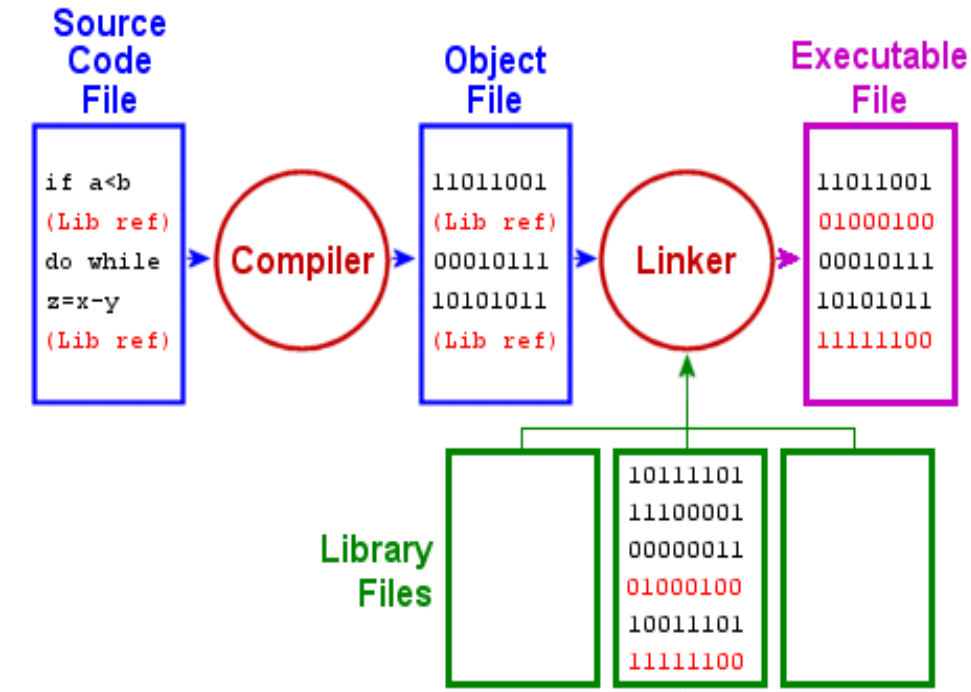
PROGRAMMING IN C++

KAUSIK DATTA
18-Oct-2017



Objectives

- Recap C
- Differences between C and C++
  - IO
  - Variable Declaration
  - Standard Library
- Introduction of C++ Feature : Class

Recap C

- Built in Types
- Enumerated Type
- Derived data type
  - struct
  - union
  - Array
  - pointer
- Type Modifiers
  - short
  - long
  - signed
  - unsigned

Recap C

- Variable
  - Is a name given to a storage area
- Operator
- Literal
- Expression
  - Must have a value
- Statement
  - Command for a specific action
    - Expression statement
    - Control statement

Quiz #1

```
struct st1
{
    int a;
    char c;
    int b[10];
};
```

- What is the size of struct st1?
  - Without using sizeof operator

DIFFERENCES BETWEEN C AND C++

Hello World Program

```
#include <stdio.h>
int main()
{
    printf("Hello World");
    printf("\n");
   return 0;
}
```

```
#include <iostream>
int main()
{
    std::cout << "Hello World";
    std::cout << std::endl;
    return 0;
}
```

Hello World Program

- IO header is stdio.h
- printf is used to print to stdout
- stdout prints to console
- printf is declared in global scope
- printf is a variadic function
- \n is the escaped new line character

- IO header is iostream
- Operator << is used stream to std::cout
- Std::cout is ostream which prints to console
- cout declared in std namespace
- << is a binary operator
- Std::endl is newline functor

Source: Programming in C++ by Prof. Partha Pratim Das

Program #2

```
#include <stdio.h>

int main()

{

  int a, b;

  int sum;

  printf(“Input two numbers:\n”);

  scanf(“%d%d”, &a, &b);

  sum = a + b;

  printf(“sum of %d and %d is: %d\n”,
  a, b, sum);

  return 0;

}
```

```
#include <iostream>

Using namespase std;

int main()

{

  int a, b;

  cout << “Input two numbers:\n”;

  cin >> a >> b;

  int sum = a + b;

  cout << “sum of “ << a << “ and “ <<
  b << “ is: ” << sum << endl;

  return 0;

}
```

Source: Programming in C++ by Prof. Partha Pratim Das

Program #3

```
#include <stdio.h>

#include <math.h>

int main() {

  double a, b;

  printf(“Input number:\n”);

  scanf(“%lf”, &a);

  b = sqrt(a);

  printf(“sq root of %lf is: %lf\n”, a, b);

  return 0;

}
```

```
#include <iostream>

#include <cmath>

Using namespase std;

int main()

{

  double a;

  cout << “Input number:\n”;

  cin >> a;

  double b = sqrt(a);

  cout << “sq root of “ << a << “ is: ” << b
  << endl;

  return 0;

}
```

Source: Programming in C++ by Prof. Partha Pratim Das

Standard Library Header Conventions

	C Heaeder	C++ Header
C program	Use .h #include <stdio.h> Names in Global namespace	Not applicable
C++ program	Prefix c no .h #include <cstdio> Names in std namespace	No .h #include <iostream>

```
#include <cmath>
…
std::sqrt(9.0);

#include <math.h>
…
sqrt(9.0);
```

Source: Programming in C++ by Prof. Partha Pratim Das

Namespace std : C++ standard library

C Standard library

- All names are global
- stdout stdin printf scanf sort etc.

C++ Standard library

- All names are within std namespace
- std::cout std::cin std::sort etc

Source: Programming in C++ by Prof. Partha Pratim Das

Namespaces

- Name Clash
  - Occurs when building large systems from pieces
  - Same globally-visible names
  - Very difficult to fix
- Defines scope for enclosing global declarations

```
namespace Mine {
  void print(int);
  float pi = 3.1415925635;
}

void bar(float y) {
  float x = y + Mine::pi;
  Mine::print(5);
}
```

Namespaces

- `using` directive brings namespaces or objects into scope

```
namespace Mine {
  float pi = 3.1415926535;
   void print(int);
}

using Mine::print;
void foo() { print(5); } // invoke Mine::print

using namespace Mine;
float twopi = 2*pi;       // Mine::pi
```

Namespaces

- Namespaces are open: declarations can be added

```
namespace Mine {
  void f(int);
}

namespace Mine {
  void g(int);              // Add Mine::g() to Mine
}
```

Namespaces

- Declarations and definitions can be separated

```
namespace Mine {
  void f(int);
}

void Mine::f(int a) {
 /* … */
}
```

bool data type

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
int main() {
  int a = TRUE;
  printf(“bool is %d\n”, a);
  return 0;
}
```

```
#include <iostream>
Using namespase std;
int main()
{
  bool a = true;
  cout << “bool is “ << a << endl;
  return 0;
}
```

- bool is a built-in data type
- true is a literal
- false is a literal
- Prints 1 for true and 0 for false

new/delete operators

- In C++, the new and delete operators provide built-in language support for dynamic memory allocation and de-allocation.

```
int *pI = new int;
int *pI = new int(102); //new initializes!!
int *pArr = new int[4*num];
```

- Arrays generally cannot be initialized.

```
delete pI;
delete [] pArr;
```

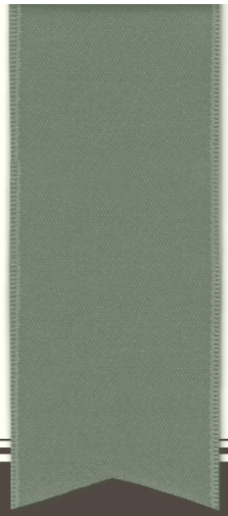
- new does more than malloc!

new/delete & malloc/free

- All C++ implementations also permit use of malloc and free routines.
- Do not free the space created by new.
- Do not delete the space created by malloc
  - Results of the above two operations is memory corruption.
- Matching operators
  - malloc-free
  - new-delete
  - new[] – delete []
- It is a good idea to use only new and delete in a C++ program.

Summary

- C++ gives more flexibility in terms of variable declaration and IO
- C functions are simplified in C++
  - Ease of programming
- Defining own namespace
- Built-in type bool



C++ FEATURE : CLASS

Example: A stack in C

```
typedef struct {
  char s[SIZE];
  int  sp;
} Stack;


stack *create() {
  Stack *s;
  s = (Stack *)malloc(sizeof(Stack));
  s->sp = 0;
  return s;
}
```

Creator function ensures stack is created properly.

Does not help for stack that is automatic variable.

Programmer could inadvertently create uninitialized stack.

Example: A stack in C

```
char pop(Stack *s) {
  if (sp = 0) error("Underflow");
  return s->s[--sp];
}

void push(Stack *s, char v) {
  if (sp == SIZE) error("Overflow");
  s->s[sp++] = v;
}
```

Not clear these are the only stack-related functions.

Another part of program can modify any stack any way it wants to, destroying invariants.

Temptation to inline these computations, not use functions.

C++ Solution: Class

```
class Stack {
  char s[SIZE];
  int sp;

public:

  Stack() { sp = 0; }
  void push(char v) {
    if (sp == SIZE) error("overflow");
    s[sp++] = v;
  }
  char pop() {
    if (sp == 0) error("underflow");
    return s[--sp];
  }
};
```

Definition of both representation and operations

Public: visible outside the class

Constructor: initializes

Member functions see object fields like local variables

C++ Stack Class

- Natural to use

```
Stack st;
st.push('a'); st.push('b');
char d = st.pop();

Stack *stk = new Stack;
stk->push('a'); stk->push('b');
char d = stk->pop();
```

C++ Stack Class

- Members (functions, data) can be public, protected, or private

```
class Stack {
  char s[SIZE];
public:
  char pop();
};

Stack st;
st.s[0] = ‘a’;     // Error: sp is private
st.pop();          // OK
```