Strings and Things

CIS 15 : Spring 2007

Functionalia

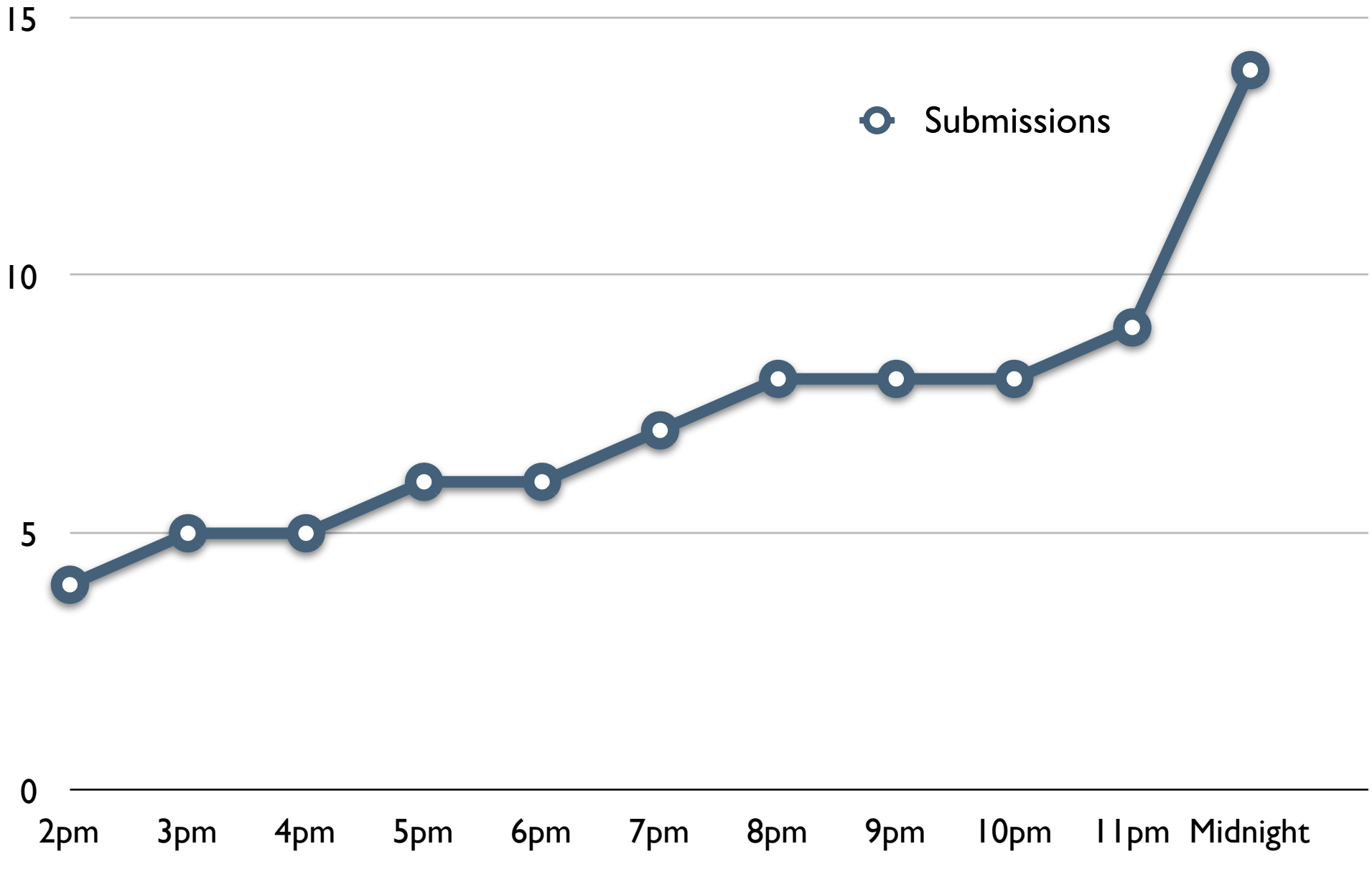
TEA Tomorrow 1:30-3:30 - 0317 N

UBUNTU Linux (?) - Anybody give it a try?

Today:

- Tail Recursion
- Strings and String Libraries

HW 1 Response



Consider this code...

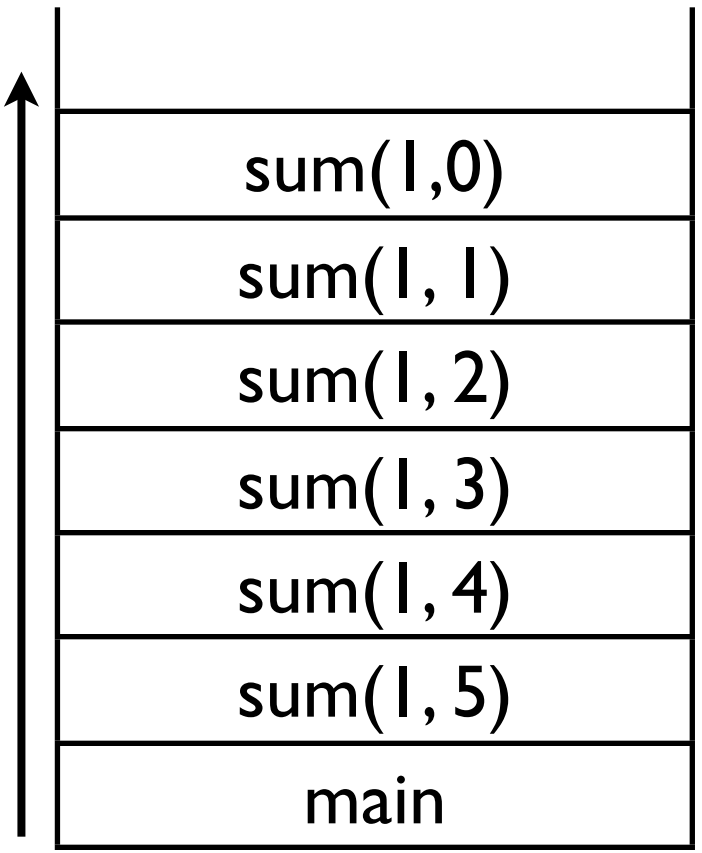
```
int sum(int val,int count)
{
    int s;
    if (count <= 0) {
        // nothing left
        s = 0;
    }
    else {
        // add one instance of 'val' to the result
        // of calling sum1 with one less count
        s = val + sum(val,count-1);
    }
    return s;
}


int main()
{
    cout << sum(1, 5);

    return 0;
}
```

Let's look at the stack...

Return Address needs to be saved on the Stack.

Which makes for a tall stack.

We can optimize by making the recursive call the **last thing** done by the sum(...) function. (That way...)



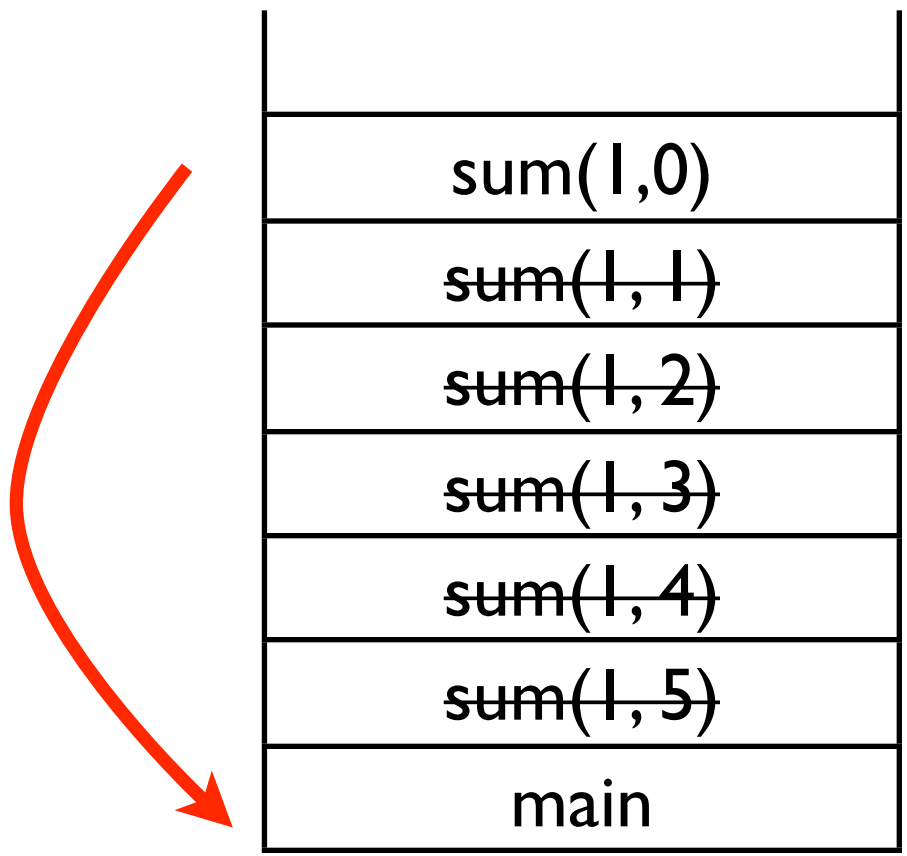
Let's look at the stack...

sum(1,0) can return directly back to main() !

Having the recursive call at the end is called being a ***tail call***.

Tail Call

```
int some_func(....)
{
     ...do stuff
     ...do stuff
     ...do stuff
     some_func(...)
}
```



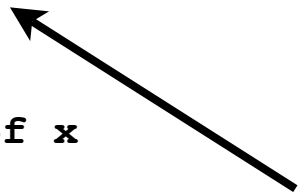
Tail Recursion

- A recursive program is **tail recursive** if all recursive calls are **tail calls**
- Tail recursive programs may be optimized (through *compiler optimization*) to be implemented as loops, thus removing the function call overhead for the recursive calls
- Tail recursion generally requires extra "accumulator" arguments to pass partial results (the intermediate sum values).
    - May require an auxiliary function

Conversion to Tail Recursion

Split your Recursive Function into a **stub** and a **cons** function.

```
int sum(int val, int count)
{
    // set up an accumulator 'x' which will hold the
    // intermediate results
    int x = 0;

    // call the stub, passing it the original parameters
    // plus a reference to the accumulator
    sum_cons(val,count, x);

    // return the value of x
    return x;
}
```



Using a Reference to X

Cons Function

```
int sum_cons(int val, int count, int &accumulator)
{
    // quit if no counts left
    // important to quit here. not allowed to quit at the end
    if (count <= 0) return 0;

    // sum into the accumlator.
    accumulator += val;

    // call back into itself.
    // now we have tail recursion. nothing happens after
    // the recursive call. the recursion will still
    // end when the count argument goes to 0
    sum_cons(val, count-1, accumulator);
}
```

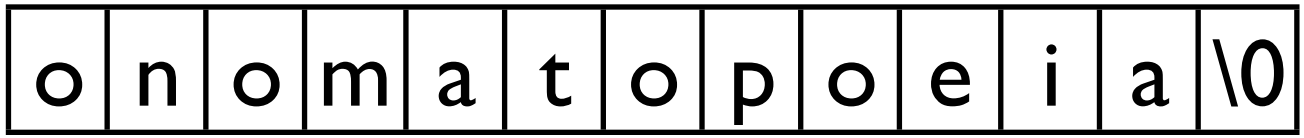
Will get an compiler error complaining about not all control paths returning a value.

Basic Pattern

```
void original(args...)
{
    declare an accumulator

    call cons_func passing original args + the accumulator

    return value of accumulator
}
```

```
void cons_func(args..., accumlator)
{
    check termination condition

        construct one element into the accumulator

    call self recursively
}
```

Enough Recursion... on to Strings



Working with Strings on your own can be tedious!

C++ String Library - **cstring**

```
#include <cstring>
```

(**NOT** `#include <string.h>` `// which will work, but let's not rely on it`)

Based on the string.h library - for detailed info: *man string*

Let's see some of the popular String Library functions...

`int strlen(char *);`

String **Len**gth - Counts the number of characters in the string (not including the \0 byte).

It does NOT know the length of the array (i.e. contiguous memory) containing the string.

Counts chars to the next \0 byte;

```
char onomatopoeia[50] = "pow!";

int length;

length = strlen(onomatopoeia);
```

Length is?

`int strlen(char *);`

String **Len**gth - Counts the number of characters in the string (not including the \0 byte).

It does NOT know the length of the array (i.e. contiguous memory) containing the string.

Counts chars to the next \0 byte;

```
char onomatopoeia[50] = "pow!";

int length;

length = strlen(onomatopoeia);
```

Length is? 4

`char * strcat(char *, char *);`

String Con**cat**enate - Appends one string onto another.

```
char abba[50] = "zip! ";

char cadabba[50] = "zop!";

cout << abba << endl;

cout << cadabba << endl;


strcat(abba, cadabba);

cout << abba << endl;

...

zip!

zop!

zip! zop!
```

`char * strcat(char *, char *);`

String Con**cat**enate - Appends one string onto another.

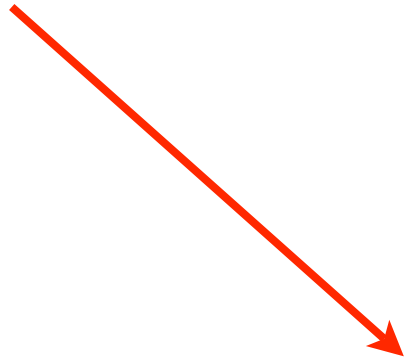
```
char abba[50] = "zip! ";

char cadabba[50] = "zop!";

cout << abba << endl;

cout << cadabba << endl;
```

```
strcat(abba, cadabba);

cout << abba << endl;

...

zip!

zop!

zip! zop!
```

Common in String Library, to save results on first parameter

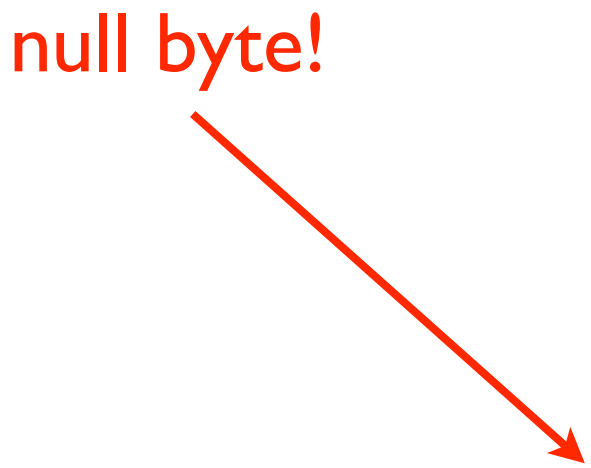
NO Error Checking in C++

Why + 1?



```
// So do something like this...


if(sizeof(abba) >= (strlen(abba) + strlen(cadabba) + 1))
   strcat(abba, cadabba);
else
   cout <<  "Opps! Get a bigger String!" << endl;
```

NO Error Checking in C++




```
// So do something like this...


if(sizeof(abba) >= (strlen(abba) + strlen(cadabba) + 1))
   strcat(abba, cadabba);
else
   cout <<  "Opps! Get a bigger String!" << endl;
```

```
char * strcpy(char *, char *);
```

String **C**o**py** - Copies the contents of one string into another.

(Including the null byte!)

```
char ono[20] = "Bahm!";

char mato[20] = "Varoooom!";

cout << ono << endl << mato << endl;

strcpy(ono, mato);

cout << ono << endl << mato << endl;

...

Bahm!
Varooom!
Varooom!
Varooom!
```

```
char * strcpy(char *, char *);
```

Another Example

```
char one[4] = "Oh,";

char two[20] = "NOOOOOOOOOOOOOOOOOOOOOOOO!!!!";

cout << one << endl << two << endl;

strcpy(one, two);

cout << one << endl << two << endl;
```

```
char * strcpy(char *, char *);
```

Another Example

```
char one[4] = "Oh,";

char two[20] = "NOOOOOOOOOOOOOOOOOOOOOOOOO!!!!";

cout << one << endl << two << endl;

strcpy(one, two);

cout << one << endl << two << endl;

...

Oh,
NOOOOOOOOOOOOOOOOOOOOOOOOO!!!!
Segmentation Fault (Core dumped)
```

```
char * strncat(char *, char *, int)

char * strncpy(char *, char *, int)
```

String **N** Con**cat**enate / **Str**ing **N C**o**py** - Just like their non-N function, but instead concatenates (or copies) exactly N characters.

```
char ono[10] = "Zip!";

char mato[25] = "Zop!Zoom!Zahm!Zowie!";

int charsLeft;

charLeft = sizeof(ono) - (strlen(ono) + 1);

strncat(ono, mato, charsLeft);

cout << ono << endl << mato << endl;
```

....

```
char * strncat(char *, char *, int)

char * strncpy(char *, char *, int)
```

String **N** Con**cat**enate / **Str**ing **N C**o**py** - Just like their non-N function, but instead concatenates (or copies) exactly N characters.

```
char ono[10] = "Zip!";

char mato[25] = "Zop!Zoom!Zahm!Zowie!";

int charsLeft;

charsLeft = sizeof(ono) - (strlen(ono) + 1);

strncat(ono, mato, charsLeft);

cout << ono << endl << mato << endl;
```

....

```
Zip!

Zip!Zop!Z
```

Another Example

```
char message[] = "I Love C++ Programming!";
char subversion[] = "GROSS!";


cout << message << endl << subversion << endl;
strncpy(message, subversion, 6);
cout << message << endl;
....
```

Another Example

```
char message[] = "I Love C++ Programming!";
char subversion[] = "GROSS!";


cout << message << endl << subversion << endl;
strncpy(message, subversion, 6);
cout << message << endl;
….
I Love C++ Programming!
GROSS!
GROSS! C++ Programming!
```

Another Example

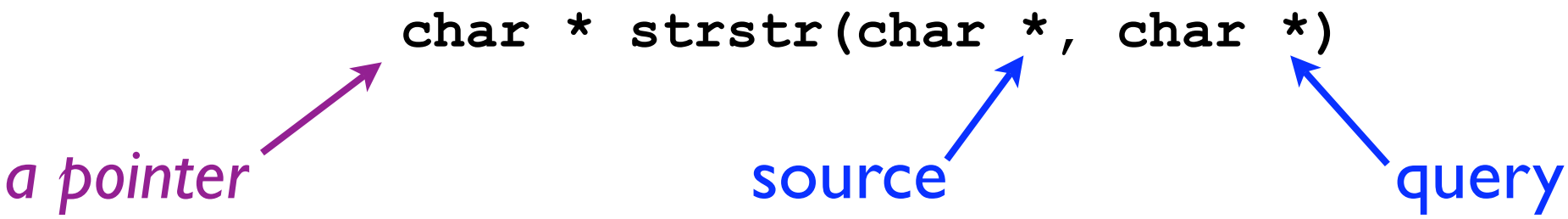
```
char message[] = "I Love C++ Programming!";
char subversion[] = "GROSS!";


cout << message << endl << subversion << endl;
strncpy(message, subversion, 7);
cout << message << endl;
....
```

Another Example

```
char message[] = "I Love C++ Programming!";
char subversion[] = "GROSS!";


cout << message << endl << subversion << endl;
strncpy(message, subversion, 7);
cout << message << endl;
….
I Love C++ Programming!
GROSS!
GROSS!
```

char * strstr(char *, char *)

Sub-String Search - Searches for a Sub-String within a Larger Str.




```
char idiom[] = "It's like trying to find a needle in a
haystack!";

char * strPtr;

cout << idiom << endl;

strPtr = strstr(idiom, "needle");

cout << strPtr << endl;

...

It's like trying to find a needle in a haystack!

needle in a haystack!
```

So many more!!!

```
char * strrchr(const char *s, int c);

int strcmp(const char *s1, const char *s2);

int strncmp(const char *s1, const char *s2, size_t count);

int strcasecmp(const char *s1, const char *s2);

int strncasecmp(const char *s1, const char *s2, size_t count);

char * strpbrk(const char *s, const char *charset);

char * strsep(char **stringp, const char *delim);

size_t strspn(const char *s, const char *charset);

size_t strcspn(const char *s, const char *charset);

char * strtok(char *s, const char *delim);
```

Write your own String Function

Typical Technical Interview Question ...
(without using any string Libraries)

```
void strncat(char dest[], char src[], int n)
{
```

Please write it on a Seperate Sheet of Paper and include your Name! (I will collect these)

```
}
```

String ⟶ Numbers Conversions

```
char number1[10] = "45387";
char number2[10] = "3.14";
char number3[10];

number3 = number1 + number2;

cout << number3 << endl;
```

Problem?

String ⟶ Numbers Conversions

```
char number1[10] = "45387";
char number2[10] = "3.14";
char number3[10];

number3 = number1 + number2;

cout << number3 << endl;
```

Problem?

Can't "+" strings.

cstdlib

```
#include <cstdlib>

int num;

long bigNum;

double realNum;

float smallerRealNum;


num = atoi("42");

bigNum = atol("8002566205");

realNum = atof("12.667");

smallerRealNum = atof("1.1");
```

Which is Larger / More Precision?

cstdlib

```
#include <cstdlib>


int num;
long bigNum;
double realNum;
float smallerRealNum;


num = atoi("42");
bigNum = atol("8002566205");
realNum = atof("12.667");
smallerRealNum = atof("1.1");
```

long larger than int

double more precision than float

Number ⟶ String Conversion

```
char * itoa(int value, char * output, int base);
```

Converts the **value** to a string (**output**) with the appropriate **base**.

```
Base: decimal = 10, octal = 8, hexadecimal = 16.
```

```
char myString[10];
int value = 256;

itoa(value, myString, 16);
cout << myString << endl;
...
0x00000100
```

Output and availability depends on compiler

C++ String Class

```
#include <string>
```

Probably what you have been using in CIS 1.5

Allows one to work with strings as a data type. (Like **`int`**, and **`char`**)

Unlike **`int`** and **`char`**, **`string`** is an abstract data type.

Defined as a class (i.e. an object).

(That means, it's defined in a library somewhere - as opposed to being a native data type.)

Advantages:

Easier of use

Dynamic in Size

Dis-Advantages:

Differing Implementation

Overhead! (Depends on how light-weight of an implementation you want)

Reading and Writing to the String Class

```
string name;

name = "anonymous";

cin >> name;

cout << "Your name is: " << name << endl;

...
Hi My Name is Chipp.
Your name is: Hi
...
```

`cin` reads only up to the first white-space.

```
   getline(cin, name);
```

Reads a line of text (including white-spaces)

```
...
Hi My Name is Chipp.
Your name is: Hi My Name is Chipp.
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special ""

string myName("Chipp Jansen");

string copyOf(myName);

string nickName(copyOf, 3);

string snore('z', 10);

string lastName(myName, 7, 6);
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special ""

string myName("Chipp Jansen"); // "Chipp Jansen"

string copyOf(myName);

string nickName(copyOf, 3);

string snore('z', 10);

string lastName(myName, 7, 6);
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special ""

string myName("Chipp Jansen"); // "Chipp Jansen"

string copyOf(myName); // "Chipp Jansen"

string nickName(copyOf, 3);

string snore('z', 10);

string lastName(myName, 7, 6);
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special ""

string myName("Chipp Jansen"); // "Chipp Jansen"

string copyOf(myName); // "Chipp Jansen"

string nickName(copyOf, 3); // "Chi"

string snore('z', 10);

string lastName(myName, 7, 6);
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special ""

string myName("Chipp Jansen"); // "Chipp Jansen"

string copyOf(myName); // "Chipp Jansen"

string nickName(copyOf, 3); // "Chi"

string snore('z', 10); // "zzzzzzzzzz"

string lastName(myName, 7, 6);
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special ""

string myName("Chipp Jansen"); // "Chipp Jansen"

string copyOf(myName); // "Chipp Jansen"

string nickName(copyOf, 3); // "Chi"

string snore('z', 10); // "zzzzzzzzzz"

string lastName(myName, 7, 6); // "Jansen"
```

Comparing and Sorting strings

Instances of the string class can be compared (much like `int` and `char`) using : `<, >, <=, >=, ==, !=`

```
string abc("abc"), def("def"), abcd("abcd"), num("123"), upper
("ABC");


if(abc < def)
   cout << "abc comes before def" << endl;
if(abc < abcd)
   cout << "abc is shorter and thus comes before abcd" << endl;
if(abc > num)
   cout << "12345 is before a" << endl;
if(abc != ABC)
   cout << "Compare is case sensitive" << endl;
```

Concatenating and Referencing

String objects can be concatenated or appended with the `+` and the `+=` operator.

```
string abba("abba");
string cadabba("cadabba");
string magic;


magic = abba + cadabba;
cout << magic << endl;
magic += cadabba;
cout << magic << endl;
magic  += cadabba += abba;
cout << magic << endl << cadabba << endl << abba << endl;
```

Concatenating and Referencing

String objects can be concatenated or appended with the `+` and the `+=` operator.

```
string abba("abba");
string cadabba("cadabba");
string magic;


magic = abba + cadabba;
cout << magic << endl;
magic += cadabba;
cout << magic << endl;
magic  += cadabba += abba;
cout << magic << endl << cadabba << endl << abba << endl;
```

```
abbacadabba
abbacadabbacadabba
abbacadabbacadabbacadabba
abba
cadabbaabba
abba
```

Menu of Member Functions

The `string` object has a large set of functions associated to it.

Since string is a class, these functions are accessed as **member functions.**

One accesses member functions through **dot notation**.

```
string chipp("chipp");
string chippsPasswd("right");


cout << chipp.length << endl;
chippsPassword.append("on");
cout << chippsPassword << endl;
chipp.swap(chippsPasswd);
cout << "login: " << chipp << endl;
cout << "passwd: " << chippsPasswd << endl;
```

Menu of Member Functions

The `string` object has a large set of functions associated to it.

Since string is a class, these functions are accessed as **member functions.**

One accesses member functions through **dot notation**.

```
string chipp("chipp");
string chippsPasswd("right");

cout << chipp.length << endl;
chippsPassword.append("on");
cout << chippsPassword << endl;
chipp.swap(chippsPasswd);
cout << "login: " << chipp << endl;
cout << "passwd: " << chippsPasswd << endl;
```

```
5
righton
login: righton
passwd: chipp
```