COMP133: **COMPUTER AND PROGRAMMING**

Modular Programming - Pointers

Dr. Radi Jarrar
Department of Computer Science
Birzeit University

Pointers & Modular Programming

- Variables are used to store values.
- A pointer, on the other hand, stores an address to another variables.
- A variable reference a value directly.
- A pointer reference a value indirectly.

Pointers

- Pointer: a memory cell that stores the address of a data item.

- Declaration:

```
int a = 5;
int *aPtr = &a;
```

- `*aPtr` is a pointer variable of type integer (meaning it will reference an integer value). In this case, it stores the address of the memory location a.

Example

- `int a = 5;`
- `int *aPtr;`
- `aPtr = &a;  //aPtr gets address of a`
- `aPtr "points to" a`

From the material of Mr. Abdallah Karakra, 2016, Birzeit University.

Example

- `int i = 5;`
- `int *iPtr; // declare a pointer variable`
- `iPtr = &i; // store address-of i to ptr`
- `printf("*iPtr = %d\n", *iPtr);  /* refer to referee of ptr */`

Example

```
#include<stdio.h>
int main(){
    int a = 5, *aPtr = &a;
    printf("The value of a=%d, the value of *aPtr=%d\n", a, *aPtr);
    printf("The value of &a=%p, the value of aPtr=%p\n", &a, aPtr);
    printf("The value of aPtr=%p, the value of &aPtr=%p\n", aPtr, &aPtr);

    printf("The value of *&aPtr=%p, the value of &*aPtr=%p\n", *&aPtr, &*aPtr);

    return 0;
}
```

Memory	
	0x0001
	0x0002
	0x0003
	0x0004
	0x0005

Example

```c
#include <stdio.h>
int main()
{
    int x, *p;
    p = &x;
    *p = 0;
    printf("x is %d\n", x);
    printf("*p is %d\n", *p);
    *p += 1;
    printf("x is %d\n", x);
    (*p)++;
    printf("x is %d\n", x);
    return 0;
}
```

Output

Example

```c
#include <stdio.h>
int main()
{
    int x, *p;
    p = &x;
    *p = 0;
    printf("x is %d\n", x);
    printf("*p is %d\n", *p);
    *p += 1;
    printf("x is %d\n", x);
    (*p)++;
    printf("x is %d\n", x);
    return 0;
}
```

Output

```
x is 0
*p is 0
x is 1
x is 2
```

Example

- `int m = 10, n = 5;`
- `int *mPtr, *nPtr;`
- `mPtr = &m;`
- `nPtr = &n;`
- `*mPtr = *mPtr + *nPtr;`
- `*nPtr = *mPtr - *nPtr;`
- `printf("%d %d\n%d %d\n", m, *mPtr, n, *nPtr);`

Output

Example

```
int m = 10, n = 5;
int *mPtr, *nPtr;
mPtr = &m;
nPtr = &n;
*mPtr = *mPtr + *nPtr;
*nPtr = *mPtr - *nPtr;
printf("%d %d\n%d %d\n", m, *mPtr, n,
 *nPtr);
```

Output

```
15      15
10      10
```

Why do we need pointers?

- As noted previously, functions can only return one value. A function cannot return more than one value.
- For example, if we need a function that takes two numbers and return their sum and subtraction, this cannot be done using return.
- However, we can send a pointer as a parameter to the function, and fill where it reference at (called output parameter).
- This allows functions to return more than one value.

Why do we need pointers?

```c
#include <stdio.h>
int sum(int,int);
int main()
{
    int num1=4,num2=5;
    int result;
    result=sum(num1,num2);
    printf("The result is %d",result);

    return 0;
}
int sum(int x,int y)
{
    return (x+y);
}
```

```c
#include <stdio.h>
void sum(int*,int,int);
int main()
{
    int num1=4,num2=5;
    int result;
    sum(&result,num1,num2);
    printf("The result is %d",result);

    return 0;
}
void sum(int*res,int x,int y)
{
    *res=x+y;
}
```

Call-by-value vs. call-by-reference

```
#include<stdio.h>
int cubeByValue(int x);

int main(){
    int x;

    printf("Enter a number");
    scanf("%d", &x);

    printf("value of x before: %d\n", x);
    printf("x cubic = %d\n", cubeByValue(x));
    printf("value of x after: %d\n", x);
}
int cubeByValue(int x){

    return x*x*x;

}
```

Call-by-value vs. call-by-reference

```
#include<stdio.h>
void cubeByReference(int *x);

int main(){
    int x;

    printf("Enter a number");
    scanf("%d", &x);

    printf("value of x before: %d\n", x);
    cubeByReference( &x );
    printf("value of x after: %d\n", x);
}
void cubeByReference(int *x){

    *x = *x * *x * *x;

}
```

Why do we need pointers?

- Write a function that takes two floats and return their summation, subtraction, multiplication, and division.