

Computer programming II
Handout 1 (Modularity using Functions)

Inter-Module Communication

The inter module communication in C++ can be done using either of the two communication methods. These are

- communication without data transfer and
- communication with data transfer

Communication without data transfer

In this type of communication

- ✓ No parameters are used in the functions involved
- ✓ No common variables are accessed by any of the functions
- ✓ Every function has a void return type

Usually this kind of communication has a purpose of menu preparation in large programs. Consider the following modular program.

<pre>void menu(); void select(); void main() { menu(); select(); } void select() { cout<<"\n Enter you choice "; cout<<"\n Choice = "; }</pre>	<pre>void menu() { cout<<"\n <u>Menu</u> cout<<"\n 1. Sum"; cout<<"\n 2. Difference "; cout<<"\n 3. Product "; cout<<"\n 4. Quotient "; cout<<"\n 5. Exit "; }</pre>
---	--

Communication with data transfer

In inter modules communication; there are **three** different data transfer **protocols**, which are based on the direction of the flow (transfer) of data. These are

1. Protocol 1 (P1):- It is a protocol from the calling to the called
(Calling $\longrightarrow$ Called)
It is a one-way channel in which the called function does not transfer data to the calling function.
2. Protocol 2 (P2):- It is a protocol from the called to the calling
(Calling $\longleftarrow$ Called)
It is a one-way channel in which the calling function does not transfer data to the called function.
3. Protocol 3 (P3):- It is a protocol both from the calling to the called and from the called to the calling.
(Calling $\longleftrightarrow$ Called)
It is a two way channel in which data security is the least.

Moreover, in inter modules communication; there are **three** data transfer **tools**. These are

1. Data transfer using common variables
2. Data transfer using the return statement (i.e. function identifier)
3. Data transfer through parameter (argument) passing

1. Data transfer using common variables

- Modules can communicate with each other through a **common variable** that is accessible by each one of them.
- Common variables support P3.
- Common variables can be used to transfer data from the calling to the called or from the called to the calling or both.
- What do you think are the advantages and the disadvantages of using common variables?

Example

The following modular program uses common variables in order to compute and display the sum of the list of numbers that has been accepted from the keyboard.

<pre>const int size=100; unsigned n; float s=0; float num[size]; void inpNumber(); void sum(); void display(); void main() { clrscr(); inpNumber(); sum(); display(); getch(); } void inpNumber() { cout<<"\nEnter the size of the numbers"; cout<<"\n\tSize = "; cin>>n;</pre>	<pre>for (int i=0; i<n; i++) { clrscr(); cout<<"\n Number " <<i +1<<" = "; cin>>num[i]; } void sum() { for (int i=0; i<n; i++) s=s + num[i]; } void display () { clrscr(); cout<<" \n The sum of the numbers \n"; for (int i=0; i<n; i++) cout<<num[i]<<" "; cout<<" \n is " << s; }</pre>
---	---

- What do you think is the effect of writing the assignment statement (n=0 ;) as the first statement inside the body of the **sum** function?

2. Data transfer using the return statement

- ✓ The function identifier acts like a variable to transfer data from the called to the calling.
- ✓ It is a means of returning only one value to the calling function.
- ✓ Supports only P2
- ✓ The called function must provide
 - The data type of the returned value
 - The actual value being returned.
- ✓ To return a single value, a function can use a return statement.

Syntax:

return <expression>;

- ✓ When the return statement is encountered inside a called function
 - The expression is evaluated first
 - The evaluated value of the expression is then automatically converted to the <returnType> declared in the function header.
 - Failure to match the data type of the return value exactly with the functions declared data type might not result in an error when your program is compiled but it may lead to undesired results because that return value is always converted to the appropriate type before being sent to the calling function.
 - Then after a single value will be returned to the calling function.
- ✓ The calling function must
 - Be alerted to the type of value to expect.
 - Properly use the return value.

Example:

The above modular program can be written using this method of data transfer as follows.

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

3. Data transfer through parameter passing

Here we have two ways of data transfer

- Parameter passing by value
- Parameter passing by reference

3. 1. Parameter passing by value

- ✓ Does not allow the called function to have direct access to the variable(s) defined in the calling function. This enables the called function to use the copy of the values of the calling function's variable(s).
- ✓ Allows functions to be written as independent entities
- ✓ Alleviates the concern of altering argument variables (calling function variables)
- ✓ It is the default parameter passing
- ✓ Can be disadvantage if a large data items is being passed. This is because coping such data can take a considerable amount of execution time. In such cases a call by reference is good for efficient performance (both space and time)
- ✓ It is not suitable if the values of the calling variables (arguments) must be modified

Example:

What is the output of the following modular program?

```
void swap(int x,int y)
{
    int temp = x;
    x=y;
    y=temp;
}

int main()
{
    int a = 5;
    int b = 7;
    cout<<"Before swap a = "<<a<<" and b = "<<b<<endl;
    swap(a,b);
    cout<<"After swap a = "<<a<<" and b = "<<b<<endl;
    return 0;
}
```

3. 2 Parameter passing by Reference

- ✓ Allows the called function to have direct access to the variable(s) defined in the calling function. This enables the called function to use and change the values of the calling function's variable(s).
- ✓ Can be advantage if a large data items is being passed. Call by reference is good for efficiency (both space and time)
- ✓ When parameter passing by reference is used, the address of the variable has to be passed to the called function.

✓ C++ Provides two types of address parameters. These are

- o Pointer parameter
- o Reference parameter and

Example:**Pointer parameter passing**

What is the output of the following modular program?

```
void swap(int *xp,int *yp)
{
    int temp = *xp;
    *xp=*yp;
    *yp=temp;
}

int main()
{
    int x = 5;
    int y = 7;

    cout<<" Before swap is called x = "<<x<<" and y = "<<y<<endl;
    swap(&x,&y);
    cout<<" After swap is called x = "<<x<<" and y = "<<y<<endl;
    return 0;
}
```

Example:**Reference (Alias or another name or nick name) parameter passing**

What is the output of the following modular program?

```
void swamp(int &xp,int &yp)
{
    int temp = xp;
    xp=yp;
    yp=temp;
}

int main()
{
    int x = 5;
    int y = 7;

    cout<<" Before swamp is called x = "<<x<<" and y = "<<y<<endl;
    swamp(x,y);
    cout<<" After swamp is called x = "<<x<<" and y = "<<y<<endl;
    return 0;
}
```

Array and Function

1. Write a program that uses the following functions:

Fill_array() takes as arguments the name of an array of double values and an array size. It prompts the user to enter double values to be entered in the array. It ceases taking input when the array is full or when the user enters non-numeric input, and it returns the actual number of entries.

Show_array() takes as arguments the name of an array of double values and an array size and displays the contents of the array.

Reverse_array() takes as arguments the name of an array of double values and an array size and reverses the order of the values stored in the array. The program should use these functions to fill an array, show the array, reverse the array, show the array, reverse all but the first and last elements of the array, and then show the array

2. Sometimes, programs use counter variables to summarize data, such as the results of a survey. We used separate counters in our die-rolling program to track the number of occurrences of each side of a die as the program rolled the die 6000 times.
3. **Using Arrays to Analyze Survey Results**
Forty students were asked to rate the quality of the food in the student cafeteria on a Scale of 1 to 10 (where 1 means awful and 10 means excellent). Place the 40 responses in an integer array, and summarize the results of the poll.
4. A parking system charges customers a minimum fee of 2.00 Birr to park their cars for up to three hours. If a car parks for more than three hours, the customer will be charged extra 0.50 cents for each additional hour. The system assumes that no car parks longer than 24 hours at a time