Overview

Today:
- More C
  - Basics
  - Debugging process
- Intro CGI
  - Background
  - Integrating c
- Some Shell Programming
Keep in touch

- Summer is short

- Make sure not to fall behind

- Here to help, so ask for it…won’t get more points for sleepless nights 😊
Reminder..

- A pointer is simply a memory location of some data
- Your job to use that memory spot appropriately
- Compiler needs to know size of memory spot in order to move pointers correctly (increment)
- Ampersand & is used to get the address of a variable location (dereference a pointer)
Arrays as pointers

- an array is a pre-allocated contiguous memory locations
- an array definition is really a pointer to the starting memory location of the array
- Remember that pointers are really integers in a sense
  - so you can perform integer arithmetic on them
  - e.g., +1 increments a pointer, -1 decrements
  - you can use this to move from one array element to another
String

- char s[6] = “ABCDE”;
- Memory storage looks like:

  A | B | C | D | E | \0

- Need to remember that you are really accessing indices 0 – (length-2) since the value at length-1 is always \0

- Need to allocate enough memory
  - +1 for many things
  - Remember to comment, or next programmer will kill program
Using strings

- printing strings
- format sequence: %s
- example:

```c
#include <stdio.h>
int main() {
  char *str0;
  char str1[6] = "ABCDE";
  str0 = (char*)malloc(sizeof(char) * 12);
  Strcpy(str0,"Hello");
  printf("str0 = %s and str1 = %s\n", str0, str1);
}
/* end of main() */
```
Useful functions

- Many useful functions

- strcat

- strcpy

- DON’T MIX THEM UP 😊
Header files

- .h files usually used to define methods or centralize definitions

- public int calculateSomething(int []);

- Can either name the variables or not
  - int[] vs int ar[]
  - In .c file use; #include “something.h”
Arrays and pointers are strongly related in C

```c
int a[10];
int *pa;
```

(remember that &a[0] is the address of the first element in a, that is the beginning of the array)

```c
pa = &a[0];
pa = a;
```

pointer arithmetic is meaningful with arrays:

```c
Pntr = &a[0]
```

then

```c
*(Pntr +1) =
```

Is whatever is at a[1]

```c
Is whatever is at a[1]
```
Array of pointers

```c
int *p[4];
```

- What is this

- Why ?
There is a difference between
- *(Pntr) + 1
- and (*Pntr +1)

Note that an array name is a pointer, so we can also do *(a+1) and in general: *(a + i) == a[i] and so are a + i == &a[i]

The difference:
- an array name is a constant, and a pointer is not
- so we can do: Pntr = a and Pntr ++

But we can NOT do: a = Pntr or a++ pr or Pntr = &a

That is you can not reassign it as a pointer
Note

- When an array name is passed to a function, what is passed is the beginning of the array, that is passed by reference.

- It is important, since this is an address, any changes to that memory location will stick when you come back from the function.
From last time

- a pointer contains the address of an object (but not in the OOP sense)
- allows one to access object “indirectly”
- \& = unary operator that gives address of its argument
- \* = unary operator that fetches contents of its argument (i.e., its argument is an address)
- note that \& and \* bind more tightly than arithmetic operators
- you can print the value of a pointer with the formatting character \%p
```c
#include <stdio.h>
main() {
    int x, y;     // declare two ints
    int *px;      // declare a pointer to an int
    x = 3;        // initialize x
    px = &x;
    y = *px;
    printf( "x=%d px=%p y=%d\n", x, px, y );
}
```
Example 2

```c
int main(void) {
    char *string1 = (char*)malloc(sizeof(char)*50);
    char *string2 = (char*)malloc(sizeof(char)*50);
    scanf("%s", string2);
    string1 = string2;  // MISTAKE THIS IS NOT A COPY

    ...
    free(string2);
    free(string1);  // ???

    return 0
}
```
Memory leak tools

- Purify
- Valgrind
- Insure++
- Memwatch (will use it next week)
- Memtrace
- Dmalloc
Dynamically allocated arrays

- C references arrays by the address of their first element
- array is equivalent to &array[0]
- you can iterate through arrays using pointers as well as indexes:

```c
int *v, *last;
int sum = 0;
last = &x[length_x-1];
for ( v = x; v <= last; v++ )
    sum += *v;
```
2 dimensional arrays

- 2-dimensional arrays
- `int weekends[52][2];`
- you can use indices or pointer math to locate elements in the array
  - `weekends[0][1]`
  - `weekends+1`
- `weekends[2][1]` is same as
  - `*(weekends+2*2+1)`*, but NOT the same as
  - `*weekends+2*2+1` *(which is an integer)!*
Be aware

- 2 dimension arrays different than how java does it

- Can create the same using arrays of pointers....
File Handling – short intro

- `File *log_file;`

- any ideas what this look like?
- use function fopen to open handle
- pass in arguments to fopen to set type
  - r  read
  - w  write
  - a  append
- need to check if not null
if( (log_file = fopen("some.txt", "w")) == NULL) 
    fprintf(stderr,"Cannot open %s\n", "log_file");

/*****
do your cool stuff here

*****/

fclose(log_file);
moving characters

- can move characters using putchar(c) and getchar()

- if no handle supplied
- putchar(c, stdout)
- getchar(stdin)
problem

- Can you think of using getchar for user input?

- How to deal with unknown input length..
strings

- fgets
- fputs
void swapNot( int a, int b ) {
    int tmp = a;
    a = b;
    b = tmp;
} // end swapNot()

void swap( int *a, int *b ) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
} // end swap()
swap

```c
int x, y; // declare two ints
int *px, *py; // declare two pointers to ints
x = 3; // initialize x
y = 5; // initialize y

printf( "before: x=%d y=%d\n", x, y );

swapNot( x, y );
printf( "after swapNot: x=%d y=%d\n", x, y );

px = &x; // set px to point to x (i.e., x's address)
py = &y; // set py to point to y (i.e., y's address)

printf( "the pointers: px=%p py=%p\n", px, py );

swap( px, py );
printf( "after swap with pointers: x=%d y=%d px=%p py=%p\n", x, y, px, py );

// you can also do this directly, without px and py:
swap( &x, &y );
printf( "after swap without pointers: x=%d y=%d\n", x, y );
```
int main()
{
    int number = 10;
    foo(&number);
    return 0;
}

void foo(int *p){
    *p = 30;
}
Question

- What's the advantage of passing in by pointer reference?

- What is the problem?

- How would we solve it?
const

- Allows the compiler to know which values shouldn’t be modified
- Added in to C later

Example:
```c
const int a = 5;

void foo(const int x) {
}
```
**const**

- Better than `#define` since error message will be easier to understand since preprocessor not involved.
- Very useful in functions to either return `const` or make sure a pointer doesn’t alter the original object.
Const pointer to non-const

- This is a pointer which always points to the same location, but the value can be modified.

```c
int * const ptr = &x;

*ptr = ??
  can’t say
ptr = & ??
```

- Example2: array name
Const pointer to const data

- Int x = 200;
- const int * const ptr = &x;
Some confusion

- `int const * X`
- `const int * X //variable pointer to const`
- `int * const Y //const pointer to int`
- `int const * const Z //const point to const`
int main( int argc, char *argv[] )

- argc is the argument count
- argv is the argument vector
  - array of strings with command-line arguments
- the int value is the return value
  - convention: return value of 0 means success,
  - > 0 means there was some kind of error
  - can also declare as void (no return value)
- Name of executable followed by space-separated arguments
  
  $ a.out 1 23 "third arg"

- this is stored like this:
  1. a.out
  2. 1
  3. 23
  4. “third arg”

- argc = 4
If no arguments, simplify:
```c
int main() {
    printf( "hello world" );
    exit( 0 );
}
```

Uses `exit()` instead of `return()` — almost the same thing.
booleans

- C doesn’t have booleans
- emulate as int or char, with values 0 (false) and 1 or non-zero (true)

- allowed by flow control statements:
  ```c
  if ( n == 0 ) {
  printf( "something wrong" );
  }
  ```
- assignment returns zero -> false
- you can define your own boolean using preprocessor directives:
  ```c
  #define FALSE 0
  #define TRUE 1
  ```
Booleans II

- This works in general, but beware:
  
  ```c
  if ( n == TRUE ) {
    printf( "everything is a-okay" );
  }
  ```

- If `n` is greater than zero, it will be non-zero, but may not be 1; so the above is NOT the same as:
  
  ```c
  if ( n ) {
    printf( "something is rotten in the state of denmark" );
  }
  ```
Logical operators

- in C logical operators are the same as in Java
- meaning  C operator
  - AND     
  - OR      
  - NOT
- since there are no boolean types in C, these are mainly used to connect clauses in if and while statements
- remember that
  - non-zero == true
  - zero      == false
Bitwise operators

- there are also bitwise operators in C, in which each bit is an operand:
  - bitwise AND &
  - bitwise or |

Example:
```c
int a = 8; /* this is 1000 in base 2 */
int b = 15; /* this is 1111 in base 2 */
a & b =
```

```
  1000 (8)
  1111 (15)
  1000 (= 8)
```

```
a | b =
```

```
  1000 (8)
  1111 (15)
  1111 (= 15)
```
Code sample

- Print out the output of the following code fragment?

```c
int a = 12, b = 7;
printf( "a && b = %d\n", a && b );
printf( "a || b = %d\n", a || b );
printf( "a & b = %d\n", a & b );
printf( "a | b = %d\n", a | b );
```
Implicit conversions

- implicit:
  ```
  int a = 1;
  char b = 97; // converts int to char
  int s = a + b; // adds int and char, converts to int
  ```

- promotion: char -> short -> int -> float -> double
  - if one operand is double, the other is made double
  - else if either is float, the other is made float

  ```
  int a = 3;
  float x = 97.6;
  double y = 145.987;
  y = x * y; // x becomes double; result is double
  x = x + a; // a becomes float; result is float
  ```

- real (float or double) to int truncates
explicit

- explicit:
- type casting
  ```
  int a = 3;
  float x = 97.6;
  double y = 145.987;
  y = (double)x * y;
  x = x + (float)a;
  ```
- – using functions (in math library...)
  1. floor() – rounds to largest integer not greater than x
  2. ceil() - round to smallest integer not smaller than x
  3. round() – rounds up from halfway integer values
Example

```c
#include <stdio.h>
#include <math.h>
int main() {
    int j, i, x;
    double f = 12.00;
    for ( j=0; j<10; j++ ) {
        i = f;
        x = (int)f;
        printf( "f=%.2f i=%d x=%d
floor(f)=%.2f ceil(f)=%.2f round(f)=%.2f\n",
            f,i,x,floor(f),ceil(f),round(f) );
        f += 0.10;
    } // end for j
} // end main()
```
Output

- f=12.00  i=12  x=12  floor(f)=12.00  ceil(f)=12.00  round(f)=12.00
- f=12.10  i=12  x=12  floor(f)=12.00  ceil(f)=13.00  round(f)=12.00
- f=12.20  i=12  x=12  floor(f)=12.00  ceil(f)=13.00  round(f)=12.00
- f=12.30  i=12  x=12  floor(f)=12.00  ceil(f)=13.00  round(f)=12.00
- f=12.40  i=12  x=12  floor(f)=12.00  ceil(f)=13.00  round(f)=12.00
- f=12.50  i=12  x=12  floor(f)=12.00  ceil(f)=13.00  round(f)=12.00
- f=12.60  i=12  x=12  floor(f)=12.00  ceil(f)=13.00  round(f)=13.00
- f=12.70  i=12  x=12  floor(f)=12.00  ceil(f)=13.00  round(f)=13.00
- f=12.80  i=12  x=12  floor(f)=12.00  ceil(f)=13.00  round(f)=13.00
- f=12.90  i=12  x=12  floor(f)=12.00  ceil(f)=13.00  round(f)=13.00
Be aware

- almost any conversion does something— but not necessarily what you intended!!
- example:
  ```c
  int x = 100000;
  short s = x;
  printf("%d %d\n", x, s);
  ```
- output is:
  100000  -31072
- WHY?
Functions ceil() and floor() come from the math library
definitions:
- ceil( x ): returns the smallest integer not less than x, as a double
- floor( x ): returns the largest integer not greater than x, as a double

in order to use these functions, you need to do two things:
1. include the prototypes (i.e., function definitions) in the source code:  
   #include <math.h>
2. include the library (i.e., functions’ object code) at link time: 
   unix$ gcc abcd.c -lm

exercise: can you write a program that rounds a floating point?
math

- some other functions from the math library (these are function prototypes):
  - double sqrt( double x );
  - double pow( double x, double y );
  - double exp( double x );
  - double log( double x );
  - double sin( double x );
  - double cos( double x );

- exercise: write a program that calls each of these functions

- questions:
  - can you make sense of /usr/include/math.h?
  - where are the definitions of the above functions?
  - what are other math library functions?
Random numbers

- with computers, nothing is random (even though it may seem so at times...)

- there are two steps to using random numbers in C:
  1. seeding the random number generator
  2. generating random number(s)

- standard library function:
  #include <stdlib.h>

- seed function:
  srand( time ( NULL ));

- random number function returns a number between 0 and RAND_MAX (which is $2^{32}$)
  int i = rand();
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main( void ) {
    int r;
    srand( time ( NULL ) );
    r = rand() % 100;
    printf( "pick a number between 0 and 100...
" );
    printf( "was %d your number?", r );
}
```
Character handling

- character handling library
  
  `#include <ctype.h>`

- digit recognition functions (bases 10 and 16)
- alphanumeric character recognition
- case recognition/conversion
- character type recognition

- these are all of the form:
  
  `int isdigit(int c);`

- where the argument `c` is declared as an int, but it is interpreted as a char

- so if `c = '0'` (i.e., the ASCII value '0', index=48), then the function returns true (non-zero int)
  but if `c = 0` (i.e., the ASCII value NULL, index=0), then the function returns false (0)
digits

- digit recognition functions (bases 10 and 16)
  ```c
  int isdigit( int c );
  ```
  returns true (i.e., non-zero int) if c is a decimal digit (i.e., in the range ’0’..’9’);
  returns 0 otherwise

- ```c
  int isxdigit( int c );
  ```
  returns true (i.e., non-zero int) if c is a hexadecimal digit (i.e., in the range ’0’..’9’,’A’..’F’);
  returns 0 otherwise
Alpha numeric

- alphanumerical character recognition
  
  int isalpha( int c );
  
  returns true (i.e., non-zero int) if c is a letter (i.e., in the range ’A’..’Z’, ’a’..’z’); returns 0 otherwise

int isalnum( int c );

- returns true (i.e., non-zero int) if c is an alphanumerical character (i.e., in the range ’A’..’Z’, ’a’..’z’, ’0’..’9’); returns 0 otherwise
Case

- **Case recognition**
  
  ```
  int islower( int c );
  ```
  
  int islower( int c );
  returns true (i.e., non-zero int) if c is a lowercase letter (i.e., in the range ‘a’..'z');
  returns 0 otherwise

  ```
  int isupper( int c );
  ```
  
  int isupper( int c );
  returns true (i.e., non-zero int) if c is an uppercase letter (i.e., in the range ‘A’..'Z');
  returns 0 otherwise

- **Case conversion**

  ```
  int tolower( int c );
  ```
  
  int tolower( int c );
  returns the value of c converted to a lowercase letter (does nothing if c is not a letter or if c is already lowercase)

  ```
  int toupper( int c );
  ```
  
  int toupper( int c );
  returns the value of c converted to an uppercase letter (does nothing if c is not a letter or if c is already uppercase)
types

- character type recognition
  int isspace( int c );
  returns true (i.e., non-zero int) if c is a space; returns 0 otherwise
int iscntrl( int c );
  returns true (i.e., non-zero int) if c is a control character; returns 0 otherwise
int ispunct( int c );
  returns true (i.e., non-zero int) if c is a punctuation mark; returns 0 otherwise
int isprint( int c );
  returns true (i.e., non-zero int) if c is a printable character; returns 0 otherwise
int isgraph( int c );
  returns true (i.e., non-zero int) if c is a graphics character; returns 0 otherwise
Next up…

- What is the internet?

- Technical overview
  - Servers - serve http request
  - Clients - browsers issue requests
Boring vs. Exciting

- Typical
  - Request is served from a file formatted in html
  - Static file of what we would like to render on a web client.
  - Example:
    - Class syllabus

- What is we could tailor each users web experience to what they want.
  - Design of protocol to handle this
How does CGI work:

1. HTTP Request
2. Call CGI
3. CGI Responds
4. HTTP Response
C + cgi

- Remember:
  - C is only a tool here
  - Don’t memorize, understand
    - Why
    - What
    - How
  - Don’t be afraid to experiment
- STDIN
  - Contents passed to your C program
- STDOUT
  - Will need HTTP headers before printing
- STDERR
  - Depends on server, sometimes just error logs, sometimes error reports on client
ENV

- This is your best friend in CGI
- Way of getting information *from* the client
- Create content is way to pass back information *to* the client
Remember

- Need to set permissions:
  - `chmod 0755 ???.cgi`
  - `-rwxr-xr-x`

- Need to place script in correct place
  - `sometimes cgi-bin/ directory`

- Naming
  - `Some web servers require the C cgi program to end in .cgi`
Sample test4.cgi

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <time.h>

int main() {
    time_t t1,t2;
    (void)time(&t1);

    printf( "Content-type: text/plain\n\n" );
    printf("this is the time is %s",ctime(&t1));
    printf( "You IP is = [%s]\n" , getenv( "REMOTE_ADDR" ));

} // end of main()
This is the time: Tue Jul 19 16:45:17 2005

and your name is 24.188.170.41
Some CGI Environmental Variables

- CONTENT_LENGTH
  - Length of data passed to cgi
- CONTENT_TYPE
- QUERY_STRING
- REMOTE_ADDR
  - Ip address of client
- REQUEST_METHOD
- SCRIPT_NAME
- SERVER_PORT
- SERVER_NAME
- SERVER_SOFTWARE
- HTTP_FROM
- HTTP_USER_AGENT
- HTTP_REFERER
- HTTP_ACCEPT
HTML

- Hyper Text Markup Language
- Standard by w3: http://www.w3.org/MarkUp/
- Way of standardizing format of documents so that users can share information between different systems seamlessly
- Evolving to XHTML format
HTML

- Hypertext Transfer Protocol
- Language used between web servers and web clients
- http url’s

```
http://www.google.com:80/search?q=shlomo
```
Google.com

http://www.google.com/search?q=shlomo
Very basics

- Html consists of matching tags
- <something> = opening tag
- </something> = close tags

HTML DOC:
- <html> <body> …… </body> </html>
Web pages

- `<title> .... </title>` (before the body section)
- `<H1> .... </H1>` (header titles h1, h2, h3)
- `<P>` paragraphs
- `<BR>` line breaks
- `<b>` ... `</b>` bold
- `<i>` ... `</i>` italicize
- `<u>` ... `</u>` underline
More basics

- `<img src="....." width="X" height="Y">`
- `<a href="www.cnn.com"> something </a>`
- `<a name="Anchor1">`  
  □ Can be referred to by page.html#Anchor1
- `<hr>  line`
- `<hr width=50%>  half line`
Lists

- Unordered list
  `<ul> <li> </li> …… </ul>`

- Ordered list
  `<ol> <li> </li> ….. </ol>`

- Nested lists
  - Lists themselves can be nested within another
Tables

- `<table>
  <tr>
    <td>Hello</td>
    <td>World</td>
  </tr>
</table>`
comments

<!--

anything you do

-->
More html

- Can get wysiwyg editors
- Word will allow you to save as html
- Can take a look at webpages source code
Browser Issues

- Although HTML should be universal, there are occasional differences between how Microsoft IE renders a webpage and Mozilla firefox
Task

How would we?

- Create a webpage counter (saying you are visitor x to this page)

- Now create a graphical counter
MD5 Sum

- MD5 – uses a 128 bit hash value
- Designed in 1991
- Known problems with collision attacks

Bottom line

- Still in very wide use
- Allows authentication of files given a file and signature
- Visually authentication against tampering
- What obvious weakness??
Md5 of a file

- If we have a bunch of data which we want to get an md5 of…
  - Write yourself
    - Learn tons of math first
    - Make up errors 😊 as you program..
  - Find someone else’s library 😊
Digests

- The 128-bit (16-byte) MD5 hashes (also termed message digests) are typically represented as 32-digit hexadecimal numbers.

- Even small change can result in a totally different hash digest
Digests II

- MD5("The quick brown fox jumps over the lazy dog") = 9e107d9d372bb6826bd81d3542a419d6
- MD5("The quick brown fox jumps over the lazy cog") = 1055d3e698d289f2af8663725127bd4b
- MD5(""") = d41d8cd98f00b204e9800998ecf8427e
Computer Security

- System and theory of ensuring the confidentiality, integrity, availability, and control of electronic information and systems.
  - Network
  - Host
  - Data
For host based security

- Want to ensure permission system
  - X should only be allowed to do A, B, and C
- Want to ensure accountability
  - If Y does something not allowed, should be noted
- Want to be able to track
  - If something has been tampered with, how can we locate it
  - Both preventative and reactionary
Forms

- One way to get information is to collect data
  - Registration
  - Payment
  - Surveys

- Commands
  - Possible choice combination
  - Actions

- Generally user needs to hit submit for anything to happen
Example

- Google.com

- Load page
- Do nothing…nothing happens
- Type search…nothing happens
  - Hit submit/return trigger action
Other way

- React to user typing (will not be doing this)
2 ways to do it

1. Create a HTML file and display a form, and your script gets input from the form

2. Have your script run
   1. If no information is being passed, print out the html for a form (then end)
   2. Else process the form information in the script
Interacting

- **GET**
  - HTTP request directly to the cgi script by appending the URL

- **POST**
  - HTTP request in content of message, i.e. it is stdin to your script

- **Format of GET (default):**
  - Value=key separated by &
  - Space replaced by +
  - URL conversion characters
Input Tag

- Each field is in an input tag
- Type
  - Text
  - Radio button
  - Checkbox
  - Pull down menus
  - etc
- Name
  - Symbolic name (so can recognize it)
- Value
  - Default value, or what the user will end up typing
Encoding

- Spaces are turned to +
- & separates field
- Special characters are turned into %?? (hex)
  - “(“ is %28

- So “class is great” = “class+is+great”
others

- Submit buttons
  - `<input type=“submit”>`
- Reset buttons
  - `<input type=“reset”>`

- Value will change the default name on the button
Putting it all together

<form action="cgi/some.cgi" method="GET">
  <p>Please enter some text:
    <input type="text" name="string"></p>
  <input type="submit">
</form>
Please enter some text: 

Submit Query
Decoding Form Input

1. Getenv("QUERY_STRING")
2. if( strcmp(getenv("REQUEST_METHOD" ,
    "POST"))
   { //check getenv("CONTENT_LENGTH")
3. Split pairs around &
4. Split keys and values
5. Decode URL
6. Remember key,values
Drawback

- A lot of work
- Pain if we have multiple values associated with one key
- Must be easier way.....
- There are cgi libraries...
The bad news

- Can’t use it in this class
- Want you to practice doing it the manual way…better for learning and later integrating CGI + C/CPP
Summary: CGI

- Minimum the web server needs to provide to allow an external process to create WebPages.

- Goal: responding to queries and presenting dynamic content via HTTP.
Requirements

- Webserver setup correctly
  - We will practice with Abyss

- Configure the cgi script
  - Will cover this now.

- Basic http/html knowledge
scripts

- Need to configure the web server to be able to run scripts

- Host configure->scripting parameters->script paths...

- /cgi-bin/*.exe
CGI Environment

- In C available through the ENV global hash access with getenv
- Changing any of the values will only be seen by your own subprocess
  - Why?
- Some of the variables will be blank
  - Why?
  - Example when run the cgi in command shell
File handling

- We covered basic file handling

- How does this change over the web?
Serving more than webpages

printf( "Content-type: text/html\n\n");

print ("Content-type: image/jpeg\n\n");
print ("Content-type: image/png\n\n");
print ("Content-type: audio/mp3\n\n");
Example

- http://..../cgi-bin/mp3server.cgi/Song.mp3
Argument passing

- Say you have a cool program which you can hook to the web…..
  - Give a cell phone
  - Give a message
  - Will send the cell phone a message
<HTML><HEAD><TITLE>Cool</TITLE></HEAD><BODY>
<form action="cgi-bin/cool.cgi" method="GET">
<p>Enter cell phone to use:<br>
<input type="text" name="cellphone"></p>
<p>Enter Message:<br>
<input type="text" name="message"></p>
<input type="submit">
</form>
</BODY></HTML>
Idea:

Want to run the program on client side from information collected on the web...
What can go wrong?
When executing command can in theory pass in the following arguments

Something ; rm –rf *.*