



**MAHIDOL  
UNIVERSITY**

*Wisdom of the Land*

[SCPY204]

# Computer Programing for Physicists

**Class 04:** 30 Jan 2023

*Content: Program's input/output, review and exercises*

**Instructor:** Puwis Amatyakul

# Today's Goals

## **Part I: Review**

**Part II:** Homework Q&A

**Part III:** Exercises

(Part IV: Intro to Python)

# Homework

Question and Answer

# Review

## 1. Steps in Programming

Problem analysis / Planning and design / Coding / Testing & debugging / Documentation

## 2. Programming in C

- ✓ C – Basic Syntax : semicolon, comment, whitespace
- ✓ C – Data Types : char, int, float, double, char, void
- ✓ C – Variable : definition, declaration
- ✓ C – Operators : arithmetic, relational, logical
- ✓ C – Decision Making (if ... else)
- ✓ C – Loop (while, for)

# Review

```
/* * Simple C Program * */

// Preprocessor
#include <stdio.h>
#include <stdlib.h>

// Prototype
int afunction(int n);

// Main
int main ()
{
    int i = 2;
    int j;

    j = afunction(i);
    printf("Hello World %d\n",);
    return 0;
}

// Function body
int afunction(int n) {
    return n;
}
```

# Review

## Decision (if)

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

```
if (a < 10)
    printf("a is less than 10\n");
else if (a == 10)
    printf("a is 10\n");
else
    printf("a is greater than 10\n");
```

- If you have compound statements then use brackets (blocks)

```
• if (a < 4 && b > 10) {
    c = a * b; b = 0;
    printf("a = %d, a\'s address = 0x%08x\n", a, (uint32_t)&a);
} else {
    c = a + b; b = a;
}
```

# Review

## Decision (switch)

```
int c = 10;
switch (c) {
    case 0:
        printf("c is 0\n");
        break;
    ...
    default:
        printf("Unknown value of c\n");
        break;
}
```

# Review

## Loops

```
for (i = 0; i < MAXVALUE; i++) {  
    dowork();  
}  
while (c != 12) {  
    dowork();  
}  
do {  
    dowork();  
} while (c < 12);
```



# Today's Goals

**Part I:** Review

**Part II:** Homework Q&A

**Part III:** Exercises

(Part IV: Intro to Python part 1)

# C Basic: Exercises

**Ex 1:** Let's write a C program to find factors of any integer.

**Ex 2:** Vector product

$$\begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 6 \\ 7 \end{bmatrix} = ?$$

**Ex 3:** Matrices Multiplication

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 4 & 5 \\ 6 & 7 \\ 8 & 9 \end{bmatrix} = ?$$

**Ex 4:** Gaussian elimination of 3 x 3

$$\begin{bmatrix} 9 & 3 & 4 \\ 4 & 3 & 4 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 8 \\ 3 \end{bmatrix}$$

# Today's Goals

**Part IV:** Additional example /  
explanation

# C Basic: array dynamic allocation

```
int **2d_array;  
2d_array = (int **)malloc(sizeof(int *)*N);
```

# Pointers in C (and C++)

Step 1:

```
int main (int argc, argv) {  
    int x = 4;  
    int *y = &x;  
    int *z[4] = {NULL, NULL, NULL, NULL};  
    int a[4] = {1, 2, 3, 4};  
    ...  
}
```

Note: The compiler converts `z[1]` or `*(z+1)` to  
*Value at address (Address of `z` + `sizeof(int)`);*

In C you would write the byte address as:  
`(char *)z + sizeof(int);`

or letting the compiler do the work for you  
`(int *)z + 1;`

Program Memory Address

<i>x</i>	4	0x3dc
<i>y</i>	0x3dc	0x3d8
	NA	0x3d4
	NA	0x3d0
<i>z[3]</i>	0	0x3cc
<i>z[2]</i>	0	0x3c8
<i>z[1]</i>	0	0x3c4
<i>z[0]</i>	0	0x3c0
<i>a[3]</i>	4	0x3bc
<i>a[2]</i>	3	0x3b8
<i>a[1]</i>	2	0x3b4
<i>a[0]</i>	1	0x3b0

# Pointers Continued

Step 1:

```
int main (int argc, argv) {  
    int x = 4;  
    int *y = &x;  
    int *z[4] = {NULL, NULL, NULL, NULL};  
    int a[4] = {1, 2, 3, 4};  
}
```

Step 2: Assign addresses to array z

```
z[0] = a;           // same as &a[0];  
z[1] = a + 1;     // same as &a[1];  
z[2] = a + 2;     // same as &a[2];  
z[3] = a + 3;     // same as &a[3];
```

Program Memory Address

<i>x</i>	4	0x3dc
<i>y</i>	0x3dc	0x3d8
	NA	0x3d4
	NA	0x3d0
<i>z[3]</i>	0x3bc	0x3cc
<i>z[2]</i>	0x3b8	0x3c8
<i>z[1]</i>	0x3b4	0x3c4
<i>z[0]</i>	0x3b0	0x3c0
<i>a[3]</i>	4	0x3bc
<i>a[2]</i>	3	0x3b8
<i>a[1]</i>	2	0x3b4
<i>a[0]</i>	1	0x3b0

# Pointers Continued

Step 1:

```
int main (int argc, argv) {
    int x = 4;
    int *y = &x;
    int *z[4] = {NULL, NULL, NULL, NULL};
    int a[4] = {1, 2, 3, 4};
```

Step 2:

```
z[0] = a;
z[1] = a + 1;
z[2] = a + 2;
z[3] = a + 3;
```

Step 3: No change in z's values

```
z[0] = (int *) ((char *)a);
z[1] = (int *) ((char *)a
                + sizeof(int));
z[2] = (int *) ((char *)a
                + 2 * sizeof(int));
z[3] = (int *) ((char *)a
                + 3 * sizeof(int));
```

Program Memory Address

<i>x</i>	4	0x3dc
<i>y</i>	0x3dc	0x3d8
	NA	0x3d4
	NA	0x3d0
<i>z[3]</i>	0x3bc	0x3cc
<i>z[2]</i>	0x3b8	0x3c8
<i>z[1]</i>	0x3b4	0x3c4
<i>z[0]</i>	0x3b0	0x3c0
<i>a[3]</i>	4	0x3bc
<i>a[2]</i>	3	0x3b8
<i>a[1]</i>	2	0x3b4
<i>a[0]</i>	1	0x3b0