

# Programming Methodology - SE1012

## Lab 7

IT23613522

### 1.Pointer Manipulation with Multiple Variables

```
vboxuser@Ubuntu: ~/IT23613522
#include <stdio.h>

void calculate(int *x, int *y, int *z) {
    int ox = *x;
    int oy = *y;
    int oz = *z;
    *x = ox + oy + oz;
    *y = ox - oy;
    *z = ox * oz;
}

int main(void) {
    int x, y, z;
    if (scanf("%d %d %d", &x, &y, &z) != 3) return 1;
    calculate(&x, &y, &z);
    printf("Modified x (sum): %d\n", x);
    printf("Modified y (difference): %d\n", y);
    printf("Modified z (product): %d\n", z);
    return 0;
}
~
~
11,0-1 All
```

```
vboxuser@Ubuntu: ~/IT23613522
vboxuser@Ubuntu:~/IT23613522$ vim s.c
vboxuser@Ubuntu:~/IT23613522$ gcc s.c -o s
vboxuser@Ubuntu:~/IT23613522$ ./s
5
3
2
Modified x (sum): 10
Modified y (difference): 2
Modified z (product): 10
vboxuser@Ubuntu:~/IT23613522$
```

## 2. Functions with Output Parameters

```
vboxuser@Ubunt: ~/IT23613522
#include <stdio.h>

void calculate_circle(float radius, float *circumference, float *area) {
    const float PI = 3.14159f;
    *circumference = 2.0f * PI * radius;
    *area = PI * radius * radius;
}

int main(void) {
    float radius = 5.0f;
    float circumference = 0.0f;
    float area = 0.0f;
    calculate_circle(radius, &circumference, &area);
    printf("Circumference: %.2f\n", circumference);
    printf("Area: %.2f\n", area);
    return 0;
}
```

17,1 All

```
vboxuser@Ubunt: ~/IT23613522
vboxuser@Ubunt:~/IT23613522$ vim s.c
vboxuser@Ubunt:~/IT23613522$ gcc s.c -o s
vboxuser@Ubunt:~/IT23613522$ ./s
Circumference: 31.42
Area: 78.54
vboxuser@Ubunt:~/IT23613522$
```

## 3. Pointer Arithmetic

```
vboxuser@Ubunt: ~/IT23613522
#include <stdio.h>

int main(void) {
    float x = 5.5f;
    float *p = &x;
    float *p_before = p;
    printf("Address before increment: %p\n", (void*)p_before);
    p++;
    printf("Address after increment: %p\n", (void*)p);
    printf("Difference in bytes: %td\n", (char*)p - (char*)p_before);
    return 0;
}
```

12,1 All

```
vboxuser@Ubunt: ~/IT23613522
vboxuser@Ubunt:~/IT23613522$ vim s.c
vboxuser@Ubunt:~/IT23613522$ gcc s.c -o s
vboxuser@Ubunt:~/IT23613522$ ./s
Address before increment: 0x7ffe978215c4
Address after increment: 0x7ffe978215c8
Difference in bytes: 4
vboxuser@Ubunt:~/IT23613522$
```

#### 4. Swapping Values Using Pointers

```
vboxuser@Ubunt: ~/IT23613522
#include <stdio.h>

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

int main(void) {
    int a, b;
    if (scanf("%d %d", &a, &b) != 2) return 1;
    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);
    return 0;
}
```

```
vboxuser@Ubunt: ~/IT23613522
vboxuser@Ubunt:~/IT23613522$ vim s.c
vboxuser@Ubunt:~/IT23613522$ gcc s.c -o s
vboxuser@Ubunt:~/IT23613522$ ./s
5
10
Before swap: a = 5, b = 10
After swap: a = 10, b = 5
vboxuser@Ubunt:~/IT23613522$
```

## 5. Using Function Pointers for Flexible Comparisons

```
vboxuser@Ubunt: ~/IT23613522
#include <stdio.h>
#include <stdlib.h>

int compare(int a, int b) {
    if (a > b) return 1;
    if (b > a) return -1;
    return 0;
}

int compare_absolute(int a, int b) {
    int aa = abs(a);
    int bb = abs(b);
    if (aa > bb) return 1;
    if (bb > aa) return -1;
    return 0;
}

int max(int a, int b, int (*cmp)(int, int)) {
    return cmp(a, b) >= 0 ? a : b;
}

int main(void) {
    int a, b;
    if (scanf("%d %d", &a, &b) != 2) return 1;
    printf("Simple Comparison: max = %d\n", max(a, b, compare));
    printf("Absolute Comparison: max = %d\n", max(a, b, compare_absolute));
    return 0;
}
```

28,1

All

```
vboxuser@Ubunt: ~/IT23613522
vboxuser@Ubunt:~/IT23613522$ vim s.c
vboxuser@Ubunt:~/IT23613522$ gcc s.c -o s
vboxuser@Ubunt:~/IT23613522$ ./s
8
-10
Simple Comparison: max = 8
Absolute Comparison: max = -10
vboxuser@Ubunt:~/IT23613522$
```

## 6. Simplifying a Fraction Using Output Parameters

```
#include <stdio.h>
int gcd(int a, int b) {
    if (a < 0) a = -a;
    if (b < 0) b = -b;
    while (b) {
        int t = a % b;
        a = b;
        b = t;}
    return a;}
void simplify_fraction(int *numerator, int *denominator) {
    if (*denominator == 0) {
        printf("Invalid denominator\n");
        return;}
    if (*numerator == 0) {
        *denominator = 1;
        return;}
    int g = gcd(*numerator, *denominator);
    *numerator /= g;
    *denominator /= g;
    if (*denominator < 0) {
        *denominator = -(*denominator);
        *numerator = -(*numerator);}}
int main(void) {
    int numerator, denominator;
    printf("Enter numerator and denominator: ");
    if (scanf("%d %d", &numerator, &denominator) != 2) return 1;
    simplify_fraction(&numerator, &denominator);
    printf("Simplified Fraction: %d/%d\n", numerator, denominator);
    return 0;
}
~
~
~
-- INSERT --
```

1,15

All

```
vboxuser@Ubunt: ~/IT23613522
vboxuser@Ubunt:~/IT23613522$ gcc s.c -o s
vboxuser@Ubunt:~/IT23613522$ ./s
Enter numerator and denominator: 8
12
Simplified Fraction: 2/3
vboxuser@Ubunt:~/IT23613522$
```