



INTRODUCTION TO ANSI C

Dr. Chokchai (Box) Leangsuksun

Louisiana Tech University

Original slides were created by Dr. Tom Chao Zhou, <https://www.cse.cuhk.edu.hk>

1



Outline

- History
- Introduction to C
 - Basics
 - If Statement
 - Loops
 - Functions
 - Switch case
 - Pointers
 - Structures
 - File I/O

Louisiana Tech University



Introduction

- The C programming language was designed by Dennis Ritchie at Bell Laboratories in the early 1970s
- Influenced by
 - ALGOL 60 (1960),
 - CPL (Cambridge, 1963),
 - BCPL (Martin Richard, 1967),
 - B (Ken Thompson, 1970)
- Traditionally used for systems programming, though this may be changing in favor of C++
- Traditional C:
 - *The C Programming Language*, by Brian Kernighan and Dennis Ritchie, 2nd Edition, Prentice Hall
 - Referred to as *K&R*

Original by Fred Kuhns)

Louisiana Tech University



Standard C

- Standardized in 1989 by ANSI (American National Standards Institute) known as ANSI C
- International standard (ISO) in 1990 which was adopted by ANSI and is known as **C89**
- As part of the normal evolution process the standard was updated in 1995 (**C95**) and 1999 (**C99**)
- C++ and C
 - C++ extends C to include support for Object Oriented Programming and other features that facilitate large software development projects
 - C is not strictly a subset of C++, but it is possible to write "*Clean C*" that conforms to both the C++ and C standards.

Original by Fred Kuhns)

Louisiana Tech University



Elements of a C Program

- A C development environment:
 - *Application Source*: application source and header files
 - *System libraries and headers*: a set of standard libraries and their header files. For example see `/usr/include` and `glibc`.
 - *Compiler*: converts source to object code for a specific platform
 - *Linker*: resolves external references and produces the executable module
- User program structure
 - there must be one main function where execution begins when the program is run. This function is called main
 - `int main (void) { ... },`
 - `int main (int argc, char *argv[]) { ... }`
 - UNIX Systems have a 3rd way to define main(), though it is not POSIX.1 compliant
 - `int main (int argc, char *argv[], char *envp[])`
 - additional local and external functions and variables

Original by Fred Kuhns)

Louisiana Tech University



A Simple C Program

- Create example file: `try.c`
- Compile using gcc:


```
gcc -o try try.c
```
- The standard C library *libc* is included automatically
- Execute program


```
./try
```
- Note, I always specify an absolute path
- Normal termination:


```
void exit(int status);
```

 - calls functions registered with `atexit()`
 - flush output streams
 - close all open streams
 - return status value and control to host environment

```
/* you generally want to
 * include stdio.h and
 * stdlib.h
 */
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
    printf("Hello World\n");
    exit(0);
}
```

Original by Fred Kuhns)

Louisiana Tech University



Source and Header files

- Header files (*.h) export interface definitions
 - function prototypes, data types, macros, inline functions and other common declarations
- Do not place source code (i.e. definitions) in the header file with a few exceptions.
 - inline'd code
 - class definitions for C++
 - const definitions
- *C preprocessor* (cpp) is used to insert common definitions into source files
- There are other cool things you can do with the preprocessor

Original by Fred Kuhns)

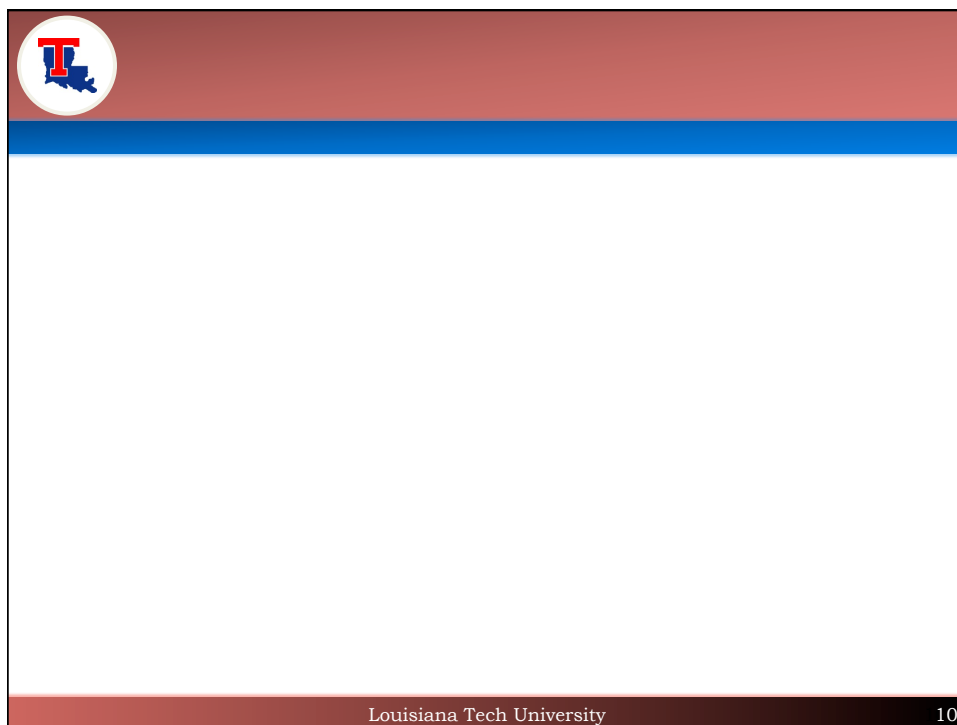
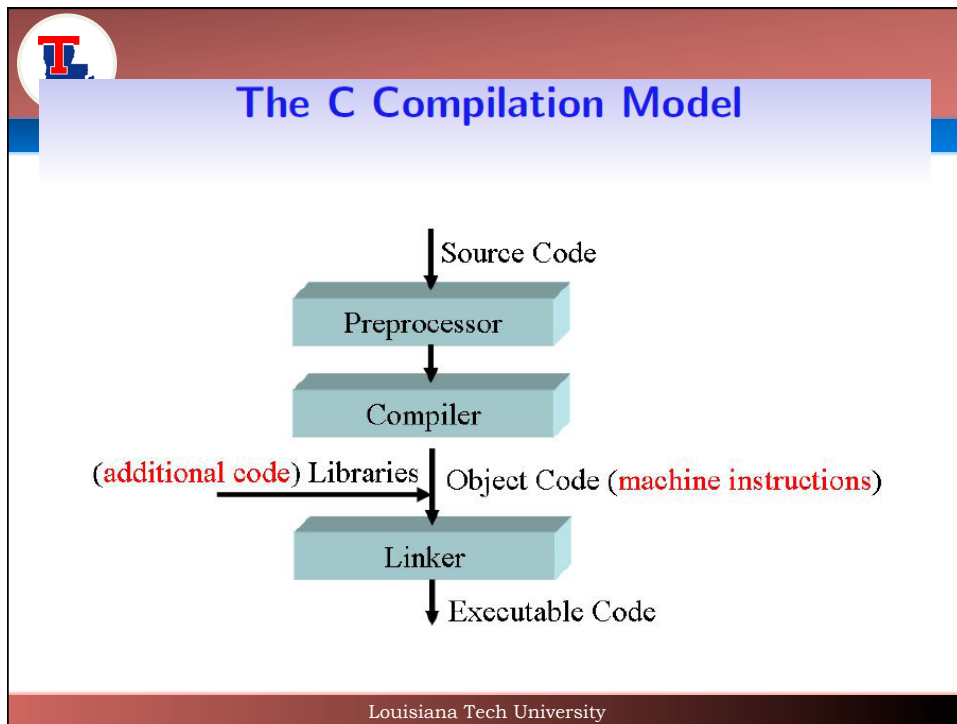
Louisiana Tech University



Introduction to C

- Basics
- If Statement
- Loops
- Functions
- Switch case
- Pointers
- Structures
- File I/O

Louisiana Tech University





Introduction to C: Basics

```
//a simple program that has variables
#include <stdio.h>
int main()
{
    int x;
    char y; //
    float z; //
    double t; //
    printf("hello world...\n");
    printf("size of int %d, size of char %d, size of float
    %d and size of double %d\n", sizeof (int), sizeof(char),
    sizeof(float), sizeof(double));
    return 0;
}
```

Louisiana Tech University



From the example

- C program must have the main() at least.
- Comments
 - // comments
 - Text surrounded by /* and */ is ignored by computer
 - Document some details to explain the program
- **#include <stdio.h>**
 - Preprocessor directive
 - Tells computer to load contents of a certain file
 - **<stdio.h>** allows standard input/output operations
 - Ex: printf & scanf

Louisiana Tech University

12



Introduction to C: Basics

```
//reading input from console
#include <stdio.h>
int main()
{
    int num1;
    int num2;
    printf( "Please enter two numbers: " );
    scanf( "%d %d", &num1,&num2 );
    printf( "You entered %d %d", num1, num2 );
    return 0;
}
```

Louisiana Tech University



keywords

- | | | |
|------------|------------|------------|
| • auto | • char | • default |
| • double | • extern | • goto |
| • int | • return | • sizeof |
| • long | • union | • volatile |
| • break | • const | • do |
| • else | • float | • if |
| • long | • short | • static |
| • switch | • unsigned | • while |
| • case | • continue | |
| • enum | • for | |
| • register | • signed | |
| • typedef | • void | |

Louisiana Tech University

14



Variables

- *storage-class-specifier type-specifier variable-names,...*

```
int bob=32;
```

```
char loop1,loop2,loop3='A';
```

```
typedef char boolean;
```

```
boolean yes=1;
```



variable

```
int bob=32;
```

- Creates variable "bob" and initializes it to the value 32.

```
char loop1,loop2,loop3='\x41';
```

```
char loop1,loop2,loop3='A';
```

- Creates three variables. The value of "loop1" and "loop2" is undefined. The value of loop3 is the letter "A".



variable

```
typedef char boolean;
```

Causes the keyword "boolean" to represent variable-type "char".

```
boolean yes=1;
```

Creates variable "yes" as type "char" and sets its value to 1.

-



The storage-class-specifier

- **typedef**
- The symbol name "*variable-name*" becomes a type-specifier of type "*type-specifier*". No variable is actually created, this is merely for convenience.
- **extern** indicates that the variable is defined outside of the current file. This brings the variable's scope into the current scope. No variable is actually created by this.
- **static** causes a variable that is defined within a function to be preserved in subsequent calls to the function.
- **auto** Causes a local variable to have a local lifetime (default).
- **register** requests that the variable be accessed as quickly as possible. This request is not guaranteed. Normally, the variable's value is kept within a CPU register for maximum speed.



type-specifier

int, signed, signed int, or no type specifier

- Defines a signed integer. If no type specifier is given, then this is the default.

unsigned int, unsigned

- Same as int, but unsigned values only.

long, signed long, long int, signed long int

- Defines a long signed integer. May be twice the bit size as a normal int, or the same as a normal int.

unsigned long, unsigned long int

- Same as long, but unsigned values only.



type-specifier

float

- A floating-point number. Consists of a sign, a mantissa (number greater than or equal to 1), and an exponent. The mantissa is taken to the power of the exponent then given the sign. The exponent is also signed allowing extremely small fractions. The mantissa gives it a finite precision.

double

- A more accurate floating-point number than float. Normally twice as many bits in size.

long double

- Increases the size of double.



type-specifier

void

- Defines an empty or NULL value whose type is incomplete.

char, signed char

- Variable is large enough to store a basic character in the character set. The value is either signed or nonnegative.

unsigned char

- Same as char, but unsigned values only.

short, signed short, short int, signed short int

- Defines a short signed integer. May be the same range as a normal int, or half the bits of a normal int.

unsigned short, unsigned short int

- Defines an unsigned short integer.



array

- Array in C is 0 origin

```
int x[5];
x[0] = 3; x[4] = 7;
```

```
char str[16]="Blueberry";
```

- Creates a string. The value at str[8] is the character "y". The value at str[9] is the null character. The values from str[10] to str[15] are undefined.



array

```
char s []="abc";
```

- Size of the array s to 4 (just long enough to hold the string plus a null character), and stores the string in the array.

```
int y[3]={4};
```

Sets the value of y[0] to 4 and y[1] and y[2] to 0.

```
int joe[4][5]={  
  {1,2,3,4,5},  
  {6,7,8,9,10},  
  {11,12,13,14,15} };
```

- The first row initializes joe[0], the second row joe[1] and so forth. joe[3] is initialized to 5 zeros.

Louisiana Tech University

23



enum

```
enum identifier {enumerator-list};
```

- *Identifier* is a handle for identification, and is optional.
- *Enumerator-list* is a list of variables to be created. They will be constant integers. Each variable is given the value of the previous variable plus 1. The first variable is given the value of 0.

- Examples:

```
enum {joe, mary, bob, fran};
```

Louisiana Tech University

24



enum

- Examples:

```
enum {joe, mary, bob, fran};
```

- Creates 4 variables. The value of joe is 0, mary is 1, bob is 2, and fran is 3.

```
enum test {larry, floyd=20, ted};
```

- Creates 3 variables with the identifier test. The value of larry is 0, floyd is 20, and ted is 21.



constant

```
const float PI=3.141;
```

Causes the variable PI to be created with value 3.141. Any subsequent attempts to write to PI are not allowed.

```
const int joe=0xFFFF;
```

Causes joe to be created with the value of 65535 decimal.

```
const float penny=7.4e5;
```

Causes penny to be created with the value of 740000.000000.



String constant

"\x41" and "A" are the same string.

```
char fred[25]="He said, \"Go away!
\"";
```

The value at fred[9] is a double quote. The value at fred[20] is the null character.

-



sizeof

size_t **sizeof** *expression*

- *or*

size_t **sizeof** (*type*)

- The sizeof keyword returns the number of bytes of the given expression or type. **size_t** is an unsigned integer result.
- Example:

```
printf("The number of bytes in an int is %d.
\n",sizeof(int));
```

-



If then else

if(*expression*) *statement1*;

or

**if(*expression*) *statement1*;
else *statement2* ;**

or

**if(*expression*) *statement1*;
else if *statement2* ;
else *statement3* ;**



Introduction to C: if statement

```
#include <stdio.h>
int main()
{
    int age;           /* Need a variable... */
    printf( "Please enter your age" ); /* Asks for age */
    scanf( "%d", &age ); /* The input is put in age */
    if ( age < 100 )
    {
        /* If the age is less than 100 */
        printf( "You are pretty young!\n" ); /* Just to show you it works... */
    }
    else if ( age == 100 )
    {
        /* I use else just to show an example */
        printf( "You are old\n" );
    }
    else
    {
        printf( "You are really old\n" ); /* Executed if no other statement is */
    }
    return 0;
}
```



Introduction to C: Loops(for)

for(*expression1* ; *expression2* ; *expression3*) *statement...*

Examples:

```
for(loop=0;loop<1000;loop++)
printf("%i\n",loop);
or
for(loop=0;loop<1000;loop++) {
printf("%i\n",loop);
}
```

Louisiana Tech University



Introduction to C: Loops(for)

```
#include <stdio.h>
int main()
{
    int x;
    /* The loop goes while x < 10, and x increases by one every loop*/
    for ( x = 0; x < 10; x++ )
    {
        /* Keep in mind that the loop condition checks
        the conditional statement before it loops again.
        consequently, when x equals 10 the loop breaks.
        x is updated before the condition is checked. */
        printf( "%d\n", x );
    }
    return 0;
}
```

Louisiana Tech University



Introduction to C: Loops(while)

while(*expression*) *statement*...

- *statement* is executed repeatedly as long as *expression* is true. The test on *expression* takes place before each execution of *statement*.

Examples:

```
while(*pointer!='j') pointer++;
```

```
while(counter<5) {  
printf("counter=%i",counter);   counter++;  
}
```

Louisiana Tech University



Introduction to C: Loops(while)

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

```
int main()  
{  
    int x = 0; /* Don't forget to declare variables */  
    while ( x < 10 )  
    { /* While x is less than 10 */  
        printf( "%d\n", x );  
        x++; /* Update x so the condition can be met eventually */  
    }  
    return 0;  
}
```

Louisiana Tech University



Introduction to C: Loops(do while)

do *statement*... **while**(*expression*);

statement is executed repeatedly as long as *expression* is true. The test on *expression* takes place after each execution of *statement*.

Examples:

```
do {
    betty++;
    printf("%i",betty); } while (betty<100);
```

Louisiana Tech University



Introduction to C: Loops(do while)

```
#include <stdio.h>
int main()
{
    int x;
    x = 0;
    do
    {
        /* "Hello, world!" is printed at least one time
           even though the condition is false*/
        printf( "%d\n", x );
        x++;
    } while ( x != 10 );
    return 0;
}
```

Louisiana Tech University



goto & label

```
goto label;
....
label:
```

Examples:

```
goto skip_point;
printf("This part was skipped.\n");
skip_point:
printf("Hi there!\n");
```

Louisiana Tech University

37



Introduction to C: Loops and continue

The continue statement can only appear in a loop body. It causes the rest of the statement body in the loop to be skipped.

```
#include <stdio.h>                                0
int main()                                          1
{
    int x;                                         2
    for(x=0;x<10;x++)                             3
    {
        if(x==5)                                  4
        {
            continue;                             5
        }
        printf("%d\n",x);                          6
    }
    return 0;                                     7
}
```

Louisiana Tech University



Examples: continue

```

:
for(loop=0;loop<100;loop++) {
    if(loop==50) continue;
    printf("%i\n",loop);
}

```

The numbers 0 through 99 are printed except for 50.



Introduction to C: Loops & break

The break statement can only appear in a switch body or a loop body. It causes the execution of the current enclosing switch or loop body to terminate.

```

#include <stdio.h>
int main()
{
    int x;
    for(x=0;x<10;x++)
    {
        if(x==5)
        {
            break;
        }
        printf("%d\n",x);
    }
    return 0;
}

```




switch & case

```
switch ( variable )
{
case const:
    statements...;
    :
default:
    statements...; }
```



Examples: switch & case


```
switch(betty) {
case 1:
    printf("betty=1\n");
case 2:
    printf("betty=2\n");
    break;
case 3:
    printf("betty=3\n");
    break;
default:
    printf("Not sure.\n");
}
```



```

#include <stdio.h>
//function declaration, need to define the function body in other places
void playgame();
void loadgame();
void playmultiplayer();
int main()
{
    int input;
    printf( "1. Play game\n" );
    printf( "2. Load game\n" );
    printf( "3. Play multiplayer\n" );
    printf( "4. Exit\n" );
    printf( "Selection: " );
    scanf( "%d", &input );
    switch ( input ) {
        case 1:          /* Note the colon, not a semicolon */
            playgame();
            break;        //don't forget the break in each case
        case 2:
            loadgame();
            break;
        case 3:
            playmultiplayer();
            break;
        case 4:
            printf( "Thanks for playing!\n" );
            break;
        default:
            printf( "Bad input, quitting!\n" );
            break;
    }
    return 0;
}

```



Introduction to C: function

```

#include <stdio.h>

//function declaration
int mult ( int x, int y );
int main()
{
    int x;
    int y;
    printf( "Please input two numbers to be multiplied: " );
    scanf( "%d", &x );
    scanf( "%d", &y );
    printf( "The product of your two numbers is %d\n", mult( x, y ) );
    return 0;
}

//define the function body
//return value: int
//utility: return the multiplication of two integer values
//parameters: take two int parameters
int mult (int x, int y)
{
    return x * y;
}

```

Louisiana Tech University



return

return *expression*;

- Examples:

```
int alice(int x, int y)
{
    if(x<y)
        return(1);
    else
        return(0);
}
```

Louisiana Tech University

45



Introduction to C: pointer variables

- Pointer variables are variables that store memory addresses.
- Pointer Declaration:

```
int x, y = 5; /* regular integer var */

int *ptr;
/*ptr is a POINTER to an integer
variable*/
```

Louisiana Tech University



Introduction to C: pointer variables

```
int x, y = 5;
int *ptr;
/*ptr is a POINTER to an integer
variable*/
```

- Reference operator &:

```
ptr = &y;
/*assign ptr to the MEMORY ADDRESS of y.*/
```

- Dereference operator *:

```
x = *ptr;
/*assign x to the int that is pointed to by ptr
*/
```

Louisiana Tech University



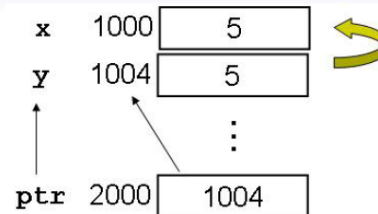
Introduction to C: pointer variables

Pointer Example 1


```
int x;
int y = 5;
int *ptr;

ptr = &y;

x = *ptr;
```



Louisiana Tech University



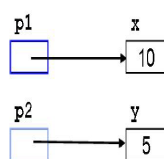
Introduction to C: pointer variables

Pointer Example 2

```

int x = 10, y = 5;
int *p1, *p2;
p1 = &x;
p2 = &y;

```



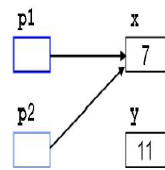
1000	10	x
1004	5	y
⋮		
2000	1000	p1
2004	1004	p2

Pointer Example 2

```


p2 = p1; // Not the same as *p2 = *p1

```



1000	7	x
1004	11	y
⋮		
2000	1000	p1
2004	1000	p2

Louisiana Tech University

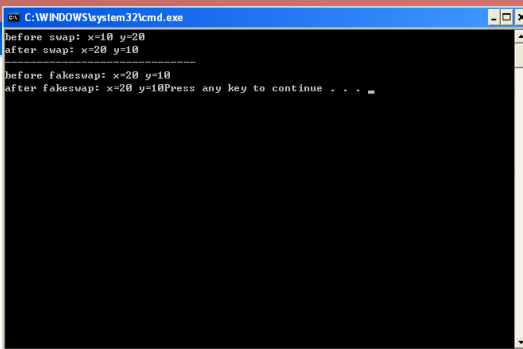


```

#include <stdio.h>
//swap two values
void swap(int* iPtrX, int* iPtrY);
void fakeswap(int x, int y);
int main()
{
    int x = 10;
    int y = 20;
    int *p1 = &x;
    int *p2 = &y;
    printf("before swap: x=%d y=%d\n", x, y);
    swap(p1, p2);
    printf("after swap: x=%d y=%d\n", x, y);

    printf("-----\n");
    printf("before fakeswap: x=%d y=%d\n", x, y);
    fakeswap(x, y);
    printf("after fakeswap: x=%d y=%d\n", x, y);
    return 0;
}
void swap(int* iPtrX, int* iPtrY)
{
    int temp;
    temp = *iPtrX;
    *iPtrX = *iPtrY;
    *iPtrY = temp;
}
void fakeswap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}

```



Louisiana Tech University



Introduction to C: struct

Structures and unions provide a way to group common variables together. A main use is for records.

```
struct structure-name {  
    variables,...  
}  
structure-variables,...;  
  
struct structure-name new-structure-variable;
```

Louisiana Tech University



example: struct

```
struct my-structure {  
    int fred[5];  
    char wilma, betty;  
    float barny=1; };  
struct my-structure account1;  
or  
struct my-structure {  
    int fred[5];  
    char wilma, betty;  
    float barny=1; } account1;
```

Louisiana Tech University



example: struct

```
#include <stdio.h>
//group things together
struct database {
    int id_number;
    int age;
    float salary;
};

int main()
{
    struct database employee;
    employee.age = 22;
    employee.id_number = 1;
    employee.salary = 12000.21;
}
```

Louisiana Tech University



union

- Unions work in the same way as structures except that all variables are contained in the same location in memory.
- Enough space is allocated for only the largest variable in the union.
- All other variables must share the same memory location. Unions are defined using the union keyword.

Louisiana Tech University

54



union

```
union my-union {
    char character_num;
    int integer_num;
    long long_num;
    float float_num;
    double double_num; } number;
```

- This defines the union number and allocates just enough space for the variable double_num.

```
number.integer_num=1;
Sets the value of integer_num to "1".
number.float_num=5;
Sets the value of float_num to "5".
```

Louisiana Tech University

55

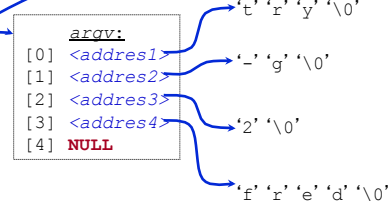


Passing Command Line Arguments

```
./try -g 2 fred
```

- When you execute a program you can include arguments on the command line.
- The run time environment will create an argument vector.
 - argv is the argument vector
 - argc is the number of arguments
- Argument vector is an array of pointers to strings.
- a *string* is an array of characters terminated by a binary 0 (NULL or '\0').
- argv[0] is always the program name, so argc is at least 1.

```
argc = 4,
argv = <address0>
```



Original by Fred Kuhns)

Louisiana Tech University



Standard Header Files you may want to use

- Standard Headers you should know about:
 - `stdio.h` – file and console (also a file) IO: *perror, printf, open, close, read, write, scanf, etc.*
 - `stdlib.h` - common utility functions: *malloc, calloc, strtol, atoi, etc*
 - `string.h` - string and byte manipulation: *strlen, strcpy, strcat, memcpy, memset, etc.*
 - `ctype.h` – character types: *isalnum, isprint, isupport, tolower, etc.*
 - `errno.h` – defines *errno* used for reporting system errors
 - `math.h` – math functions: *ceil, exp, floor, sqrt, etc.*
 - `signal.h` – signal handling facility: *raise, signal, etc*
 - `stdint.h` – standard integer: *intN_t, uintN_t, etc*
 - `time.h` – time related facility: *asctime, clock, time_t, etc.*

Original by Fred Kuhns)

Louisiana Tech University



The Preprocessor

- The C preprocessor permits you to define simple macros that are evaluated and expanded prior to compilation.
- Commands begin with a `'#'`. Abbreviated list:
 - `#define` : defines a macro
 - `#undef` : removes a macro definition
 - `#include` : insert text from file
 - `#if` : conditional based on value of expression
 - `#ifdef` : conditional based on whether macro defined
 - `#ifndef` : conditional based on whether macro is not defined
 - `#else` : alternative
 - `#elif` : conditional alternative
 - `defined()` : preprocessor function: 1 if name defined, else 0

```
#if defined(__NetBSD__)
```

Original by Fred Kuhns)

Louisiana Tech University



Preprocessor: Macros

- Using macros as functions, exercise caution:
 - **flawed example:** `#define mymult(a,b) a*b`
 - Source: `k = mymult(i-1, j+5);`
 - Post preprocessing: `k = i - 1 * j + 5;`
 - **better:** `#define mymult(a,b) (a)*(b)`
 - Source: `k = mymult(i-1, j+5);`
 - Post preprocessing: `k = (i - 1)*(j + 5);`
- Be careful of **side effects**, for example what if we did the following
 - **Macro:** `#define mysq(a) (a)*(a)`
 - **flawed usage:**
 - Source: `k = mysq(i++)`
 - Post preprocessing: `k = (i++)*(i++)`
- Alternative is to use inline'd functions
 - `inline int mysq(int a) {return a*a};`
 - `mysq(i++)` works as expected in this case.

Original by Fred Kuhns)

Louisiana Tech University



Preprocessor: Conditional Compilation

- Its generally better to use inline'd functions
- Typically you will use the preprocessor to define constants, perform conditional code inclusion, include header files or to create shortcuts
- `#define DEFAULT_SAMPLES 100`
- `#ifdef __linux`


```
static inline int64_t
gettime(void) {...}
```
- `#elif defined(sun)`

```
static inline int64_t
gettime(void) {return (int64_t)gethrtime();}
```
- `#else`

```
static inline int64_t
gettime(void) {... gettimeofday()...}
```
- `#endif`

Original by Fred Kuhns)

Louisiana Tech University



```

//content in in.list
//foo 70
//bar 98
//biz 100
#include <stdio.h>

int main()
{
    FILE *ifp, *ofp;
    char *mode = "r+";
    char outputFilename[] = "out.list";
    char username[9];
    int score;
    ifp = fopen("in.list", mode);
    if (ifp == NULL) {
        fprintf(stderr, "Can't open input file in.list!\n");
        exit(1);
    }
    ofp = fopen(outputFilename, "w");
    if (ofp == NULL) {
        fprintf(stderr, "Can't open output file %s!\n", outputFilename);
        exit(1);
    }
    while (fscanf(ifp, "%s %d", username, &score) == 2) {
        fprintf(ofp, "%s %d\n", username, score+10);
    }
    fclose(ifp);
    fclose(ofp);
    return 0;
}


```

mode:

- r - open for reading
- w - open for writing (file need not exist)
- a - open for appending (file need not exist)
- r+ - open for reading and writing, start at beginning
- w+ - open for reading and writing (overwrite file)
- a+ - open for reading and writing (append if file exists)

File I/O

Louisiana Tech University



References

- http://www.acm.uiuc.edu/webmonkeys/book/c_guide/index.html

Louisiana Tech University

62