Miscellany and Practical Examples

Returning a Pointer From a Function

```
char * dup1(const char *
    str){
    char * p;
    int n,k;
    for(n=0;str[n]!='\0';n++)
            ;
    p = (char*) malloc(2*n+1);
    for(k=0;k<n;k++){
       p[2*k] = str[k];
       p[2*k+1] = str[k];
    }
    return p;
}
```

```
int main(int argc, char *
    argv[]){
    char *p;
    if(argc != 2){
      printf("Usage: %s str\n",
    argv[0]);
       return 0;
    }
    p = dup1(argv[1]);
    puts(p);
    free(p);
}
```

A Wrong Version

```
char * dup2(const char *
   str){
   char p[1000];
   int n,k;
   for(n=0;str[n]!='\0';n++)
         ;

   for(k=0;k<n;k++){
      p[2*k] = str[k];
      p[2*k+1] = str[k];
   }
   return p; /* WRONG!!! */
}
```

- Anything that comes automatically also goes automatically.
- A calling function cannot use the returned value to access the 1000-char array anymore.
- In this case, the array p disappears off of the stack as soon as dup2() finishes, and is gone forever!

A Different Wrong Version

```
char * dup3(const char *
    str, char *dest){
    int n,k;
    for(n=0;str[n]!='\0';n++)
            ;
    for(k=0;k<n;k++){
       dest[2*k] = str[k];
       dest[2*k+1] = str[k];
    }
    return dest;
}
```

```
int main(int argc, char *
    argv[]){
    char *p, buf[1000];
    if(argc != 2){
      printf("Usage: %s str\n",
    argv[0]);
       return 0;
    }
    p = dup3(argv[1], buf);
    puts(p);
    free(p);  /* WRONG!!! */
}
```

Error Handling Functions (1)

- C has very few debugging facilities.

```
#include <assert.h>
void assert (int expression) ;
```

- Expression is anything you want to test.
  - If true, assert does nothing.
  - If false, assert displays an error message on stderr and terminates immediately with a core file.
- Example:

```
assert (interest_rate >=0) ;
```

  - If interest_rate<0 then a relatively informative error message is printed and the program terminates abnormally.

Error Handling Functions (2)

- Many standard C functions set an internal variable called errno when they fail to reflect the cause of the failure.

```
#include <errno.h>
int errno;
void perror(const char *s);
```

- For example:

```
if ((fp = fopen("testfile", "r")) == NULL) {
  status = errno;   /* Must save before it is reset!!! */
  perror("fopen() of testfile failed");
  fprintf(stderr, "errno was %d!\n", status);
}
```

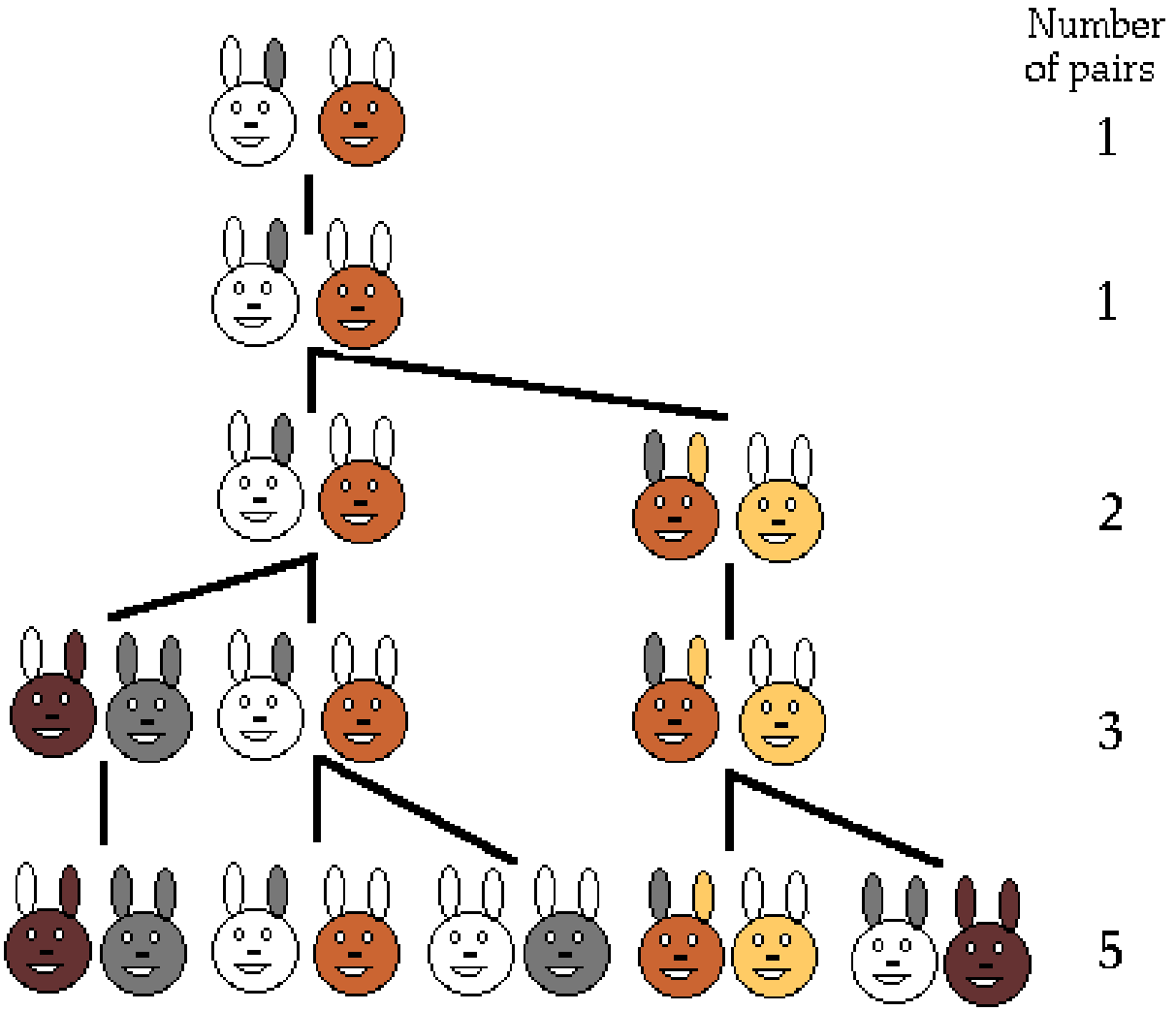
Fibonacci

- We use the following question as an example to demonstrate how to use recursion to solve a problem.
- Fibonacci Numbers

  1, 1, 2, 3, 5, 8, 13, ... (add the last two to get the next)
- An interesting fact about Fibonacci numbers is that the division between two consecutive numbers approach the Golden Number

  (sqrt(5)-1)/2 ~ 0.6180339890

The Original Example of Fibonacii Numbers



- Find more interesting examples at
  http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fib.html

Recursion for Fibonacci Numbers (1)

- Fib(1)=1;
- Fib(2)=1;
- Fib(n)=Fib(n-1)+Fib(n-2);

```
/* fib.c */
int Fib(int n){
    if(n<=2) return 1;
    else return Fib(n-1)+Fib(n-2);
}

/* fib.h */
int Fib(int n);
```

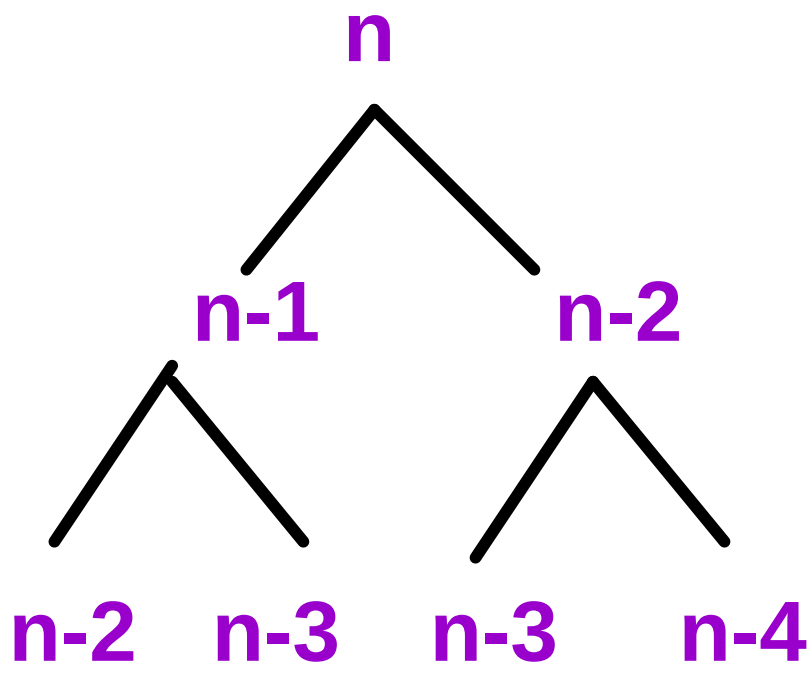
Recursion for Fibonacci Numbers (2)

```
* main.c */
#include <stdio.h>
#include "fib.h"
nt main(int argc, char * argv[]){
  if(argc!=2){
    printf("Usage: %s nnn\n", argv[0]);
    return 0;
  }
  int n = atoi(argv[1]);
  printf("Fib(%d)=%d\n", n, Fib(n));
  return 0;
```

```
# Makefile
fib: main.o fib.o
    gcc main.o fib.o -o fib
main.o: main.c fib.h
    gcc -c main.c
fib.o: fib.c fib.h
    gcc -c fib.c
clean:
    rm -f fib *.o core
```

More Efficient Programming

- The previous program is simple, but it is very inefficient!
- Programming is not only coding, but also about the design of data structures and algorithms.
- The time needed by the previous Fib(n) function call is Fib(n) time units.



Time(1)=Time(2)=constant;
Time(n)=Time(n-1)+Time(n-2

Another Method to Compute Fib

```
Fib(int n){
    int * fib = (int *) malloc(n
       * sizeof (int) );
    int i;
    fib[0]=fib[1]=1;
    for(i=2;i<n;i++){
        fib[i]=fib[i-1] + fib[i-2];
    }
    i = fib[n-1];
    free(fib);
    return i;
}
```

- This only needs to loop the for loop n times.
- This type of method/algorithm is called *Dynamic Programming*. You'll see more about this in future computer science courses!

Using Strings to Represent Large Integers

- This is an exercise:

- Using a string to represent a series of digits, e.g.

  "123456789012345678 90"

  \+ "111"

  ="123456789012345680 01"

- Then modify the previous Fib() function and main() function to compute Fib(211), which should be:

558350732953004655366280865857866723572343 89