; }
```

Output:

43
0x7ffe5367e044
0x7ffe5367e044

Write-down the output of the following program segment

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

-

Answer : Write-down the output of the following program segment Changing Value Pointed by Pointers

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

Another example - Changing Value Pointed by Pointers

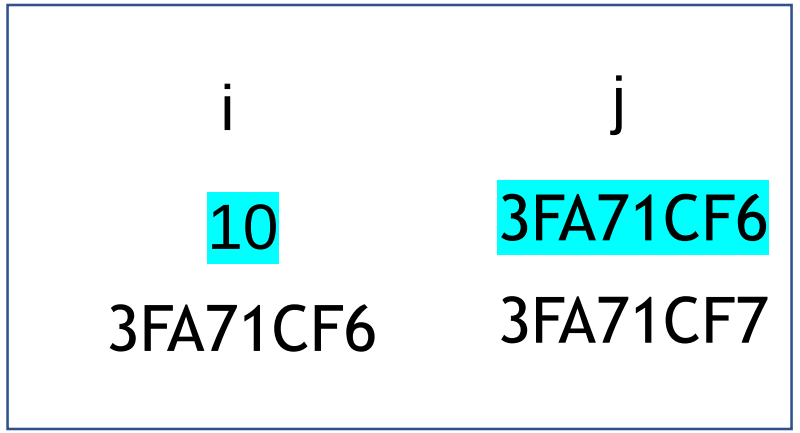
```
Int* pc, c;
 c = 5;
pc = &c;
 *pc = 1;
printf("%d", *pc); // Ouptut: 1
 printf("%d", c); // Output: 1
```

Address of an Address…

Code Snippet

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

Memory Map



Ex. : Write-down the output of the following program segment

```
#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;
 }
```

Answer : Write-down the output of the following program segment

Example : Working of Pointers

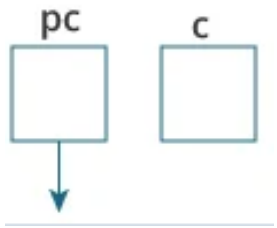
```
#include <stdio.h>
int main() {
int* pc, c;
c = 22;
printf("Address of c: %p\n", &c);
printf("Value of c: %d\n\n", c); // 22
pc = &c;
printf("Address of pointer pc: %p\n", pc);
printf("Content of pointer pc: %d\n\n", *pc); // 22
c = 11;
printf("Address of pointer pc: %p\n", pc);
printf("Content of 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: 2686784
Value of c: 22
Address of pointer pc: 2686784
 Content of pointer pc: 22
Address of pointer pc: 2686784
Content of pointer pc: 11
 Address of c: 2686784
Value of c: 2
```

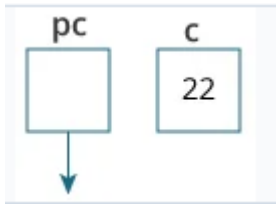
Explanation of the program

1. int* pc, c;



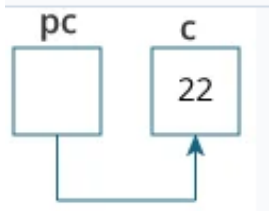
- pc is a pointer variable and c is a normal variable

2. c = 22;



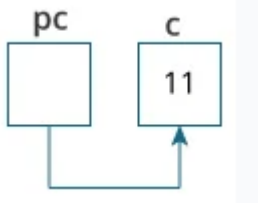
- This assigns 22 to the variable c, That is, 22 is stored in the memory location of variable c.

3. pc = &c;



This assigns the address of variable c to the pointer pc.

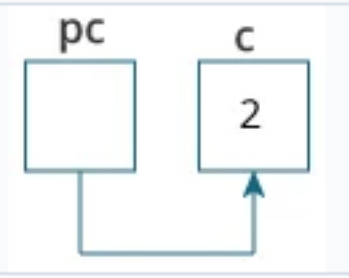
4. C=11;



This assigns 11 to variable

Explanation of the program

5. *pc = 2;



- This change the value at the memory location pointed by the pointer pc to 2

Write the output of the following program:

```
 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 ess 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
```

C - Pointer arithmetic

- You can perform arithmetic operations on a pointer just as you can on a numeric value.
- There are four arithmetic operators that can be used on pointers: ++, --, +, and –
- Example
- let us consider that **ptr** is an integer pointer which points to the address 1000.
- Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer
- ptr++
- After the above operation, the **ptr** will point to the location 1004 because each time ptr is incremented, it will point to the next integer location which is 4 bytes next to the current location.

Question : Write-down the output of the following program segment

```
#include <stdio.h>
const int MAX = 3;
int main () {
 int  var[] = {10, 100, 200};
int  i, *ptr;   /* let us have array address in pointer */
   ptr = var;
for ( i = 0; i < MAX; i++) {
 printf("Address of var[%d] = %x\n", i, ptr );
 printf("Value of var[%d] = %d\n", i, *ptr );      /* move
to the next location */
 ptr++;
}
return 0;
}
```

Answer : Write-down the output of the following program segment

Incrementing a Pointer

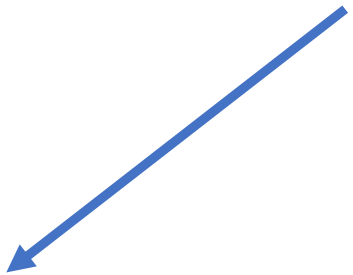
```
#include <stdio.h>
const int MAX = 3;
int main () {
 int  var[] = {10, 100, 200};
int  i, *ptr;   /* let us have array address in pointer */
   ptr = var;
for ( i = 0; i < MAX; i++) {
 printf("Address of var[%d] = %x\n", i, ptr );
 printf("Value of var[%d] = %d\n", i, *ptr );      /* move
to the next location */
 ptr++;
}
return 0;
}
```

```
Address of var[0] = 614057cc
Value of var[0] = 10
Address of var[1] = 614057d0
Value of var[1] = 100
Address of var[2] = 614057d4
Value of var[2] = 200
```

C - Array of pointers

- Before we understand the concept of arrays of pointers, let us consider the following example, which uses an array of 3 integers

```
#include <stdio.h>
 const int MAX = 3;
int main () {
int  var[] = {10, 100, 200};
 int i;
for (i = 0; i < MAX; i++) {
     printf("Value of var[%d] = %d\n",
i, var[i] );
}
 return 0;
}
```



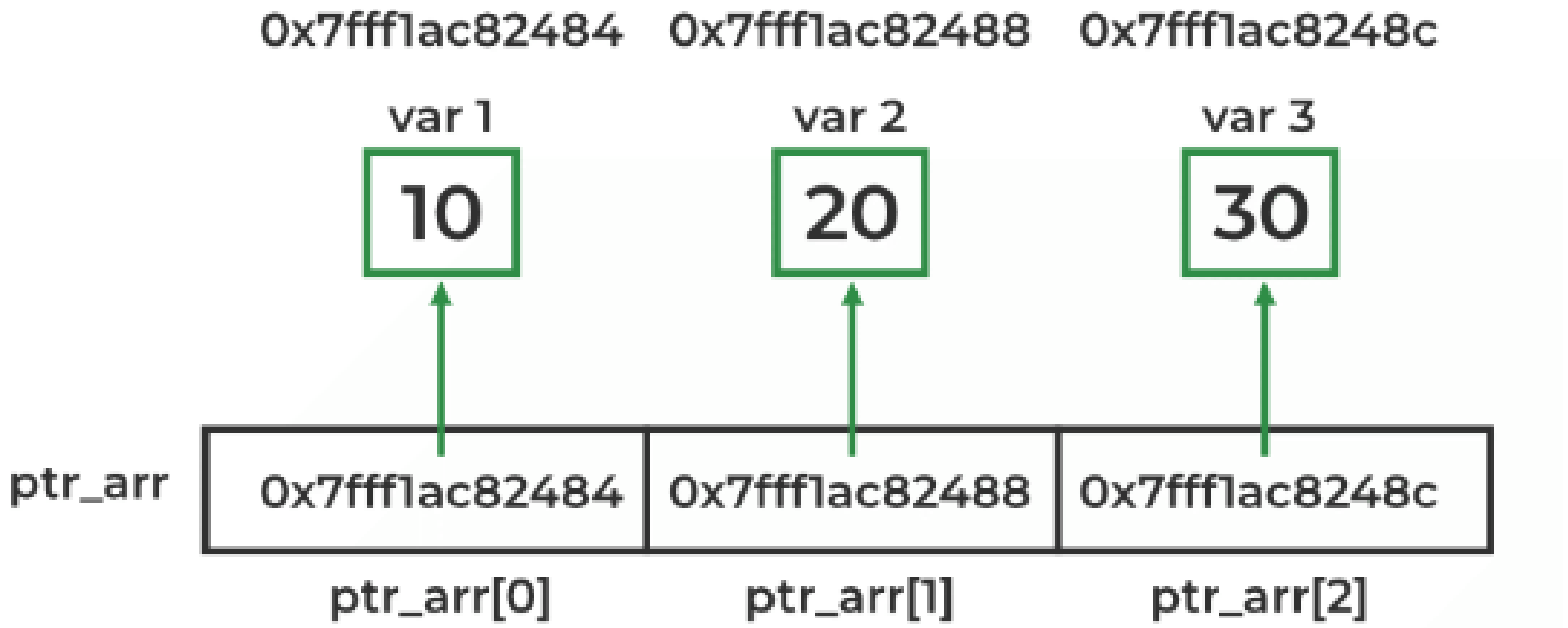
- Output:
- Value of var[0] = 10
- Value of var[1] = 100
- Value of var[2] = 200

C - Array of pointers

```c
#include <stdio.h>

int main()
{
    // declaring some temp variables
    int var1 = 10;
    int var2 = 20;
    int var3 = 30;

    // array of pointers to integers
    int* ptr_arr[3] = { &var1, &var2, &var3 };

    // traversing using loop
    for (int i = 0; i < 3; i++) {
        printf("Value of var%d: %d\tAddress:
%p\n", i + 1, *ptr_arr[i], ptr_arr[i]);
    }

    return 0;
}
```

- **Output**
- Value of var1: 10
  Address: 0x7fff1ac82484
  Value of var2: 20
  Address: 0x7fff1ac82488
  Value of var3: 30
  Address: 0x7fff1ac8248c

Explanation



C - Pointer to Pointer

- A pointer to a pointer is a form of multiple indirection, or a chain of pointers.
- Normally, a pointer contains the address of a variable.
- When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown in the diagram.

- For example,
- the following declaration declares a pointer to a pointer of type int –

```
int **var;
```

Example: Pointer to Pointer

```
#include <stdio.h>
int main () {
int var;
int *ptr;
int **pptr;
 var = 3000;
/* take the address of var */
ptr = &var;
/* take the address of ptr using address of operator & */
pptr = &ptr;
/* take the value using pptr */
printf("Value of var = %d\n", var );
printf("Value available at *ptr = %d\n", *ptr );
printf("Value available at **pptr = %d\n", **pptr);
return 0;
 }
```

Output:

Value of var = 3000

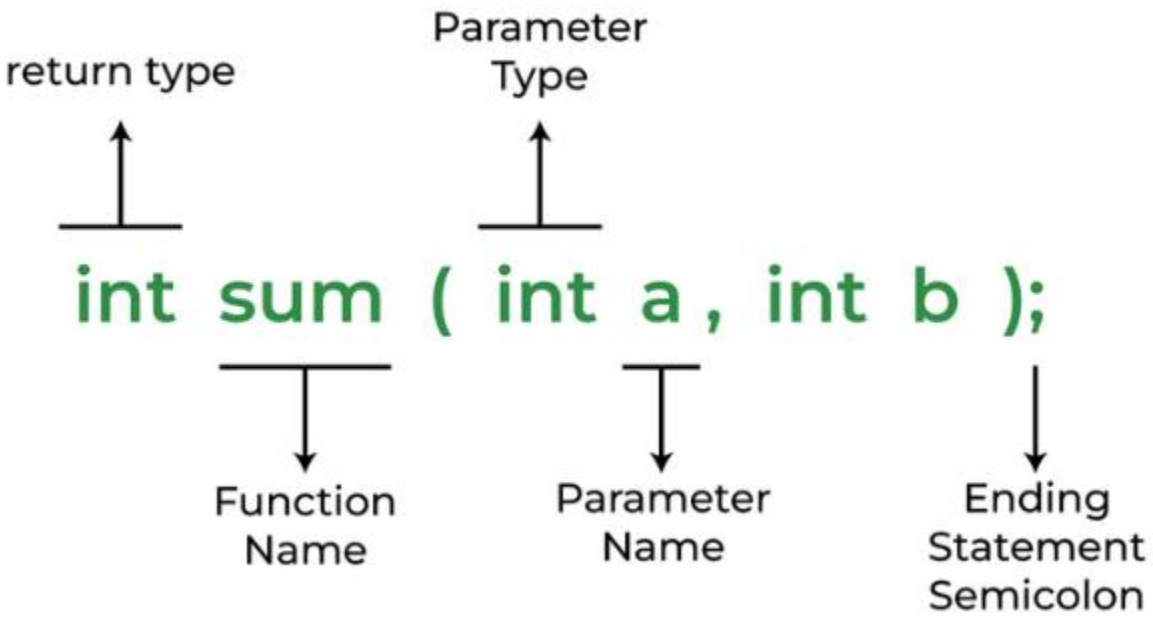
Value available at *ptr = 3000

Value available at **pptr = 3000

Passing Pointers to Functions in C

- Before learning Passing Pointers to Functions in C, we should have knowledge of the following:
- Pointers in C (already discussed)
- Functions in C (previously did)

- `int sum(int a, int b);`
  `int sum(int , int);`



Passing Pointers to Functions in C

- First we can study how to pass arguments to function without a pointer:
- E.g.

```
How to write C program to swap two values without
passing pointer to swap function.
```

Passing Pointers to Functions in C

- First we can study how to pass arguments to function without a pointer:
- E.g.

  How to write C program to swap two values without passing pointer to swap function.

```
#include <stdio.h>

void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

// Driver code
int main()
{
  int a = 10, b = 20;
  swap(a, b);
  printf("Values after swap function
are: %d, %d",
          a, b);
  return 0;
}
```

Arguments Passing with pointers

- A pointer is passed instead of a variable and its address is passed instead of its value.
- As a result, any change made by the function using the pointer is permanently stored at the address of the passed variable.

```
#include <stdio.h>
 void swap(int* a, int* b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
// Driver code
int main()
{
  int a = 10, b = 20;
  printf("Values before swap
function are: %d, %d\n",
          a, b);
  swap(&a, &b);
  printf("Values after swap
function are: %d, %d",
          a, b);
  return 0;
}
```

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.