

# **CS100**

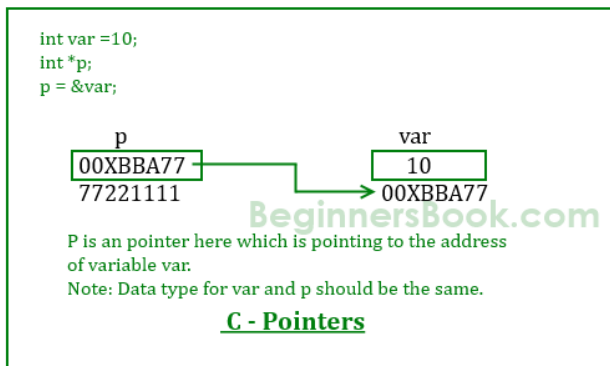
# **Introduction to Programming**

## **Lecture 6. Pointers**

# **Address and Pointer**

# Address of a Variable

- **What is the address in C?**
  - An integer indicating the numerical number of memory storage unit
  - Usually in binary or hexadecimal format



Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00	sum	int (4 bytes)	000000FF (255 <sub>10</sub> )
90000001	00			
90000002	00			
90000003	FF			
90000004	FF	age	short (2 bytes)	FFFF (-1 <sub>10</sub> )
90000005	FF			
90000006	1F			
90000007	FF			
90000008	FF	average	double (8 bytes)	1FFFFFFFFFFFFFFF (4.45015E-308 <sub>10</sub> )
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF			
9000000D	FF			
9000000E	90			
9000000F	00			
90000010	00	ptrSum	int* (4 bytes)	90000000
90000011	00			

Note: All numbers in hexadecimal

# Address Operator (&)

```
#include <stdio.h>
int main(void)
{
    int num = 5;

    printf("num = %d, &num = %p\n", num, &num);
    scanf("%d", &num);
    printf("num = %d, &num = %p\n", num, &num);
    return 0;
}
```

This value is just for **illustration**, and may be different for another run.

## Output:

num = 5, &num = 1024

**10**

num = 10, &num = 1024

# Pointer Variables

- We may have variables which store the **addresses** of memory locations of some data objects. These variables are called **pointers**.
- **A pointer variable is declared by dataType \*pointerName**, for example:





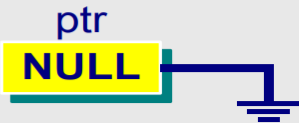
<code>int *ptrI;</code>	<code>/* Variable ptrI is a pointer. It stores the address of a memory location for an integer */</code>
<code>float *ptrF;</code>	<code>/* Variable ptrF is a pointer. It stores the address of a memory location for a float */</code>
<code>char *ptrC;</code>	<code>/* Variable ptrC is a pointer. It stores the address of a memory location for a char */</code>

- The **value** of a pointer variable is an **address**.

# Pointer Variables

## Example:

```
int a = 20; float b = 40.0; char c = 'a';  
int *ptrI; float *ptrF; char *ptrC;  
ptrI = &a; ptrF = &b; ptrC = &c;
```

Statement	Operation
int *ptrI	 Uninitialized Pointer
ptrI = &a;	 Address = 1000
ptrF = &b;	 Address = 2000
ptrC = &c;	 Address = 3000
int *ptr = NULL;	

# Pointer To Pointer

- A pointer storing the value of another pointer



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

```
int main(void)
{
    float num=12;
    float* p1=&num;
    float** p2=&p1;

    return 0;
}
```

# Indirection Operators (\*)



- The **content** of the memory location pointed to by a pointer variable is referred to by using the **indirection operator \***.
- If a pointer variable is defined as **ptr**, we use the expression **\*ptr** to dereference the pointer to obtain the value stored at the address pointed to by the pointer **ptr**.



# Indirection Operator – Example 1

```
#include <stdio.h>
int main(void)
{
    int num = 3;
    int *ptr;
```

```
    ptr = &num;
    printf("num = %d, &num = %p\n", num, &num);
    printf("ptr = %p, *ptr = %d\n", ptr, *ptr);
    *ptr = 10;
    printf("num = %d, &num = %p\n", num, &num);
    return 0;
}
```

Statement	Operation
<code>ptr = &amp;num;</code>	 ptr: 1024 → num: 3 Address = 1024
<code>*ptr = 10;</code>	 ptr: 1024 → num: 10 Address = 1024

## Output:

```
num = 3, &num = 1024
ptr = 1024, *ptr = 3
num = 10, &num = 1024
```

# Indirection Operator – Example 2

```
/* example to show the use of pointers */
#include <stdio.h>

int main(void)
{
    int num1 = 3, num2 = 5;
    int *ptr1, *ptr2;

    ptr1 = &num1; // put the address of num1 into ptr1
    printf("num1 = %d, *ptr1 = %d\n", num1, *ptr1);

    (*ptr1)++; /* increment by 1 the content of the
                memory location pointed to by ptr1 */
    printf("num1 = %d, *ptr1 = %d\n", num1, *ptr1);

    ptr2 = &num2; // put the address of num2 into ptr2
    printf("num2 = %d, *ptr2 = %d\n", num2, *ptr2);
}
```

Code continues in next slide ...

## Output:

```
num1 = 3, *ptr1 = 3
num1 = 4, *ptr1 = 4
num2 = 5, *ptr2 = 5
```

```
*ptr2 = *ptr1; /* copy the content of the location
                pointed to by ptr1 into the
                location pointed to by ptr2 */
printf("num2 = %d, *ptr2 = %d\n", num2, *ptr2);

*ptr2 = 10; /* 10 is copied into the location
            pointed to by ptr2 */
num1 = *ptr2; /* copy the content of the memory
              location pointed to by ptr2
              into num1 */
printf("num1 = %d, *ptr1 = %d\n", num1, *ptr1);

*ptr1 = *ptr1 * 5;
printf("num1 = %d, *ptr1 = %d\n", num1, *ptr1);

ptr2 = ptr1; // address in ptr1 copied into ptr2
printf("num2 = %d, *ptr2 = %d\n", num2, *ptr2);

return 0;
}
```

**Output:**

```
num2 = 4, *ptr2 = 4
num1 = 10, *ptr1 = 10
num1 = 50, *ptr1 = 50
num2 = 10, *ptr2 = 50
```

# Indirection Operator – Example 2

Statement	num1 (addr = 1024)	num2 (addr = 2048)	ptr1	ptr2
int num1 = 3, num2 = 5;	3	5		
int *ptr1, *ptr2;	3	5	?	?
ptr1 = &num1;	3	5	1024	?
(*ptr1)++;	4	5	1024	?
ptr2 = &num2;	4	5	1024	2048
*ptr2 = *ptr1;	4	4	1024	2048
*ptr2 = 10;	4	10	1024	2048
num1 = *ptr2;	10	10	1024	2048
*ptr1 = *ptr1 * 5;	50	10	1024	2048
ptr2 = ptr1;	50	10	1024	1024

# Multiple Indirection



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

```
int main(void)
```

```
{
```

```
    float num=12;
```

```
    float* p1=&num;
```

```
    float** p2=&p1;
```

```
    printf("the content with indirection once: %f", *p1);
```

```
    printf("the content with indirection twice: %f", *(*p2));
```

```
    return 0;
```

```
}
```

# How function is called?

- **Entry point**

- the first instruction a program is executed
- In C/C++, the main() function

```
int main(void);  
int main();  
  
int main(int argc, char **argv);
```

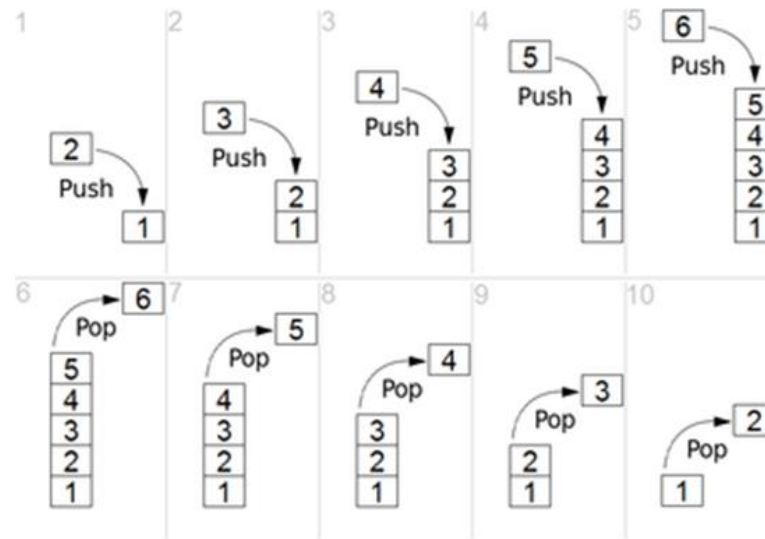
- Loader of operating system will load the program into memory, giving the entry point a specific address
- This marks the transition from load time to run time.

# How function is called?

- **Function (call) stack**

- What is a stack?

- A data structure to store data in a first-in-last-out order
- Two operations:
  - push (into the stack) / pop (out of the stack)



# How function is called?

- **Function call process**

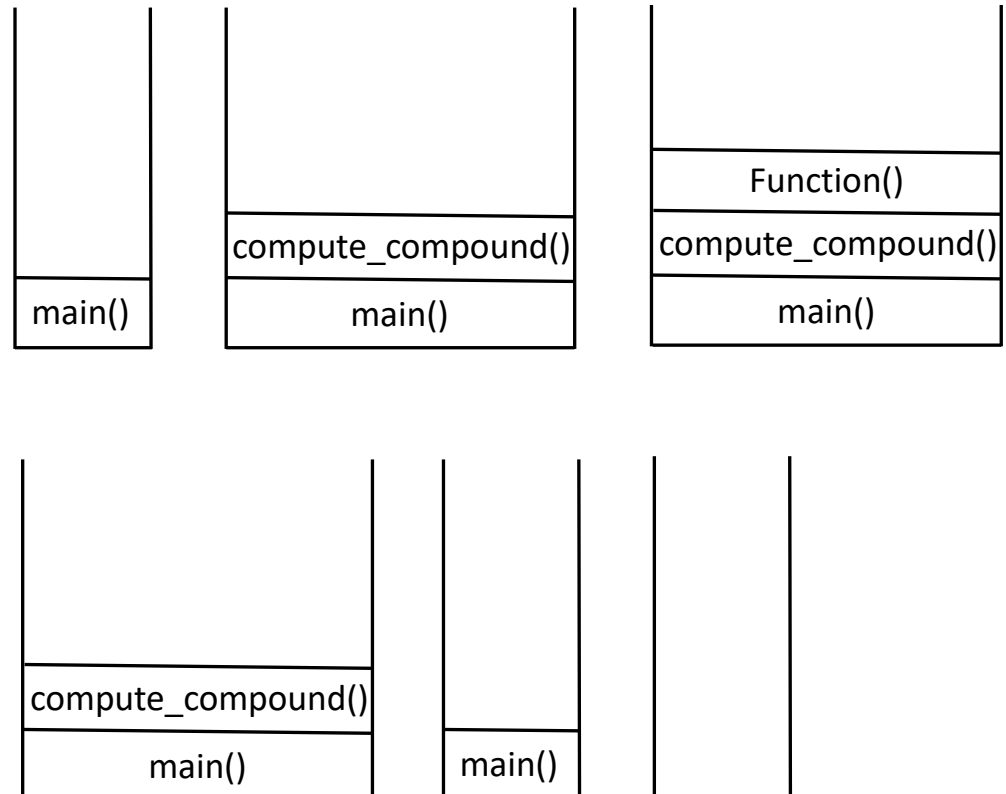
- With the aid of function call stack, storing pointers

```
#include <stdio.h>
float compute_compound(float x);
float function(float x);

void main()
{
    float f=compute_compound(10.0);
    printf("the result is: %f\n", f);
    return 0;
}

float compute_compound()
{
    return exp(function(x));
}

float function(float x)
{
    return sin(x)*sin(x);
}
```





# Call by Pointer

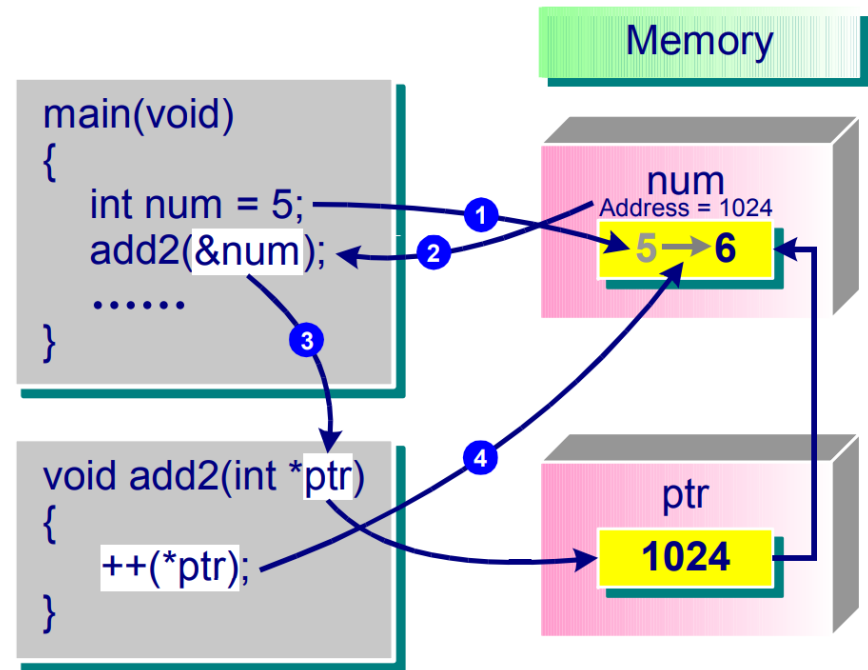
- **Parameter passing between functions has two modes:**
  - Call by value
  - Call by pointer
- **Call by value:** The argument of a function has a local copy when it is passed to the function.
- **Call by pointer:** The argument of a function shares the same address of the argument variable, no argument copy is performed.
  - Therefore, a change to the value pointed to by the parameter **changes** the argument value (instantly).

# Call by Pointer – Example 1

```
#include <stdio.h>
void add2(int *ptr);
int main(void)
{
    int num = 5;
    // passing the address of num
    add2(&num);
    printf("Value of num is: %d",
           num);
    return 0;
}

void add2(int *ptr)
{
    ++(*ptr);
}
```

**Output:**  
Value of num is: 6



# Call by Pointer – Example 2

```
#include <stdio.h>

void function1(int a, int *b);
void function2(int c, int *d);
void function3(int h, int *k);

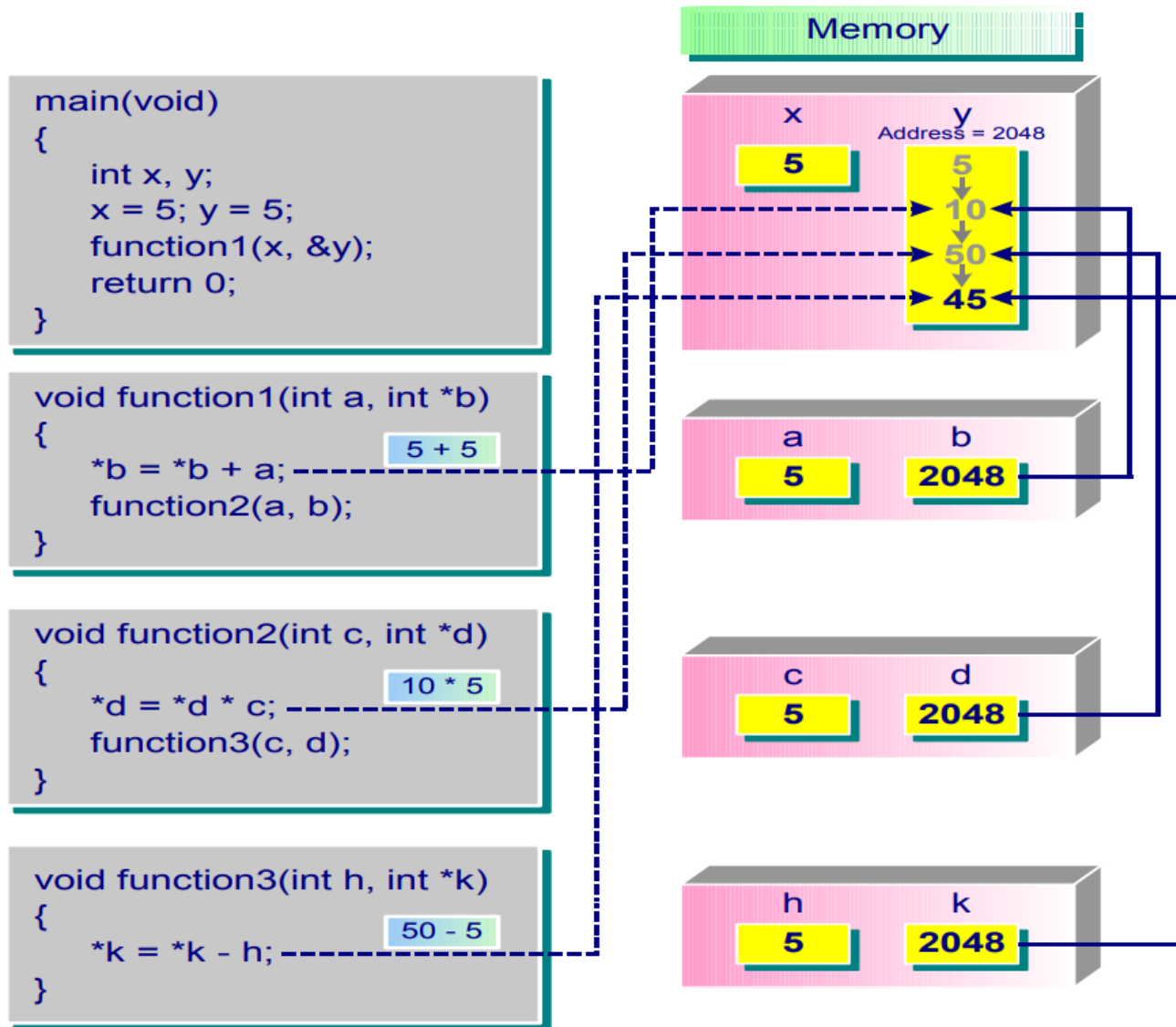
int main(void) {
    int x, y;
    x = 5; y = 5;           /* (i) */
    function1(x, &y);       /* (x) */
    return 0;
}

void function1(int a, int *b) { /* (ii) */
    *b = *b + a;               /* (iii) */
    function2(a, b);           /* (ix) */
}

void function2(int c, int *d) { /* (iv) */
    *d = *d * c;               /* (v) */
    function3(c, d);           /* (viii) */
}

void function3(int h, int *k) { /* (vi) */
    *k = *k - h;               /* (vii) */
}
```

# Call by Pointer – Example 2



# Call by Pointer – Example 2

	x	y	a	*b	c	*d	h	*k	remarks
(i)	5	5	-	-	-	-	-	-	
(ii)	5	5	5	5	-	-	-	-	
(iii)	5	10	5	10	-	-	-	-	
(iv)	5	10	5	10	5	10	-	-	
(v)	5	50	5	50	5	50	-	-	
(vi)	5	50	5	50	5	50	5	50	
(vii)	5	45	5	45	5	45	5	45	
(viii)	5	45	5	45	5	45	-	-	
(ix)	5	45	5	45	-	-	-	-	
(x)	5	45	-	-	-	-	-	-	

# When to Use “Call by Pointer”?

- When you need to pass **more than one value back** from a function.
- When using *call by value*, it results in a **large piece of information** being **copied** to the local memory, e.g. passing large arrays.
  - In such cases, for the sake of **efficiency**, we’d better use call by reference.

# Call by Reference

- **Call by reference**: The argument of a function is another name of the same variable, no argument copy is performed.
  - Therefore, a change to the value of the parameter **changes** the argument value (instantly).
  - Declaration of a reference variable

```
float a=10.0;
float& b=a; //reference variable should be initialized.

b=15;
printf("the value of a is: %f\n",a);
```

# Example: multiple function return values

- Get the area and circumference of a circle

```
#include <stdio.h>
#define PI 3.1415926

void GetCircleInfo(float R, float *area, float *circum);

int main(void) {
    float R=0, area=0, circum=0;
    scanf("Input circle radius: %f", &R);

    GetCircleInfo(R, &area, &circum);
    printf("The circle area is :%f\n", area);
    printf("The circle circumference is :%f\n", area);

    return 0;
}

void GetCircleInfo(float R, float *area, float *circum){
    *area=PI*R*R;
    *circum=2*PI*R;
}
```

*/\*Using pointer implementation\*/*



# Example: multiple function return values

- Get the area and circumference of a circle

```
#include <stdio.h>
#define PI 3.1415926

void GetCircleInfo(float R, float &area, float &circum);

int main(void) {
    float R=0, area=0, circum=0;
    scanf("Input circle radius: %f", &R);

    GetCircleInfo(R, area, circum);
    printf("The circle area is :%f\n", area);
    printf("The circle circumference is :%f\n", area);

    return 0;
}

void GetCircleInfo(float R, float *area, float *circum){
    area=PI*R*R;
    circum=2*PI*R;
}
```

*/\*Using reference implementation\*/*

# Function Pointers

- **A subroutine pointer or procedure pointer**
  - A pointer that points to a function
  - Points to executable code within memory

```
#include <stdio.h> /* for printf */
#include <string.h> /* for strchr */

double cm_to_inches(double cm) {
    return cm / 2.54;
}

// "strchr" is part of the C string handling (i.e., no need for declaration)
// See https://en.wikipedia.org/wiki/C\_string\_handling#Functions

int main(void) {
    double (*func1)(double) = cm_to_inches;
    char * (*func2)(const char *, int) = strchr;
    printf("%f %s", func1(15.0), func2("Wikipedia", 'p'));
    /* prints "5.905512 pedia" */
    return 0;
}
```

# Function Pointer as Function Parameter

```
1 #include <math.h>
2 #include <stdio.h>
3
4 // Function taking a function pointer as an argument
5 double compute_sum(double (*funcp)(double), double lo, double hi) {
6     double sum = 0.0;
7
8     // Add values returned by the pointed-to function '*funcp'
9     int i;
10    for(i = 0; i <= 100; i++) {
11        // Use the function pointer 'funcp' to invoke the function
12        double x = i / 100.0 * (hi - lo) + lo;
13        double y = funcp(x);
14        sum += y;
15    }
16    return sum / 101.0;
17 }
18
19 double square(double x) {
20     return x * x;
21 }
22
```

```
23 int main(void) {
24     double sum;
25
26     // Use standard library function 'sin()' as the pointed-to function
27     sum = compute_sum(sin, 0.0, 1.0);
28     printf("sum(sin): %g\n", sum);
29
30     // Use standard library function 'cos()' as the pointed-to function
31     sum = compute_sum(cos, 0.0, 1.0);
32     printf("sum(cos): %g\n", sum);
33
34     // Use user-defined function 'square()' as the pointed-to function
35     sum = compute_sum(square, 0.0, 1.0);
36     printf("sum(square): %g\n", sum);
37
38     return 0;
39 }
```