First Year First Semester Course
M.Tech. (CS) [Batch 2021-23]

Lecture #12

Introduction to Programming

Introduction to C++: Data Types, Operators, Expressions, Control Structures, Functions

Course Instructor:

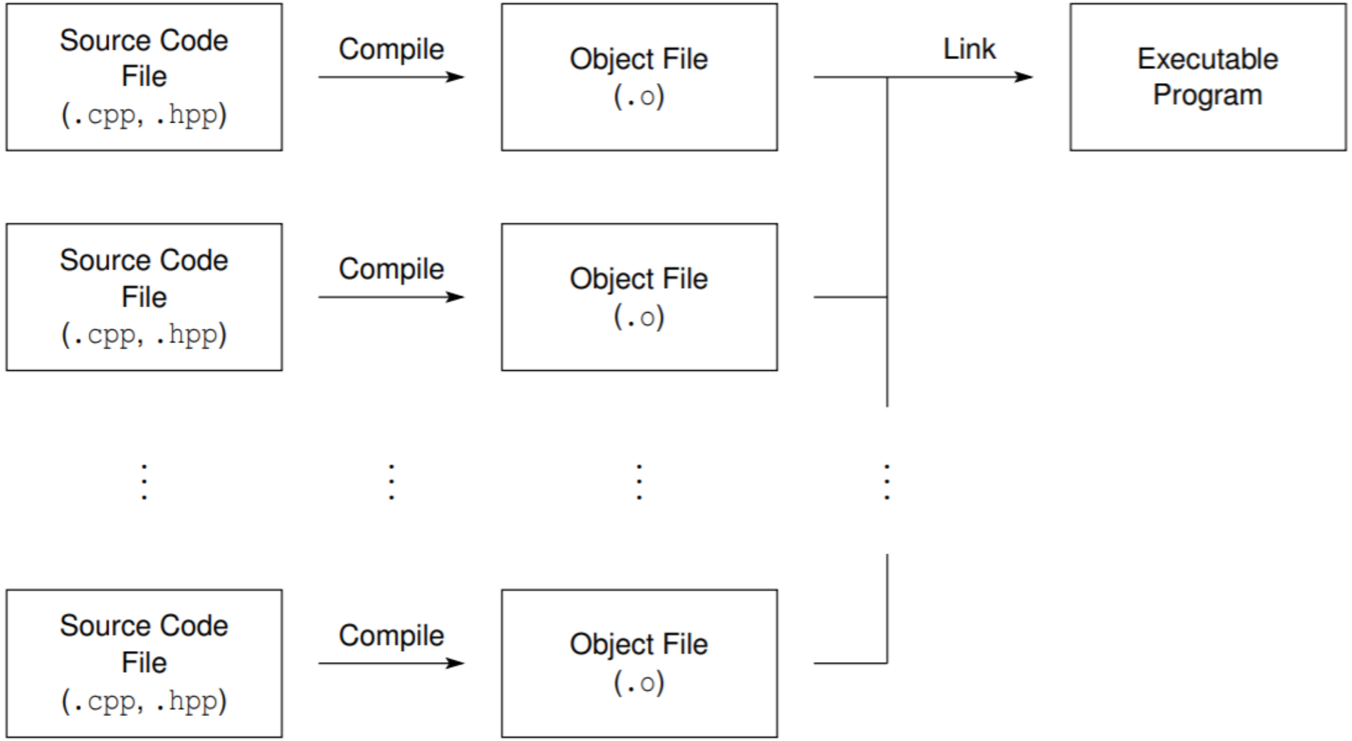
Dr. Monidipa Das

DST-INSPIRE Faculty

Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)

Indian Statistical Institute (ISI) Kolkata, India

Software Build Process



GNU Compiler Collection (GCC) C++ Compiler

- g++ command provides both compiling and linking functionality
- command-line usage:

```
g++ [options] input file . . .
```

- many command-line options are supported
- compile C++ source file file.cpp to produce object code file file.o:

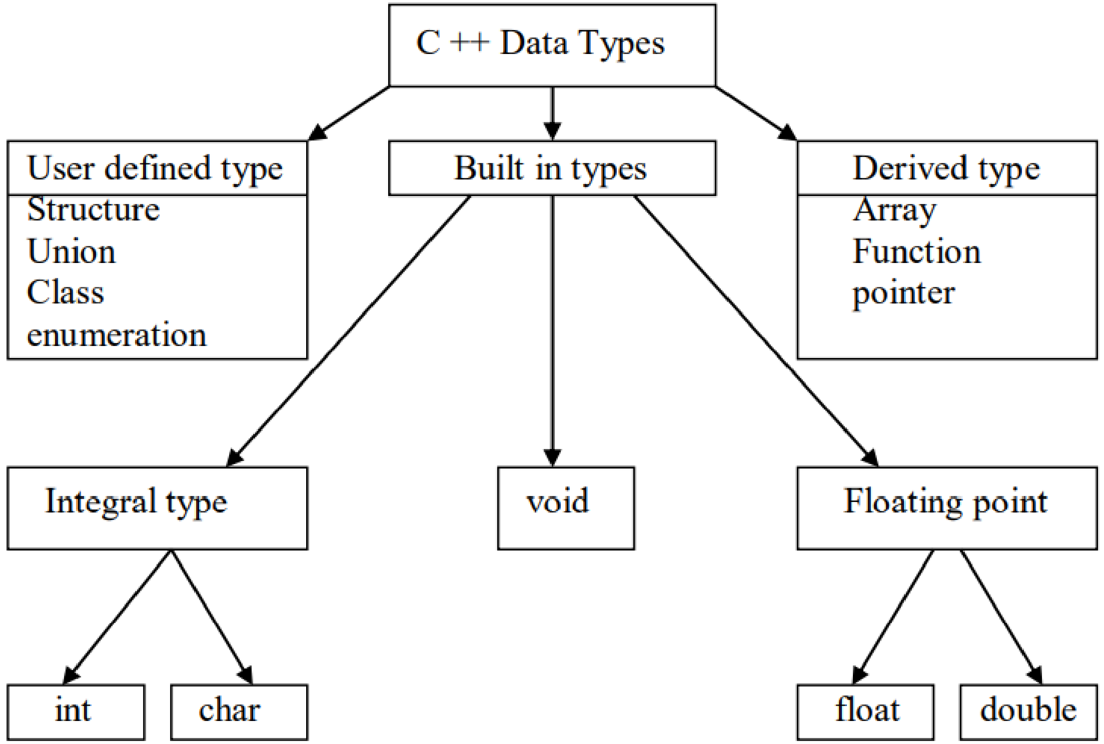
```
g++ -c file.cpp
```

- link object files file 1.o, file 2.o, ... to produce executable file executable:

```
g++ -o executable file 1.o file 2.o ...
```

Data Types, Operators, Expressions, Control Structures in C++

Basic Data Types In C++



Declaration of Variables

- C++ allows the declaration of variable anywhere in the scope

```
int main()
{
   float x;                      // declaration
   float sum=0;
   for(int i=0; i<5; i++)        // declaration
   {
     cin>>x;
     sum=sum+x;
   }
   float average;                // declaration
   average=sum/(i-1);
   cout<<average;

   return 0;
}
```

Reference Variables

- Provides an alias (nick name) for a previously defined variable

```
data-type & reference-name = variable-name
```

- Example:

```
float total = 100;
float & sum = total;
cout<<total;
cout<<sum;
total=total+10;
sum=0;
```

```
int x;
int *ptr = &x;
int & m =*ptr;

int & n=25;
```

Reference Variables [contd.]

```
void f(int & x)  // uses reference
{
   x=x+10;
}
int main()
{
   int v=10;
   f(v);         // function call…
                 // call by reference
}
```

Operators in C++

- All operators in C are also valid in C++
- C++ introduces some new operators

::	Scope resolution operator
::*	Pointer-to-member declarator
->*	Pointer-to-member operator
.*	Pointer-to-member operator
delete	Memory release operator
new	Memory allocation operator

Scope resolution operator

```
#include<iostream>

using namespace std;
int m=10;    //global m

int main()
{
    int m=20; //local m

    {
            int t=m;
            int m=30;    //m local to this block

            cout<<"we are in inner block \n";
            cout<<“t="<<t<<"\n";
            cout<<"m="<<m<<"\n";
            cout<<"::m = "<< ::m <<"\n";

    }
    cout<<"\n We are in outer block \n";
    cout<<"m="<<m<<"\n";
    cout<<"::m="<<::m<<"\n";

    return 0;
}
```

Member Dereferencing Operator

::*	To declare a pointer to a member of a class
.*	To access a member using object name and a pointer to that member
->*	To access a member using a pointer to the object and a pointer to that member

Member Dereferencing Operator [contd.]

```
#include<iostream>
using namespace std;
class M
{
       int x;
       int y;
   public:
       void set_xy(int a, int b)
       {
              x=a;
              y=b;
       }
       friend int sum(M m);
}
```

```
int sum(M m)
{
       int M ::* px= &M :: x;
       int M ::* py= &M :: y;
       M *pm = &m;
       int S= m.*px + pm->*py;
       return S;
}
```

Memory Management Operators

- C++ supports calloc(), malloc(), free() etc.
- Further defines two unary operators
  - new
  - delete

pointer-variable = ***new*** data-type;

pointer-variable = ***new*** data-type(value);

pointer-variable = ***new*** data-type[size];

delete pointer-variable;

delete [size] pointer-variable;

Type Cast Operator

```
(type-name) expression    // C notation
type-name (expression)    // C++ notation

average = sum/(float)num;   // C notation
average = sum/float (num);   // C++ notation

ptr= int * (q);     //illegal
typedef int * int_pt;
p= int_pt(q);
```

Expressions in C++

• Constant expressions	17 + 8 / 3.0
• Integral expressions	6 + int(9.0)
• Float expressions	6 + float(9)
• Pointer expressions	&m ptr+1
• Relational expressions	a <= b+c
• Logical expressions	a >b && b>0
• Bitwise expressions	y << 2

Operator Overloading and Operator Precedence

- Assigning different meaning to an operator depending on the context
- Operator overloading in C: *, &
- Example in C++

```
cout << 75.86;
cout << “well done”;
```

- The ***scope resolution operator*** **::** has the highest priority

Control Structures

- Sequence structure
- Selection (branching) structure
  - if-else
  - switch case
- Loop (iteration or repetition) structure
  - do-while
  - while, for

```
if(expr is true)
{
        action1;
}
action2;
action3;
```

```
if(expr is true)
{
        action1;
}
else
{
        action2;
}
action3;
```

Control Structures [contd.]

```
switch(expression)
{
         case1:
         {
                  action1;
         }
         case2:
         {
                  action2;
         }
         case3:
         {
                  action3;
         }
         default:{
                  action4;}
}
action5;
```

```
do
{
         action1;
}
while(condition is true);
action2;
```

```
while(condition is true)
{
         action1;
}
action2;
```

```
for(initialize; test; increment)
{
         action1;
}
action2;
```

Functions in C++

The Main Function

- In C++, the main function returns a value of type integer

- Main function prototypes in C++

```
int main();
int main(int argc, char* argv[]);
```

- Better to always include the return statement

Function Prototyping and Declaration

- C++ makes the prototyping essential

```
type function-name (argument-list);
```

- Example

```
float volume(float a, float b, float c);
float volume(float x, float y, float z)
{
        float v= x*y*z;
        - - - -
         - - - -
}
```

Call by Reference

- Passing argument by reference
- Formal arguments in the called function becomes the aliases to the actual arguments in the calling function

```
void swap(int &a, int &b) // a and b are reference variables
{
    int t=a;
    a=b;
    b=t;
}
```

Return by Reference

- In C++ a function can also return a reference

```
int & max(int &x, int &y)
{
    if(x>y)
        return x;
    else
        return y;
}
```

Inline Function

- A function that is expanded in line when it is invoked
- Eliminates the cost of calls to small function

```
inline function-header
{
        function body
}
```

- Example

```
inline double cube(double a)
{
        return(a*a*a);
}
```

Friend and Virtual Function

- C++ introduces two new types of functions
  - Friend function
    - Common function; friendly with multiple classes
    - Not in the scope of the class to which it has been declared as friend
    - It cannot be called using the object of that class
    - It cannot access the member names directly
  - Virtual Function
    - Used in the context of inheritance
    - Helps achieving runtime polymorphism when accessed through a pointer to the base class

Default and Const Arguments

- Default Arguments

```
float amount(float principal, int period, float rate=0.15);
value= amount(5000,7);

int mul(int i=2, int j);   //illegal
int mul(int i=0, int j, int k=10);   //illegal
```

- const Arguments

int strlen(**const** char *p);

Function Overloading

- Function polymorphism
- The same function name can be used to create functions that perform a variety of different tasks
- Argument list is different

- Example

```
int add(int a, int b, int c);
int add(int a, int b);
double add(int a, double b);
double add(double a, int b);
```

```
cout<< add(5,10,25);
cout<< add(15,25);
cout<< add(15,12.5);
cout<< add(1.5,8);
```

Questions?