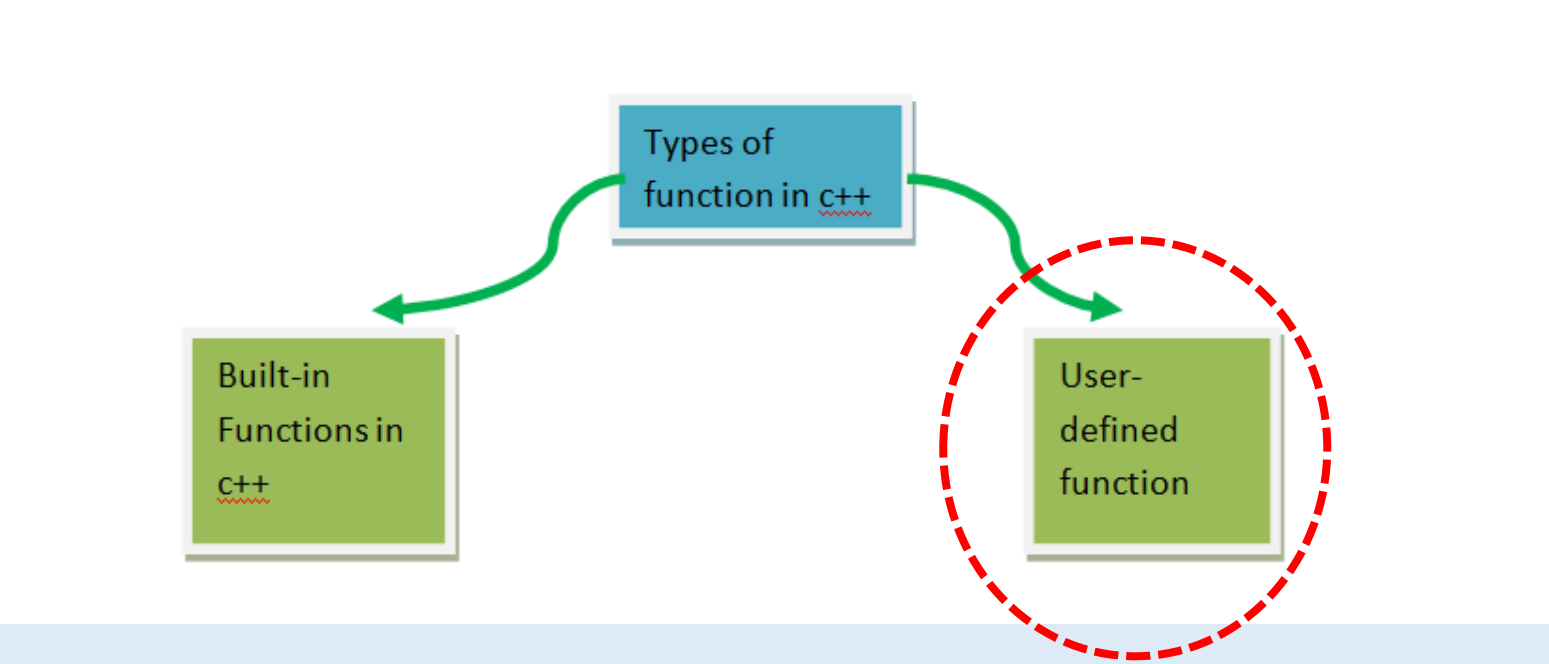
Lab #7

Functions (2)

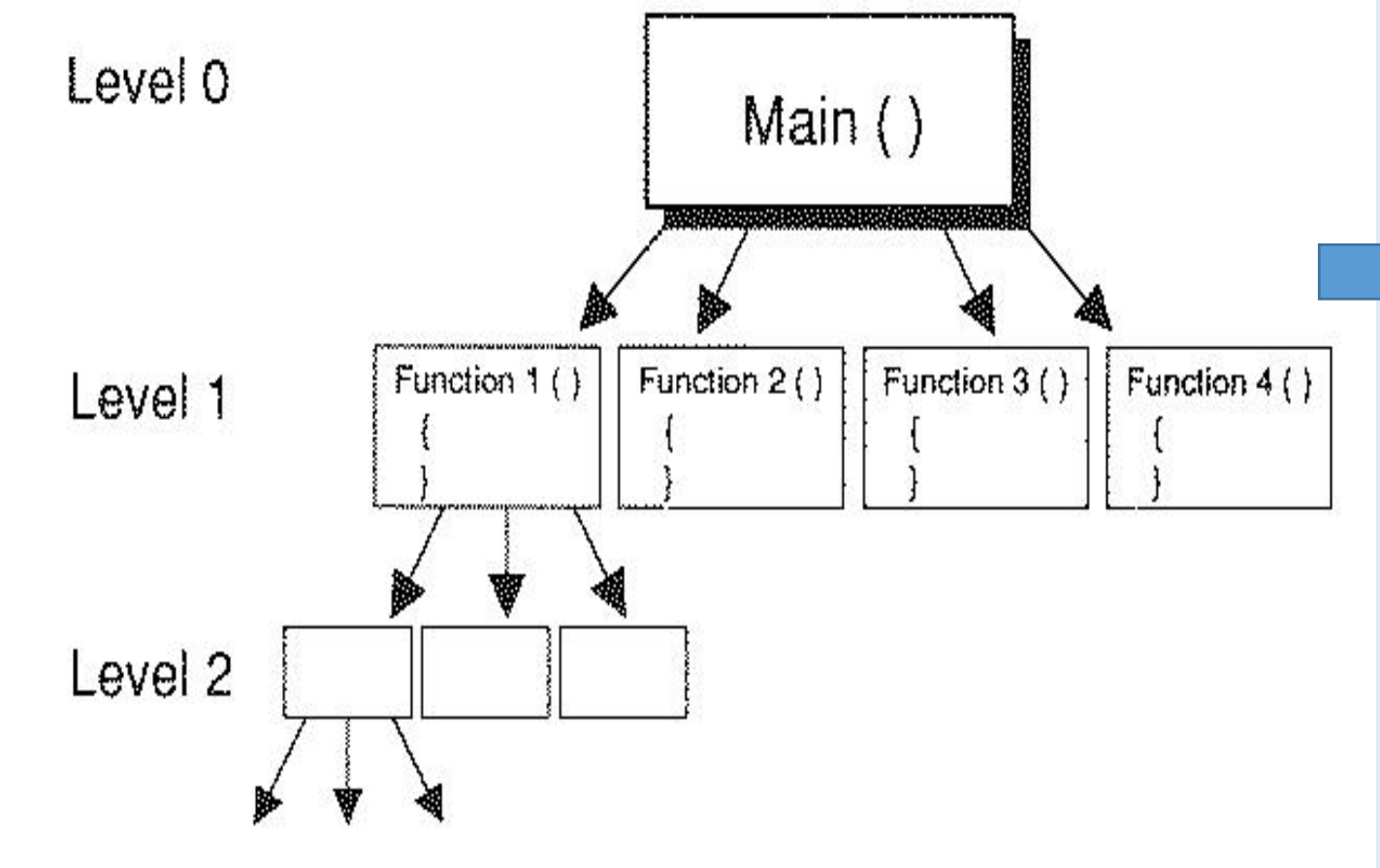
Structured Programming 2017/2018

Today's Lab

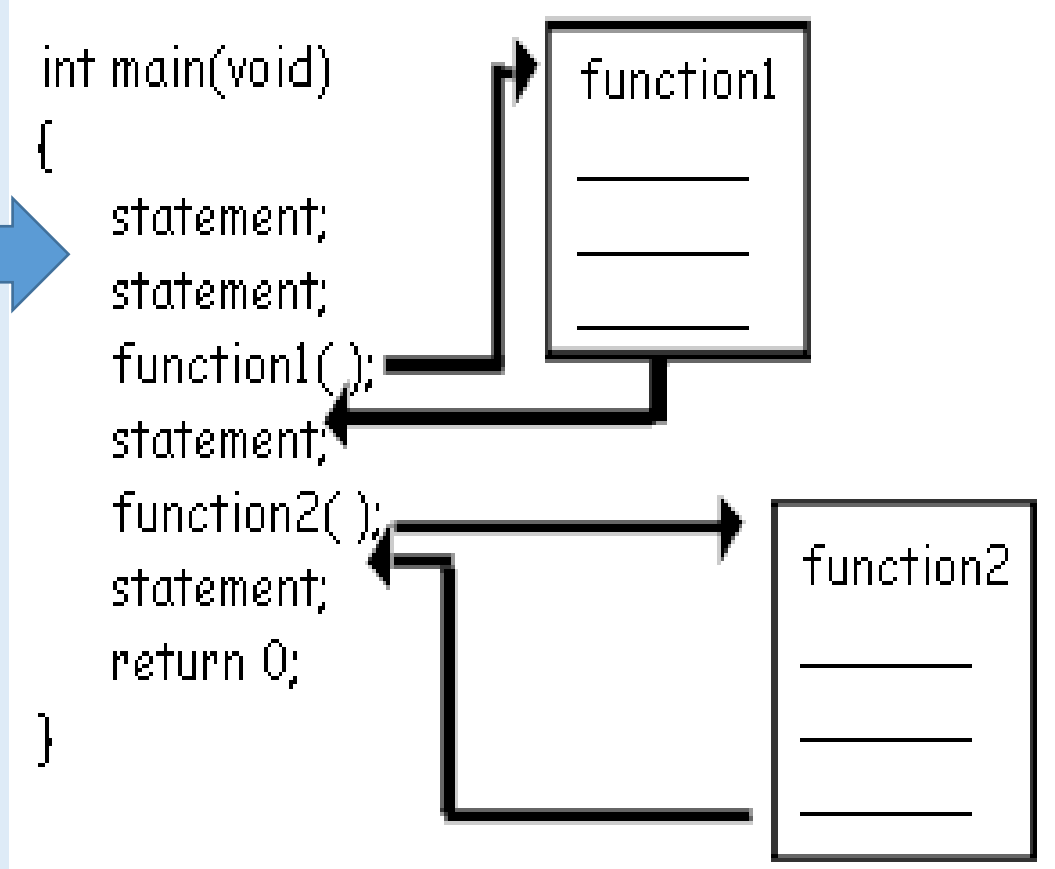

Types of function in c++
Built-in Functions in c++
User-defined function

What is a Function? Why?

Design



Code



User Defined Functions

In order to create user defined functions in C++ you need to provide the following:

- Function Declaration.
- Function Definition.
- Function Calling.

Function Declaration

```
return_type Fn_name(parameter_list type);
```

```
6  double sum(int par1, double par2, double par3);
7  int sub(double par1, int par2);
8  bool check(char);
9  char myFn(int, bool);
10 void yourFn(void);
```

Where?

Function Definition

```
return_type Fn_name(parameter_list)
{
    // your code goes here
}
```

```
26 double sum(int A, double B, double C)
27 {
28     double sum = A+B+C;
29     return sum;
30 } // end sum
```

Where?

Function Call

```
Fn_name(arguments_list);

result = Fn_name(arguments_list);
```

```
sum(1, 2, 3);
```

Where?

1. Factorial

- Write a function to take a number from the user, compute the factorial and display the result.

- Modify the code so the main() takes the input from the user.

- Modify the code so the main() takes the input and displays the result.

Function has no input
Function has no return

```
#include<iostream>
using namespace std;
void factorial(); /*prototype*/
void main()
{
    cout<<"Enter a number : ";
    factorial();  /*calling*/
}
void factorial( ) /*definition*/
{
    int number;
    cin>>number;
    int f=1;
    for(int i=1;i<=number;i++)
        f*=i;
    cout<<"Factorial of "<<number<<" = "<<f<<endl;
}
```

Function has no input
Function has no return

1. Factorial – (cont.)

- Write a function to take a number from the user, compute the factorial and display the result.

- Modify the code so the main() takes the input from the user.

- Modify the code so the main() takes the input and displays the result.

Function has one input (parameter)
Function has no return

```
#include<iostream>
using namespace std;
void factorial(int); /*prototype*/
void main()
{
    int number;
    cout<<"Enter number : ";
    cin>>number;
    factorial(number);  /*calling BY VALUE*/
}
void factorial(int number) /*definition*/
{
    int f=1;
    for(int i=1;i<=number;i++)
        f*=i;
    cout<<"Factorial of "<<number<<" = "<<f<<endl;
}
```

A copy of the argument fills the function's parameter, even if different names

Arguments and parameters are different variables in memory

Function has one input (parameter)
Function has no return

1. Factorial – (cont.)

- Write a function to take a number from the user, compute the factorial and display the result.

- Modify the code so the main() takes the input from the user.

- Modify the code so the main() takes the input and displays the result.

Function has one input (parameter)
Function has a return value

```
#include<iostream>
using namespace std;
int factorial(int); /*prototype*/
void main()
{
    int number,fact;
    cout<<"Enter number : ";
    cin>>number;
    fact=factorial(number);  /*calling BY VALUE*/
    cout<<"Factorial of "<<number<<" = "<<fact<<endl;
}

int factorial(int number) /*definition*/
    {
    int f=1;
    for(int i=1;i<=number;i++)
    f*=i;
    return f;
}
```

A copy of the argument fills the function's parameter, even if different names

Arguments and parameters are different variables in memory

Function has one input (parameter)
Function has a return value

Ready… Steady… Code!

1.Power Function

Write a C++ function that calculate the power for a given number

Sample run:

```
Please Enter Number: 3
Please Enter Power: 4
Result: 81
```

```
#include <iostream>
using namespace std;
int power(int b,int e) /*definition*/
{
    int p=1;
    for(int i=1;i<=e;i++)
        p*=b;
    return p;
}
```

2. Functions calling other functions

Remember this equation: $e^x = \sum_{i=0}^{n} \frac{x^i}{i!}$

We want to re-code the program in a functional structure, using:

1) ***factorial*** function ***(that we just did)***
2) ***power*** function ***(that you just did)***
3) ***exponential*** function *(you do this!!)*

Sample run:

```
Please enter the power of e and the limit of the series: 3 2

e to the power of 3 to limit 2 = 8.5
```

```
float expo(int x, int n)
{
    float result=0;
    for(int i=0;i<=n;i++)
        result+= power(x,i) / factorial(i);
    return result;
}
void main()
{
    int pow,limit;
    cout<<"Please enter the power of e and the limit of the series:\n";
    cin>>pow>>limit;
    float e=expo(pow,limit); /*calling BY VALUE*/
    cout<<"e to the power "<<pow<<" to the "<<limit<<" limit="<<e<<endl;

}
```

Arguments and parameters are different variables in memory

3. Distance Summation (Structures & Functions)

Write a program that represents the **distance** in the form of a **structure** containing **meters**, and **centimeters**.

Program should read from user 2 distances, then calculate the sum of the given distances.

Hint: Write function to calculate sum

Sample of Execution:

```
Enter First Distance: 1 40

Enter Second Distance: 2 70

Result: 4 10
```

```
struct Distance
{
	int meters;
	int centimeters;
};
Distance Sum_Distance (Distance D1 , Distance D2)
{
	Distance Result;
	Result.meters = D1.meters + D2.meters;
	Result.centimeters = D1.centimeters + D2.centimeters;
	while (Result.centimeters >= 100)
	{
		Result.meters++;
		Result.centimeters-=100;
	}
	return Result;
}
```

```
int main()
{
      Distance FirstDistance, SecondDistance , ResultDistance;
      cout << "Enter First Distance: ";
      cin>> FirstDistance.meters>>FirstDistance.centimeters;
      cout << "Enter Second Distance: ";
      cin>> SecondDistance.meters>>SecondDistance.centimeters;
      ResultDistance = Sum_Distance(FirstDistance,SecondDistance);
      cout << "Result: " << ResultDistance.meters << " " <<
      ResultDistance.centimeters << endl;
      system("pause");
      return 0;
}
```

4. Emirp

An Emirp (Prime spelt backwards) is a Prime that gives you a different Prime when its digits are reversed. For example, 17 is Emirp because 17 as well as 71 are Prime. In this problem, you have to decide whether a number N is Non-prime or Prime or Emirp.

Sample of Execution:

```
Enter Number: 17
17 is Emirp
Enter Number: 19
17 is Prime
Enter Number: 18
17 is not Prime
```

```
#include <iostream>
using namespace std;

bool prime(int);
int reverse(int);
bool emirp(int);
void main()
{
    int number;
    cout<<"Enter a number:";
    cin>>number;
    if(prime(number))
    {
        if(emirp(number))
           cout<<number<<" is emirp \n";
        else
           cout<<number <<" is prime \n";
    }
    else
        cout<<"number is NOT Prime"<<endl;
}
```

```
bool prime(int x)
{
   for(int i=2;i<=x/2;i++)
   {
      if(x%i==0)
       return false;
   }
   return true;
}
int reverse (int n)
{
   int rev=0;
   while(n!=0)
   {
     int digit=n%10;
     rev*=10;
     rev+=digit;
     n=n/10;
   }
   return rev;
}
```

```
bool emirp(int x)
{
   int y=reverse(x);
   if(prime(y))
     return true;

   return false;

}
```

Password Validation Example (Built-in Functions)

- Write a program that validates a password entered by the user without displaying it. The password should contain at least one of these special characters ($, % , _ , #, @) and at least one number.

- Hints: - library: conio.h, built-in function _getch() → returns entered char
  - use two boolean variables isNum, isSpecial
  - use The ASCII code of numbers 0-9 is 48-57

Sample of Execution:

```
Enter Password: abc
Output: Not valid

Enter Password: abc_1
Output: Valid
```

```
#include<iostream>
#include<conio.h>
using namespace std;

int main()
{
char ch;
bool isSpecial = false, isNum = false;

cout << "Enter Password: ";
```

```
while (true)
{
   ch = _getch();
   If (ch == '\r')
   {
      Break;
   }
   if (ch == '$' || ch == '%' || ch == '_' || ch == '#' || ch == '@')
   {
      isSpecial = true;
   }
   if (ch >= 48 && ch <= 57)
   {
      isNum = true;
   }
}
if (isSpecial && isNum)
cout << endl << "Password is valid" << endl;
else
cout << endl << "Password is not valid" << endl;
return 0;
}
```

Thank you!