

C Programming

Basics of a C program

Decisions and repetition

Common operations (Mathematical, relational, logical)

Arrays and strings

File input and output

Pointers and dynamic memory allocations

Random numbers

Bitwise operations

Functions

James Tam

The Benefits And Pitfalls Of C

Benefits

- It's a powerful language (doesn't restrict you)
- Fast
- A practical language (real applications)

Pitfalls

- It's a powerful language (No programmer fail-safes)
- Compiled code is not portable

James Tam

The Smallest Compilable C Program

```
main ()
{
}
```

James Tam

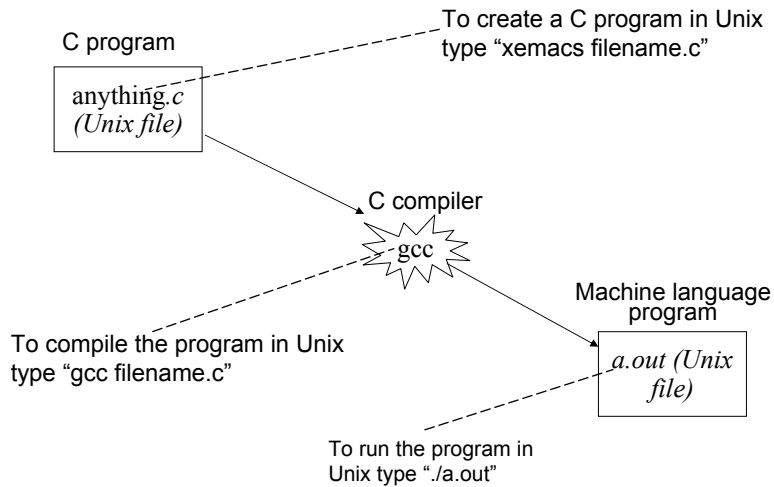
A Small Program With Better Style

```
/*
  Author: James Tam
  Date: march 24, 2003

  A slightly larger C program that follows good "C" conventions.
*/
int
main ()
{
  return(0);
}
```

James Tam

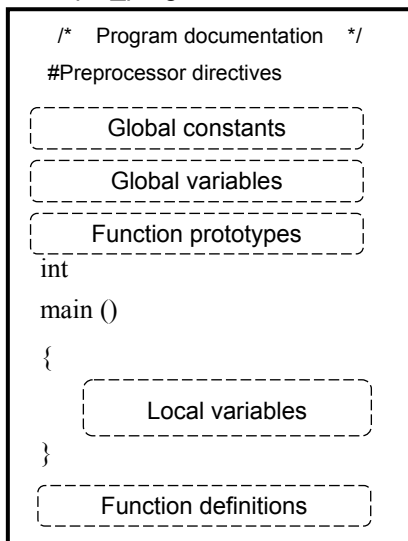
Creating And Compiling Programs: On The Computer Science Network



James Tam

The Basic Structure Of A C Program

example_program.c



James Tam

Some Variable Types

Type	Characteristics
char	8 bit character variable
int	32 bit signed integer value
float	32 bit real number value
double	64 bit real number value

James Tam

C Naming Conventions

- First character in a name must be alphabetic or an underscore
 - e.g., int num or char _ch;
- Subsequent characters can be any combination of alphabetic characters, numbers or the underscore.

James Tam

Constant Declarations

Identified by the keyword “const”

Capitalize the name of the constant

e.g., `int const UNCHANGEABLE = 10;`

e.g., `int const CONSTANT_VALUE = 10;`

`char const CH;`

James Tam

Constant Declarations

Identified by the keyword “const”

Capitalize the name of the constant

e.g., `int const UNCHANGEABLE = 10;`

e.g., `int const CONSTANT_VALUE = 10;`

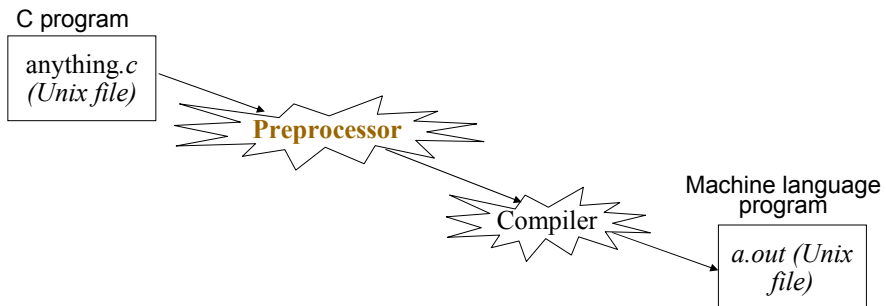
~~`char const CH;`~~

NO!

James Tam

Preprocessor Directives

Are substitutions that occur prior to compilation



Types of preprocessors directives

- #define
- #include

James Tam

Preprocessor “#define”

Performs substitutions

Format

```
#define <value to be substituted> <value to substitute>
```

Example

```
#define SIZE 10
```

James Tam

Example: Before The Preprocessor

```
/*
  A nonsense program to illustrate how the preprocessor substitution works.
*/

#define MYCONST 10

int
main ()
{
  int i = MYCONST;
  int j = 1;
  if (j == MYCONST)
    return (1);
  else
    return(0);
}
```

James Tam

Example: After The Preprocessor

```
/*
  A nonsense program to illustrate how the preprocessor substitution works.
*/

#define MYCONST 10

int
main ()
{
  int i = 10 ;
  int j = 1;
  if (j == 10 )
    return (1);
  else
    return(0);
}
```

James Tam

Preprocessor “#include”

Substitutes in the code from header files

Format:

```
#include <name of header file>
```

Example:

(Header file produced by AT&T)

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

(Give time, later examples may show how to create your own header files)

James Tam

Some Useful Header Files

Character handling: `ctype.h`

- `int isalnum (int c), int isalpha (int c)...`
- `return (0, ≠ 0)`

Mathematics: `math.h`

- `double exp (double x), double pow (double x, double y)...`

String handling: `string.h`

- `char *streat (char *s1, const char *s2)`
- `int strcmp (const char *s1, const char *s2)`

James Tam

Some Useful Header Files (2)

General utilities: **stdlib.h**

- void * malloc (n), calloc (n, sizeof_n), void free (*ptr)
- int rand (), void srand (time(NULL))

Input/Output: **stdio.h**

- Console input and output
- File input and output

James Tam

Displaying Output “printf”

Format:

```
printf(“const char *control_string”, arguments);
```

Example:

```
printf(“Simple output string”);  
printf(“ch=%c\n”, ch);  
printf(“num1=%d \t num2=%d”, num1, num2);
```

James Tam

Analysis Of Printf

Printf (“%c”, ch)

Format specifier: %c

Format code: c

Argument: ch

James Tam

Some Format Codes

Specifier	What the argument is displayed as
%c	Character
%d	Integer (decimal)
%o	Integer (octal)
%x	Integer (hexadecimal)
%f	Float
%lf	Long float (double)
%s	String
:	:

James Tam

More Output Examples

First group:

```
char ch = 'A';
int in = 100;
float fl = 1.1;
double db = 1.2;
char arr[10] = "hi there";
printf("ch=%c in=%d fl=%f db=%lf arr=%s\n", ch, in, fl, db, arr);
```

Second group:

```
printf("%d\n", ch);
printf("%c\n", in);
```

James Tam

Formatting Output

Setting the field width

Determining the number of places of precision

Justifying output

James Tam

Setting The Field Width

Add an integer between the “%” and the format code

Format:

```
printf(“%<width><format code>”, argument);
```

Example:

```
char arr1 [8] = "hello";  
char arr2 [8] = "there!";  
char arr3 [8] = "foo";  
char arr4 [8] = "choc";  
printf(“%8s %8s\n”, arr1, arr2);  
printf(“%8s %8s\n”, arr3, arr4);
```

James Tam

Determining Precision

An integer that follows the field width specifier (if there is one)

Format:

```
printf(“%<width>.<precision> <format code>”, argument);  
printf(“%.<precision> <format code>”, argument);
```

Example:

```
float f1 = 1.23;  
printf(“%2.1f %3f”, f1, f1);
```

James Tam

Justifying Output

Default is right justification

Left justify output by adding a negative integer between the “%” and the format code

Format:

```
printf(“%-<width><format code>”, argument);
```

Example:

```
char arr1 [8] = "hello";  
char arr2 [8] = "there!";  
char arr3 [8] = "foo";  
char arr4 [8] = "choc";  
printf(“%-8s %-8s\n”, arr1, arr2);  
printf(“%-8s %-8s\n”, arr3, arr4);
```

James Tam

Getting Input

Many different functions

Simplest (but not always the best) is scanf

Format:

```
scanf(“<control string>”, addresses);
```

Control String:

- Determines how the input will be interpreted
- Format codes determine how to interpret the input (see printf)

James Tam

Getting Input: A Common Problem & Work-Around

Example:

```
int num;
char ch;
printf("Enter an integer: ");
scanf("%d%c", &num, &ch);
printf("Enter a character: ");
scanf("%c", &ch);
```

James Tam

Getting Input: A Common Problem (Strings)

```
int const SIZE = 5;
char word [SIZE];
int i;
char ch;

for (i = 0; i < SIZE; i++)
    word[i] = -1;

printf("Enter a word no longer than 4 characters: ");
scanf("%s", word);

printf("Enter a character: ");
scanf("%c", &ch);

for (i = 0; i < SIZE; i++)
    printf("word[%d]=%c,%d ", i, word[i], word[i]);
printf("ch=%d,%c\n", ch);
```

James Tam

Getting Input: An Example Solution (Strings)

```
int const SIZE = 5;
char word [SIZE];
int i;
char ch;

for (i = 0; i < SIZE; i++)
    word[i] = -1;

printf("Enter a word no longer than 4 characters: ");
gets(word);

printf("Enter a character: ");
scanf("%c", &ch);

for (i = 0; i < SIZE; i++)
    printf("word[%d]=%c,%d ", i, word[i], word[i]);
printf("ch=%d,%c\n", ch);
```

James Tam

Decision Making Constructs

Almost identical to Java:

- If
- If-else
- If-else, else-if
- Switch

Keep in mind how true and false are usually defined

- = 0 (false)
- ≠ 0 (true)

e.g.,

- If (0)
- If (1)
- If (-1)

James Tam

The “?” Alternative

Used instead of if-else

Format:

<expression 1> ? <expression 1> : <expression 2>

Example:

```
x > 0 ? printf("x positive\n") : printf("x negative\n");
```

(Is equivalent to)

```
if (x > 0)
    printf("x positive\n");
else
    printf("x negative\n");
```

James Tam

Looping Constructs

Pre-test loops

- for
- while

Post-test loops

- do-while

Infinite loops:

- For (;)
- while (1)

James Tam

Mathematical Operators

Operator	Action	Example
-	Subtraction (also unary minus)	$x - y$, $-x * y$
+	Addition	$x + y$
*	Multiplication	$2 * 3$
/	Division	$10 / 2$
%	Modulus	$10 \% 3$
--	Decrement	--x (prefix) x-- (postfix)
++	Increment	++x (prefix) x++ (postfix)

James Tam

Relational Operators

Operator	Meaning of the operator
<	Less than
<=	Less than, equal to
>	Greater than
>=	Greater than, equal to
==	Equal to
!=	Not equal to

James Tam

Logical Operators

&&	Logical AND
	Logical OR
!	Logical NOT

Used to combine logical expressions

e.g., $(x > 0) \ \&\& \ (y > 0)$

James Tam

Arrays

Very similar to arrays in other languages

- An array of size n has an index from 0 to $n-1$

Statically allocated but:

- Initial size is determined at compile time (“cc”)
- Initial size can be determined at run time (“gcc”)

```
void
fun (int size)
{
    int i;
    int arr [size];
}
```

James Tam

Arrays (2)

Always remember that C provides no built-in bounds checking

```
int i;
int arr[4];
i = 0;
while (i <= 4)
{
    arr[i] = 0;
    i++;
}
```

James Tam

Strings

Character arrays that are NULL terminated

e.g.,

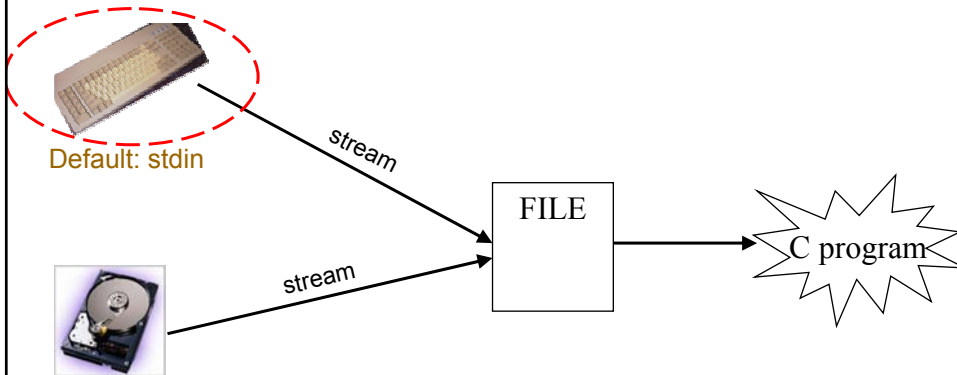
```
char arr[4] = "hi";
```

[0]	[1]	[2]	[3]
'h'	'i'	NULL	?

James Tam

File Input In C

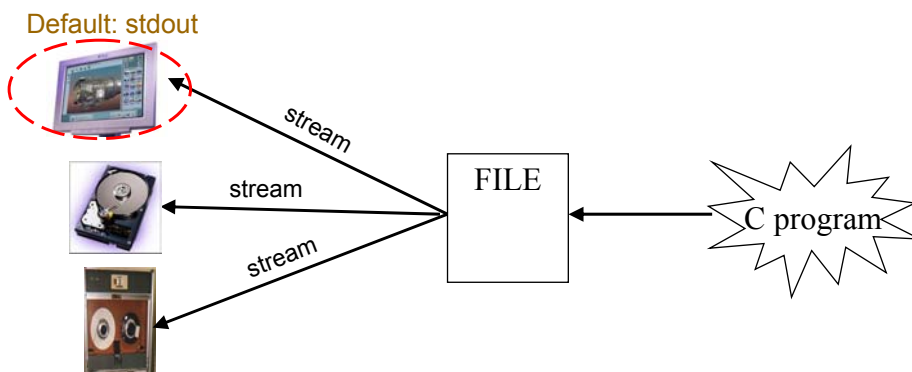
The C type, “FILE” allows for a consistent approach to input regardless of the actual physical device



James Tam

File Output In C

Similar to file input, the C type “FILE” allows for a consistent approach to output regardless of the actual physical device



James Tam

Declaring a FILE Variable

Format:

```
FILE *<name of pointer to file>;
```

Example:

```
FILE *fp;
```

Be sure to include the following preprocessors:

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

```
#include <stdlib.h>
```

James Tam

Opening Files

Format:

```
FILE * fopen (const char * filename, const char * mode);
```

Example:

```
fp = fopen ("data", "r");
```

```
fp = fopen ("/home/profs/tamj/data", "r");
```

Error conditions

- File does not exist
- Insufficient permissions to open the file

James Tam

Opening Files (2)

Filename

- Must reside in the same directory as C program being run
- Alternatively you can include the path to the file

Some File open modes	
Mode	Effect
r	Open text file for reading
w	Open text file for writing
a	Open text file for writing, start writing at the end
r+	Open file for reading and writing
w+	Open file for writing and reading

James Tam

Closing Files

Disassociates the file with the physical device

Remaining contents of the stream a flushed

Format:

```
int fclose (FILE * fp);
```

Example:

```
fclose(fp);
```

James Tam

Writing To A File “fprintf”

Format:

```
fprintf (FILE *fp, “const char *control_string”, arguments);
```

Example:

```
char arr[8] = “hiya”;  
fprintf(fp, “%s”, arr);
```

Similar to printf except you can also write to a file other than the screen (standard out)

James Tam

Reading From A File “fscanf”

Format:

```
fscanf (FILE *fp, “<control string>”, addresses);
```

Example:

```
char arr[8];  
fscanf (fp, “%s”, chunk);
```

Similar to scanf except you can also read from a file other than the keyboard (standard in)

James Tam

Reading From A File “fgets”

Format:

```
fgets(char *line, int n, FILE *fp);
```

Example:

```
fgets(chunk, 80, fp);
```

James Tam

File IO: First Example

The full example can be found in Unix:

```
/home/profs/tamj/233/examples/c_examples/files/file1.c
```

```
#include <stdio.h>
#include <stdlib.h>
int const LENGTH = 80;
main ()
{
    FILE *fp_in, *fp_out ;
    char arr[LENGTH];
    char fn_in [LENGTH];
    char fn_out [LENGTH];

    printf("Enter name of input file: ");
    scanf("%s", fn_in);
    printf("Enter name of output file: ");
    scanf("%s", fn_out);
```

James Tam

File IO: First Example (2)

```
fp_in = fopen(fn_in, "r");
fp_out = fopen(fn_out, "w");

if (!fp_in)
{
    printf("Can't open input file\n");
    return(-1);
}

if (!fp_out)
{
    printf("Can't open output file\n");
    return(-1);
}
```

James Tam

File IO: First Example (3)

```
while (feof(fp_in) == 0)
{
    // Using fscanf it stops scanning at whitespace
    fscanf(fp_in, "%s", arr);
    fprintf(fp_out, "%s-", arr);
}
return(0);
}
```

James Tam

File IO: Second Example

The full example can be found in Unix:

`/home/profs/tamj/233/examples/c_examples/files/file2.c`

This example is identical to the first example except for the loop:

```
while (feof(fp_in) == 0)
{
    // Grabs whole line up 79 characters or the end-of-file (whichever comes first)
    fgets (arr, 80, fp_in);

    // Writes out to output file as-is.
    fprintf (fp_out, "%s", arr);
}
```

James Tam

File IO: Third Example

The full example can be found in Unix:

`/home/profs/tamj/233/examples/c_examples/files/file3.c`

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

int const LENGTH = 80;
main ()
{
    FILE *fp_in;
    char arr[LENGTH];
    char fn_in [LENGTH];

    printf("Enter name of input file: ");
    scanf("%s", fn_in);
```

James Tam

File IO: Third Example (2)

```
fp_in = fopen(fn_in, "r");
if (!fp_in)
{
    printf("Can't open input file\n");
    return(-1);
}

while (feof(fp_in) == 0)
{
    // Grabs whole line up 79 characters or the end-of-file
    fgets(arr, 80, fp_in);

    // Writes out to output file as-is. Output file is the screen
    fprintf(stdout, "%s", arr);
}
return(0);
}
```

James Tam

Pointers

(Review)

- A variable that contains an address (rather than data)

Format (declaration):

*<type pointed to> * <name of pointer>;*

Example (declaration):

```
int * int_ptr;
```

James Tam

Pointer Operators

Star “*”

Used in the declaration of a pointer e.g., `char *ch_ptr;`

Used to dereference a pointer e.g., `ch = *ch_ptr;`

Ampersand “&”

Refers to the “address of” a variable

e.g., `ch_ptr = &ch;`

James Tam

First Pointer Example

The full example can be found in Unix:

`/home/profs/tamj/233/examples/c_examples/pointers/pointer1.c`

```
int num1, num2;
int *int_ptr;

num1 = 1;
num2 = 2;
printf("num1=%d \t num2=%d\n", num1, num2);

int_ptr = &num2;
num1 = *int_ptr;
printf("num1=%d \t num2=%d\n", num1, num2);
```

James Tam

Functions To Dynamically Allocate Memory

1. Malloc
2. Calloc

Make sure you include the standard library header:

```
#include <stdlib.h>
```

James Tam

Malloc

The amount of allocated memory is equal to the number of bytes specified by size.

The memory is **not** initialized

Format:

```
void * malloc (unsigned int size);
```

Example:

```
ch_ptr = malloc(SIZE);
```

Note:

- If the call to malloc was successful then it will return the base address of the dynamically allocated space.
- If the call was unsuccessful then NULL (0) is returned.

James Tam

Malloc: An Example

The full example can be found in Unix:

/home/profs/tamj/233/examples/c_examples/pointers/pointer2.c

```
int const SIZE = 8;
char *ch_ptr;
int i;
ch_ptr = malloc(SIZE);
if (ch_ptr != NULL) // Alternatively if (!ch_ptr)
{
    for (i = 0; i < SIZE; i++)
    {
        *ch_ptr = 'a' + i;
        printf("%c\n", *ch_ptr);
        ch_ptr++;
    }
}
```

James Tam

Calloc

- Allocates a continuous block of memory for an array of (size) elements with the size of each element being determined by the (element size) parameter
- Total memory allocated = (no of elements allocated x size of each element)
- The memory is initialized to 0.

Usage:

```
void * calloc (unsigned int no_elements, unsigned int element_size);
```

Example:

```
int_ptr = calloc (SIZE, sizeof(int));
```

Note:

- If the call to calloc was successful then it will return the base address of the dynamically allocated space.
- If the call was unsuccessful then NULL/0 is returned.

James Tam

Calloc: An Example

The full example can be found in Unix:
`/home/profs/tamj/233/examples/c_examples/pointers/pointer3.c`

```
int const SIZE = 8;
int *int_ptr;
int i;
int_ptr = calloc(SIZE, sizeof(int));
if (int_ptr != NULL)
{
    for (i = 0; i < SIZE; i++)
    {
        *int_ptr = 0 + i;
        printf("%d\n", *int_ptr);
        int_ptr++;
    }
}
```

James Tam

Cleaning Up Dynamically Allocated Memory

Unlike Java, C **does not** employ automatic garbage collection.

To free up dynamically allocated memory that you no longer need use “free”.

Format:

```
free (void * ptr);
```

Example:

```
free (ch_ptr);
```

Passing a NULL pointer to free will have no effect

Passing an invalid pointer to free will likely cause a system crash.

James Tam

Pointers To Pointers

A pointer that contains the address of a pointer (which contains the address of data variable)

```
int ** int_ptr2;
int *int_ptr;
int num = 1;
int_ptr = &num;
int_ptr2 = &int_ptr;
printf("%d %d %d", num, *int_ptr, **int_ptr2);
```

James Tam

Command Line Arguments

Inputs to the program as the program is run.

There is always at least one argument (e.g., a.out)

For example:

```
./a.out<SP>first<SP>second
```

James Tam

A Simple Program To Read Command Line Arguments (C Language Version)

The full example can be found in Unix:

`/home/profs/tamj/233/examples/c_examples/pointers/command_line.c`

```
int
main (int argc, char *argv[])
{
    int i;
    for (i = 0; i < argc; i++)
    {
        printf("argv[%d] = %s\n", i, argv[i]);
    }
    return(0);
}
```

James Tam

A Simple Program To Read Command Line Arguments (Java Version)

```
class Command_Line
{
    public static void main (String [] argv)
    {
        int i;
        for (i = 0; i < argv.length; i++)
        {
            System.out.println("argv[" + i + "]=" + argv[i]);
        }
    }
}
```

James Tam

Random Numbers

Apply a mathematical function

e.g., $f(n) = f(n-1) \% 2$

With $f(0) = 3$ the function yields 1, 1, 1...

The starting point for the function is called the seed

James Tam

Random Numbers In C

Need to include the standard library:

```
#include <stdlib.h>
```

Seeding the random number generator in C:

- `srand(time(NULL));`

Generating a random integer in C (calling the function):

- `int rand(void);`

James Tam

A Simple Random Number Generator

The full example can be found in Unix:

`/home/profs/tamj/233/examples/c_examples/random/generator.c`

```
#include <stdlib.h>
int const DIE_SIZE = 6;
int const NO_DICE = 3;
int roll_score ();
```

James Tam

A Simple Random Number Generator (2)

```
int
main ()
{
    int strength, dexterity, constitution;
    srand(time(NULL));
    strength = roll_score ();
    dexterity = roll_score ();
    constitution = roll_score ();
    printf("\nThe ability scores of your brave adventurer are...\n");
    printf("STR=%d DEX=%d CONST=%d\n\n", strength, dexterity,
        constitution);
    return(0);
}
```

James Tam

A Simple Random Number Generator (3)

```
int
roll_score ()
{
  int i, total, die_roll;
  total = 0;
  for (i = 0; i < NO_DICE; i++)
  {
    die_roll = (rand () % DIE_SIZE) + 1;
    total = total + die_roll;
  }
  return (total);
}
```

James Tam

Bitwise Operators

Operator	Action	Example
	Bitwise AND	x y
&	Bitwise OR	x & y
^	Bitwise XOR (Exclusive OR)	x ^ y
~	Bitwise NOT	~x
>>	Bitwise shift right (<i>variable >> no positions</i>)	x >> 1
<<	Bitwise shift left (<i>variable << no positions</i>)	x << 2

Apply the operation on each part of a variable
i.e., apply the truth tables for each bit

James Tam

Truth Table: AND

Truth table		
X	Y	X AND Y
False	False	False
False	True	False
True	False	False
True	True	True

Truth table		
X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

James Tam

Truth Table: OR

Truth table		
X	Y	X OR Y
False	False	False
False	True	True
True	False	True
True	True	True

Truth table		
X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

James Tam

Truth Table: NOT

Truth table	
X	Not X
False	True
True	False

Truth table	
X	Not X
0	1
1	0

James Tam

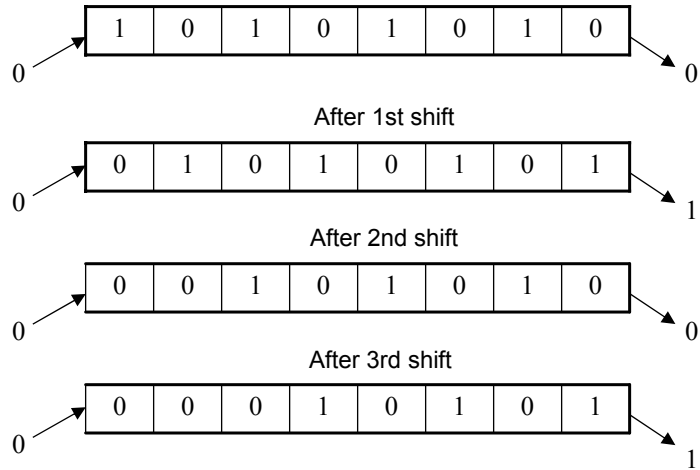
Truth Table: Exclusive-OR (XOR)

Truth table		
X	Y	X XOR Y
False	False	False
False	True	True
True	False	True
True	True	False

Truth table		
X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

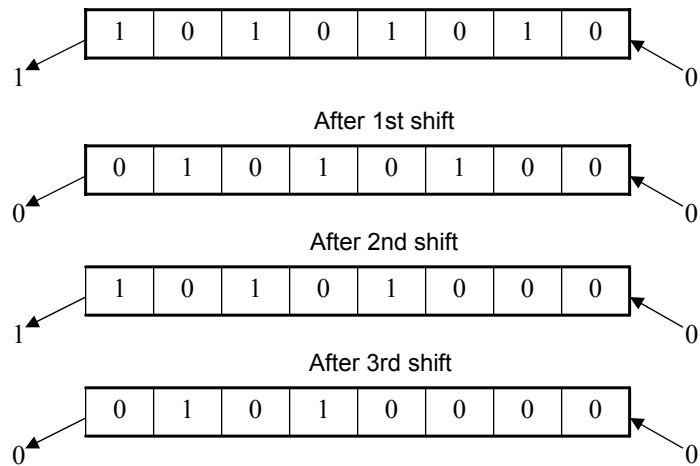
James Tam

Bitwise Shift: Right



James Tam

Bitwise Shift: Left



James Tam

Bitwise Operations: An Example

The full example can be found in Unix:

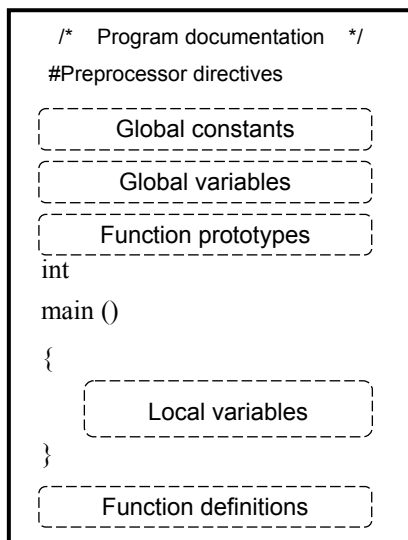
`/home/profs/tamj/233/examples/c_examples/low_level/bit_bashing.c`

```
char key = 58; /* Character = ':' ASCII = 58 */
char ch = 'u'; /* Character = 'u' ASCII = 117 */
char temp;

temp = ch & key;
temp = ch | key;
temp = ch ^ key;
temp = ~(-34);
temp = 32;
temp = temp << 1;
temp = (temp >> 3) - 1;
```

James Tam

Functions



James Tam

Function Prototypes And Function Definitions

Example:

```
int fun (int, int, int);
```

```
main ()
```

```
{  
:  
}
```

```
int
```

```
fun (int n1, int n2, int n3)
```

```
{  
    return(0);  
}
```

James Tam

Function Prototypes

Was not part of the original C language (pre C89)

Still not strictly needed by modern C (can be bypassed)

Can be used to check:

- The return type of a method
- The type of the parameters
- The number of parameters

Benefit:

- Can be used a design template (similar to Java interfaces)

James Tam

Function Prototypes (2)

Mandatory when two functions can call each other

```
void fun1 ();  
void fun2 ();
```

```
int  
main ()  
{  
    return (0);  
}
```

James Tam

Function Prototypes (3)

```
void  
fun1 ()  
{  
    fun2 ();  
}
```

```
void  
fun2 ()  
{  
    fun1 ();  
}
```

James Tam

Scope

A variable or constant comes into scope after it has been declared:

- If the variable or constant is global then it is accessible for the life of the program
- If the variable or constant is local then it is accessible until the end of the enclosing block (typically the end brace)

A function comes into scope:

- After the body of the function has been defined (don't use prototype)
- Or
- After a function prototype has been declared (use prototype)

James Tam

Scope: An Example Program “hideous.c”

The full example can be found in Unix:

`/home/profs/tamj/233/examples/c_examples/functions/hideous.c`

Note: This example program uses rather hideous code (but good conventions for naming the program) to illustrate how dangerous C can be for the undisciplined programmer.

James Tam

Scope: An Example Program “hideous.c” (2)

```
int num1 = 97;

void fun1 ();

int fun2 ()
{
    return(num1/2);
}

float fun3 (float);
```

James Tam

Scope: An Example Program “hideous.c” (3)

```
int
main ()
{
    char ch = 'A';
    printf("%c %d\n", ch, num1);
    fun1();
    printf("%c %d\n", ch, num1);
    num1 = fun2 ();
    printf("%c %d\n", ch, num1);
    num1 = fun3 (49.0);
    printf("%c %d\n", ch, num1);
    return(0);
}
```

James Tam

Scope: An Example Program “hideous.c” (4)

```
void
fun1 ()
{
    char ch = num1++;
    printf("%c %d\n", ch, num1);
    return;
}

float
fun3 (float f1)
{
    float f2 = 2.0;

    f1 = f2 * num1;
    return f1;
}
```

James Tam

Parameter Passing

All parameters (save arrays) are passed as value parameters
Pointers must be used to refer to the original parameters
(rather than the copies)

James Tam

Parameter Passing: An Example

The full example can be found in Unix:

`/home/profs/tamj/233/examples/c_examples/functions/parameters.c`

```
void no_swap (int, int);
void real_swap (int *, int *);

int
main ()
{
    int num1, num2;

    num1 = 1;
    num2 = 2;

    printf("num1=%d \t num2=%d \n", num1, num2);
```

James Tam

Parameter Passing: An Example (2)

```
no_swap(num1, num2);
printf("num1=%d \t num2=%d \n", num1, num2);
real_swap(&num1, &num2);
printf("num1=%d \t num2=%d \n", num1, num2);
return(0);
}

void
no_swap (int num1, int num2)
{
    int temp = num1;
```

James Tam

Parameter Passing: An Example (3)

```
void
real_swap (int *num1_ptr, int *num2_ptr)
{
    int temp = *num1_ptr;
    *num1_ptr = *num2_ptr;
    *num2_ptr = temp;
}
```

James Tam

Passing Arrays As Parameters

A pointer to the array is passed (you cannot pass by value)

A full example can be found in Unix:

/home/profs/tamj/233/examples/c_examples/functions/parameter2.c

```
#include <stdbool.h>
```

```
int const SIZE = 10;
```

```
// Prototypes indicate a pointer to an integer will be the parameters
```

```
void initialize (int *);
```

```
void display (int *);
```

James Tam

Passing Arrays As Parameters (2)

```
int
main ()
{
    int list[SIZE];
    display(list);
    initialize(list);
    display(list);
    return(0);
}
```

James Tam

Passing Arrays As Parameters (3)

```
// Passing an integer array of an unspecified length
void
initialize (int list[])
{
    int i;
    for (i = 0; i < SIZE; i++)
    {
        list[i] = i;
    }
}
```

James Tam

Passing Arrays As Parameters (4)

```
// Passing an array of length = SIZE
void
display (int list[SIZE])
{
    int i;
    for (i = 0; i < SIZE; i++)
    {
        printf("%d ", list[i]);
    }
    printf("\n");
}
```

James Tam

Summary

What you should now know about C programming

- What is the basic structure of a program
- How to create and compile programs of moderate size
- Basic data types and constants
- Decision making and loop constructs
- Common operators (mathematical, relational, logical, bitwise)
- 1D and 2D array
- Strings
- Simple file input and output
- Pointers and dynamic memory allocation

James Tam

Summary (2)

- Random number generators
- Functions:
 - Prototypes
 - Scope
 - Parameter passing