



Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

Module 01: Programming in C++

Recap of C

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Tanwi Mallick
Srijoni Majumdar
Himadri B G S Bhuyan



Module Objectives

Module 01

Partha Pratim
Das

Objectives & Outline

Recap of C

Data Types
Variables
Literals
Operators
Expressions
Statements
Control Flow
Arrays
Structures
Unions
Pointers
Functions
Input / Output

Std Library

Organization

Build Process

References

Summary

- Revisit the concepts of C language
- Revisit C Standard Library components
- Revisit the Organization and Build Process for C programs
- Create the foundation for the concepts of C++ with backward compatibility to C



Module Outline

Module 01

Partha Pratim
Das

Objectives & Outline

Recap of C

Data Types
Variables
Literals
Operators
Expressions
Statements
Control Flow
Arrays
Structures
Unions
Pointers
Functions
Input / Output

Std Library

Organization

Build Process

References

Summary

- Recap of C features
 - Data types
 - Variables
 - Literals
 - Operators
 - Expressions
 - Statements
 - Control Constructs – Conditional Flow & Loops
 - Arrays
 - Structures & Unions
 - Pointers
 - Functions
 - Input / Output
- C Standard Library
- Source Organization for a C program
- Build Process



Module 01: Lecture 01

Module 01

Partha Pratim
Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- Recap of C features

- Data types
- Variables
- Literals
- Operators
- Expressions
- Statements
- Control Constructs – Conditional Flow & Loops



First C program

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types
Variables
Literals
Operators
Expressions
Statements
Control Flow
Arrays
Structures
Unions
Pointers
Functions
Input / Output

Std Library

Organization

Build Process

References

Summary

● Print "Hello World"

Source Program

```
#include <stdio.h>

int main() {

    printf("Hello World");
    printf("\n");

    return 0;
}
```

- `stdio.h` header included for input / output
- `main` function is used to start execution
- `printf` function is used to print the string "Hello World"



Data Types

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

Data types in C are used for declaring variables and deciding on storage and computations:

- **Built-in / Basic** data types are used to define raw data

- char
- int
- float
- double

Additionally, C99 defines:

- bool

All data items of a given type has the same size (in bytes). The size is implementation-defined.

- **Enumerated Type** data are internally of int type and operates on a select subset.



Data Types

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

Data types in C further include:

- **void**: The type specifier void indicates no type.
- **Derived** data types include:
 - Array
 - Structure – struct & union
 - Pointer
 - Function
 - String – C-Strings are really not a type; but can be made to behave as such using functions from `<string.h>` in standard library
- **Type modifiers** include:
 - short
 - long
 - signed
 - unsigned



Variables

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- A variable is a name given to a storage area
- Declaration of Variables:
 - Each variable in C has a specific type, which determines the size and layout of the storage (memory) for the variable
 - The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore

```
int    i, j, noOfData;  
char   c, endOfSession;  
float  f, velocity;  
double d, dist_in_light_years;
```




Variables

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- Initialization of Variables:

- Initialization is setting an initial value to a variable at its definition

```
int    i = 10, j = 20, numberOfWorkDays = 22;  
char   c = 'x';  
float  weight = 4.5;  
double density = 0.0;
```



Literals

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- Literals refer to fixed values of a built-in type
- Literals can be of any of the basic data types

```
212    // (int) Decimal literal
0173   // (int) Octal literal
0b1010 // (int) Binary literal
0xF2   // (int) Hexadecimal literal
3.14   // (double) Floating-point literal
'x'    // (char) Character literal
"Hello" // (char *) String literal
```

- In C99, literals are constant values having const types as:

```
212    // (const int) Decimal literal
0173   // (const int) Octal literal
0b1010 // (const int) Binary literal
0xF2   // (const int) Hexadecimal literal
3.14   // (const double) Floating-point literal
'x'    // (const char) Character literal
"Hello" // (const char *) String literal
```



Operators

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- An operator denotes a specific operation. C has the following types of operators:
 - Arithmetic Operators: `+` `-` `*` `/` `%` `++` `--`
 - Relational Operators: `==` `!=` `>` `<` `>=` `<=`
 - Logical Operators: `&&` `||` `!`
 - Bit-wise Operators: `&` `|` `~` `<<` `>>`
 - Assignment Operators: `=` `+=` `-=` `*=` `/=` `...`
 - Miscellaneous Operators: `.` `,` `sizeof` `&` `*` `?:`
- **Arity of Operators:** Number of operand(s) for an operator
 - `+`, `-`, `*`, `&` operators can be *unary* (1 operand) or *binary* (2 operands)
 - `==`, `!=`, `>`, `<`, `>=`, `<=`, `&&`, `||`, `+=`, `-=`, `*=`, `=`, `/=`, `&`, `|`, `<<`, `>>` can work only as *binary* (2 operands) operators
 - `sizeof` `!` `~` `++` `--` can work only as *unary* (1 operand) operators
 - `?:` works as *ternary* (3 operands) operator. The condition is the first operand and the if true logic and if false logic corresponds to the other two operands.



Operators

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- **Operator Precedence:** Determines which operator will be performed first in a chain of different operators

The precedence of all the operators mentioned above is in the following order: (left to right – Highest to lowest precedence)

(), [], ++, --, + (unary), -(unary), !~, *, &, sizeof, *, /, %, +, -, <, <, >, >, ==, !=, *=, =, /=, &, |, &&, | |, ?:, =, +=, -=, *=, =, /=, < <=, > >=

- **Operator Associativity:** Indicates in what order operators of equal precedence in an expression are applied
- Consider the expression $a \sim b \sim c$. If the operator \sim has left associativity, this expression would be interpreted as $(a \sim b) \sim c$. If the operator has right associativity, the expression would be interpreted as $a \sim (b \sim c)$.
 - Right-to-Left: $?:, =, +=, -=, *=, =, /=, <<=, >>=, -, +, !~, *, \&, \text{sizeof}$
 - Left-to-Right: $*, /, \%, +, -, <<, >>, ==, !=, *=, =, /=, \&, |, \&\&, | |$



Expressions

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- Every expression has a value
 - A literal is an expression
 - A variable is an expression
 - One, two or three expression/s connected by an operator (of appropriate arity) is an expression
 - A function call is an expression
- Examples:
 - For

```
int i = 10, j = 20, k;  
int f(int x, int y) { return x + y; }
```
 - Expression are:

```
2.5           // Value = 2.5  
i             // Value 10  
-i           // Value -10  
i - j        // Value -10  
k = 5        // Value 5  
f(i, j)      // Value 30  
i + j == i * 3 // Value true  
(i == j)? 1: 2 // Value 2
```



Statement

Module 01

Partha Pratim
Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- A statement is a command for a specific action. It has no value
 - A ; (semicolon) is a (null) statement
 - An expression terminated by a ; (semicolon) is a statement
 - A list of one or more statements enclosed within a pair of curly braces { and } or block is a compound statement
 - Control constructs like if, if-else, switch, for, while, do-while, goto, continue, break, return are statements

- Example: *Expression statements*

Expressions	Statements
<code>i + j</code>	<code>i + j;</code>
<code>k = i + j</code>	<code>k = i + j;</code>
<code>funct(i,j)</code>	<code>funct(i,j);</code>
<code>k = funct(i,j)</code>	<code>k = funct(i,j);</code>

- Example: *Compound statements*

```
{  
    int i = 2, j = 3, t;  
  
    t = i;  
    i = j;  
    j = t;  
}
```



Control Constructs

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- These statements control the flow based on conditions:

- Selection-statement*: if, if-else, switch
- Labeled-statement*: Statements labeled with identifier, case, or default
- Iteration-statement*: for, while, do-while
- Jump-statement*: goto, continue, break, return

- Examples:

```
if (a < b) {  
    int t;  
  
    t = a;  
    a = b;  
    b = t;  
}
```

```
if (x < 5)  
    x = x + 1;  
else {  
    x = x + 2;  
    --y;  
}
```

```
switch (i) {  
    case 1: x = 5;  
        break;  
    case 3: x = 10;  
    default: x = 15;  
}
```

```
int sum = 0;  
for(i = 0; i < 5; ++i) {  
    int j = i * i;  
    sum += j;  
}
```

```
while (n) {  
    sum += n;  
    if (sum > 20)  
        break;  
    --n;  
}
```

```
int f(int x, int y)  
{  
    return x + y;  
}
```



Module 01: End of Lecture 01

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- Recap of C features
 - Data types
 - Variables
 - Literals
 - Operators
 - Expressions
 - Statements
 - Control Constructs – Conditional Flow & Loops



Module 01: Lecture 02

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- Recap of C features

- Arrays
- Structures
- Unions
- Pointers



Arrays

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- An array is a collection of data items, all of the same type, accessed using a common name

- Declare Arrays:

```
#define SIZE 10
int name[SIZE];    // SIZE must be an integer constant greater than zero
double balance[10];
```

- Initialize Arrays:

```
int primes[5] = {2, 3, 5, 7, 11}; // Size = 5
```

```
int primes[] = {2, 3, 5, 7, 11};
int sizeofPrimes = sizeof(primes)/sizeof(int); // size is 5 by initialization
```

```
int primes[5] = {2, 3};           // Size = 5, last 3 elements set to 0
```

- Access Array elements:

```
int primes[5] = {2, 3};
int EvenPrime = primes[0]; // Read 1st element
primes[2] = 5;             // Write 3rd element
```

- Multidimensional Arrays:

```
int mat[3][4];

for(i = 0; i < 3; ++i)
    for(j = 0; j < 4; ++j)
        mat[i][j] = i + j;
```



Structures

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- A structure is a collection of data items of different types. Data items are called *members*. The size of a structure is the sum of the size of its members.

- **Declare Structures:**

```
struct Complex { // Complex Number
    double re;    // Real component
    double im;    // Imaginary component
} c;             // c is a variable of struct Complex type
printf("size = %d\n", sizeof(struct Complex)); // Prints: size = 16
```

```
typedef struct _Books {
    char title[50];    // data member
    char author[50];   // data member
    int book_id;       // data member
} Books; // Books is an alias for struct _Books type
```

- **Initialize Structures:**

```
struct Complex x = {2.0, 3.5}; // Both members
struct Complex y = {4.2};      // Only the first member
```

- **Access Structure members:**

```
struct Complex x = {2.0, 3.5};
double norm = sqrt(x.re*x.re + x.im*x.im); // Using . (dot) operator
```

```
Books book;
book.book_id = 6495407;
strcpy(book.title, "C Programming");
```



Unions

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- A union is a special structure that allocates memory only for the largest data member and holds only one member as a time

- **Declare Union:**

```
typedef union _Packet { // Mixed Data Packet
    int    iData;        // integer data
    double dData;        // floating point data
    char   cData;        // character data
} Packet;
printf("size = %d\n", sizeof(Packet)); // Prints: size = 8
```

- **Initialize Union:**

```
Packet p = {10}; // Initialize only with a value of the type of first member
printf("iData = %d\n", p.iData); // Prints: iData = 10
```

- **Access Union members:**

```
p.iData = 2;
printf("iData = %d\n", p.iData); // Prints: iData = 2
p.dData = 2.2;
printf("dData = %lf\n", p.dData); // Prints: dData = 2.200000
p.cData = 'a';
printf("cData = %c\n", p.cData); // Prints: cData = a

p.iData = 97;
printf("iData = %d\n", p.iData); // Prints: iData = 97
printf("dData = %lf\n", p.dData); // Prints: dData = 2.199999
printf("cData = %c\n", p.cData); // Prints: cData = a
```



Pointers

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- A pointer is a variable whose value is a memory address
- The type of a pointer is determined by the type of its pointee

```
int    *ip;    // pointer to an integer
double *dp;    // pointer to a double
float  *fp;    // pointer to a float
char   *ch     // pointer to a character
```

- Using a pointer:

```
int main() {
    int i = 20;    // variable declaration
    int *ip;       // pointer declaration
    ip = &i;       // store address of i in pointer

    printf("Address of variable: %p\n", &i); // Prints: Address of variable : 00A8F73C

    printf("Value of pointer: %p\n", ip);    // Prints: Value of pointer : 00A8F73C

    printf("Value of pointee: %d\n", *ip);   // Prints: Value of pointee : 20

    return 0;
}
```



Pointers

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

● Pointer-Array Duality

```
int a[] = {1, 2, 3, 4, 5};
int *p;

p = a;
printf("a[0] = %d\n", *p);    // a[0] = 1
printf("a[1] = %d\n", **p);   // a[1] = 2
printf("a[2] = %d\n", *(p+1)); // a[2] = 3

p = &a[2];
*p = -10;
printf("a[2] = %d\n", a[2]);  // a[2] = -10
```

● malloc-free

```
int *p = (int *)malloc(sizeof(int));

*p = 0x8F7E1A2B;
printf("%X\n", *p);    // 8F7E1A2B

unsigned char *q = p;
printf("%X\n", *q++);  // 2B
printf("%X\n", *q++);  // 1A
printf("%X\n", *q++);  // 7E
printf("%X\n", *q++);  // 8F

free(p);
```

● Pointer to a structure

```
struct Complex { // Complex Number
    double re;    // Real component
    double im;    // Imaginary component
} c = { 0.0, 0.0 };

struct Complex *p = &c;

(*p).re = 2.5;
p->im = 3.6;

printf("re = %lf\n", c.re); // re = 2.500000
printf("im = %lf\n", c.im); // im = 3.600000
```

● Dynamically allocated arrays

```
int *p = (int *)malloc(sizeof(int)*3);

p[0] = 1; p[1] = 2; p[2] = 3;

printf("p[1] = %d\n", *(p+1)); // p[1] = 2
free(p);
```



Module 01: End of Lecture 02

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- Recap of C features

- Arrays
- Structures
- Unions
- Pointers



Module 01: Lecture 03

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- Recap of C features
 - Functions
 - Input / Output
- C Standard Library
- Source Organization for a C program
- Build Process



Functions

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- A function performs a specific task or computation
 - Has 0, 1, or more parameters / arguments. Every argument has a type (void for no argument)
 - May or may not return a result. Return value has a type (void for no result)
 - Function declaration:

```
// Function Prototype / Header / Signature
// Name of the function: funct
// Parameters: x and y. Types of parameters: int
// Return type: int
```

```
int funct(int x, int y);
```

- Function definition:

```
// Function Implementation
int funct(int x, int y)
```

```
// Function Body
{
    return (x + y);
}
```



Functions

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- **Call-by-value** mechanism for passing arguments. The value of an actual parameter copied to the formal parameter
- **Return-by-value** mechanism to return the value, if any.

```
int funct(int x, int y) {
    ++x; ++y;                // Formal parameters changed
    return (x + y);
}

int main() {
    int a = 5, b = 10, z;

    printf("a = %d, b = %d\n", a, b); // prints: a = 5, b = 10

    z = funct(a, b); // function call by value
                    // a copied to x. x becomes 5
                    // b copied to y. y becomes 10
                    // x in funct changes to 6 (++x)
                    // y in funct changes to 11 (++y)
                    // return value (x + y) copied to z

    printf("funct = %d\n", z); // prints: funct = 17

    // Actual parameters do not change on return (call-by-value)
    printf("a = %d, b = %d\n", a, b); // prints: a = 5, b = 10

    return 0;
}
```



Functions

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- A function may be recursive (call itself)
 - Has recursive step/s
 - Has exit condition/s
- Example:

```
// Factorial of n
unsigned int factorial(unsigned int n) {
    if (n > 0)
        return n * factorial(n - 1); // Recursive step
    else
        return 1; // Exit condition
}

// Number of 1's in the binary representation of n
unsigned int nOnes(unsigned int n) {
    if (n == 0)
        return 0; // Exit condition
    else // Recursive steps
        if (n % 2 == 0)
            return nOnes(n / 2);
        else
            return nOnes(n / 2) + 1;
}
```



Function pointers

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

```
#include <stdio.h>
struct GeoObject {
    enum { CIR = 0, REC, TRG } gCode;
    union {
        struct Cir { double x, y, r; } c;
        struct Rec { double x, y, w, h; } r;
        struct Trg { double x, y, b, h; } t;
    };
};
```

```
typedef void(*DrawFunc) (struct GeoObject);
```

```
void drawCir(struct GeoObject go) {
    printf("Circle: (%lf, %lf, %lf)\n",
        go.c.x, go.c.y, go.c.r); }
```

```
void drawRec(struct GeoObject go) {
    printf("Rect: (%lf, %lf, %lf, %lf)\n",
        go.r.x, go.r.y, go.r.w, go.r.h); }
```

```
void drawTrg(struct GeoObject go) {
    printf("Triag: (%lf, %lf, %lf, %lf)\n",
        go.t.x, go.t.y, go.t.b, go.t.h); }
```

```
DrawFunc DrawArr[] = { // Array of func. ptrs
    drawCir, drawRec, drawTrg };
```

```
int main() {
    struct GeoObject go;
```

```
    go.gCode = CIR;
    go.c.x = 2.3; go.c.y = 3.6;
    go.c.r = 1.2;
    DrawArr[go.gCode](go); // Call by ptr
```

```
    go.gCode = REC;
    go.r.x = 4.5; go.r.y = 1.9;
    go.r.w = 4.2; go.r.h = 3.8;
    DrawArr[go.gCode](go); // Call by ptr
```

```
    go.gCode = TRG;
    go.t.x = 3.1; go.t.y = 2.8;
    go.t.b = 4.4; go.t.h = 2.7;
    DrawArr[go.gCode](go); // Call by ptr
```

```
    return 0;
```

```
}
```

```
Circle: (2.300000, 3.600000, 1.200000)
Rect: (4.500000, 1.900000, 4.200000, 3.800000)
Triag: (3.100000, 2.800000, 4.400000, 2.700000)
```



Input / Output

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- `int printf(const char *format, ...)` writes to stdout by the format and returns the number of characters written
- `int scanf(const char *format, ...)` reads from stdin by the format and returns the number of characters read
- Use `%s`, `%d`, `%c`, `%lf`, to print/scan string, int, char, double

```
#include <stdio.h>
```

```
int main() {
```

```
    char str[100];
```

```
    int i;
```

```
    printf("Enter a value :");           // prints a constant string
```

```
    scanf("%s %d", str, &i);             // scans a string value and an integer value
```

```
    printf("You entered: %s %d ", str, i); // prints string and integer
```

```
    return 0;
```

```
}
```



Input / Output

- To write to or read from file:

```
#include <stdio.h>

int main() {

    FILE *fp = NULL;
    int i;

    fp = fopen("Input.dat", "r");
    fscanf(fp, "%d", &i);          // scan from Input.dat
    fclose(fp);

    fp = fopen("Output.dat", "w");
    fprintf("%d^2 = %d\n", i, i*i); // prints to Output.dat
    fclose(fp);

    return 0;
}
```

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary



C Standard Library

● Common Library Components:

Component	Data Types, Manifest Constants, Macros, Functions, ...
stdio.h	Formatted and un-formatted file input and output including functions <ul style="list-style-type: none">• printf, scanf, fprintf, fscanf, sprintf, sscanf, feof, etc.
stdlib.h	Memory allocation, process control, conversions, pseudo-random numbers, searching, sorting <ul style="list-style-type: none">• malloc, free, exit, abort, atoi, strtold, rand, bsearch, qsort, etc.
string.h	Manipulation of C strings and arrays <ul style="list-style-type: none">• strcat, strcpy, strcmp, strlen, strtok, memcpy, memmove, etc.
math.h	Common mathematical operations and transformations <ul style="list-style-type: none">• cos, sin, tan, acos, asin, atan, exp, log, pow, sqrt, etc.
errno.h	Macros for reporting and retrieving error conditions through error codes stored in a static memory location called errno <ul style="list-style-type: none">• EDOM (parameter outside a function's domain – sqrt(-1)),• ERANGE (result outside a function's range), or• EILSEQ (an illegal byte sequence), etc.

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary



Source Organization for a C program

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

Header Files

- A header file has extension `.h` and contains C function declarations and macro definitions to be shared between several source files
- There are two types of header files:
 - Files that the programmer writes
 - Files from standard library
- Header files are included using the `#include` pre-processing directive
 - `#include <file>` for system header files
 - `#include "file"` for header files of your own program



Source Organization for a C program

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

● Example:

```
// Solver.h -- Header files
int quadraticEquationSolver(double, double, double, double*, double*);

// Solver.c -- Implementation files
#include "Solver.h"

int quadraticEquationSolver(double a, double b, double c, double* r1, double* r2) {
    // ...
    // ...
    // ...
    return 0;
}

// main.c -- Application files
#include "Solver.h"

int main() {
    double a, b, c;
    double r1, r2;

    int status = quadraticEquationSolver(a, b, c, &r1, &r2);

    return 0;
}
```



Build Flow

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

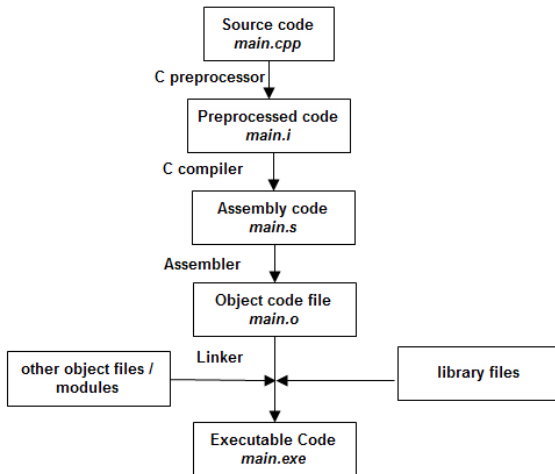
Std Library

Organization

Build Process

References

Summary





Build Process

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types
Variables
Literals
Operators
Expressions
Statements
Control Flow
Arrays
Structures
Unions
Pointers
Functions
Input / Output

Std Library

Organization

Build Process

References

Summary

- C Pre-processor (CPP) substitutes and includes functions, headers and macros before compilation

```
int sum(int, int);  
int main() {  
    int a = sum(1,2);  
    return a;  
}
```

- The compiler translates the pre-processed C code into assembly language, which is a machine level code that contains instructions that manipulate the memory and processor directly
- The linker links our program with the pre-compiled libraries for using their functions
- In the running example, `function.c` and `main.c` are first compiled and then linked

```
int sum(int a,int b) { return a+b; }  
  
int main() {  
    int a = sum(1,2); // as files are linked, uses functions directly  
    return a;  
}
```



Tools

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- Development IDE: Code::Blocks 16.01
- Compiler: `-std=c++98` and `-std=c99`



References

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- Kernighan, Brian W., and Dennis M. Richie. The C Programming Language. Vol. 2. Englewood Cliffs: Prentice-Hall, 1988.
- King, Kim N., and Kim King. C programming: A Modern Approach. Norton, 1996.



Module Summary

Module 01

Partha Pratim Das

Objectives & Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

- Revised the concept of variables and literals in C
- Revised the various data types and operators of C
- Re-iterated through the control constructs of C
- Re-iterated through the concepts of functions and pointers of C
- Re-iterated through the program organization of C and the build process.



Instructor and TAs

Module 01

Partha Pratim
Das

Objectives &
Outline

Recap of C

Data Types

Variables

Literals

Operators

Expressions

Statements

Control Flow

Arrays

Structures

Unions

Pointers

Functions

Input / Output

Std Library

Organization

Build Process

References

Summary

Name	Mail	Mobile
Partha Pratim Das, <i>Instructor</i>	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, <i>TA</i>	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, <i>TA</i>	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, <i>TA</i>	himadribhuyan@gmail.com	9438911655