String & Makefile

Computer Programming for Engineers (DASF003-41)

Instructor:

Sungjae Hwang (jason.sungjae.hwang@gmail.com)

C-STRING

C-Strings

- **Array with base type *char***
  - One character per indexed variable
  - One extra character: "\0"
    - Called "null character"
    - End-of-string marker
    - Crucial to find the length of a string
- **We have used C-strings**
  - Literal "Hello" stored as c-string
    - “Hello\0”

C-String Variable

Array of characters:

```
char s[10] = "Hi Mom!";
```

- Declares a c-string variable to hold up to 9 characters
- + one null character " \0"
- Initialization places "\0" at end

Typically "partially-filled" array

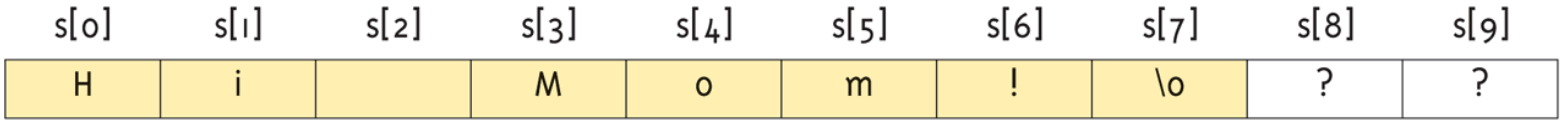
- Declare large enough to hold max-size string
- Prone to overflow error
- Must contain null character

C-String Storage

A standard array

- If s contains string "Hi Mom!", stored as:

```
char s[10] = "Hi Mom!";
```



char overflow (01_)


```
#include <iostream>

using namespace std;

int main()
{
  // Following code has errors
  char name1[5] = {'T', 'z', 'u', 'y', 'u'};
  char name2[4] = {'S', 'a', 'n', 'a'};
  char name3[6] = {'D', 'a', 'h', 'y', 'u', 'n'};

  cout << name1 << endl;
  cout << name2 << endl;
  cout << name3 << endl;
}
```

C-String Initialization

Can omit array-size:

```
char short_string[4] = “abc";
char short_string[] = “abc”;
```

- Automatically makes size one more than length of quoted string
- NOT same as:

```
char short_string[] = “abc”;
char short_string[] = {"a", "b", "c" };
```

short string (04_)


```
#include <iostream>

using namespace std;

int main()
{
  char short_string1[] = "abcdefg";
  cout << "String1: " << short_string1 << endl;
  cout << "Size: " << sizeof(short_string1) << endl;

  // What happens with the below code?
  char short_string2[] = {'k', 'l', 'm'};
  // error with ""
  //char short_string2[] = {"k", "l", "m"};
  cout << "String2: " << short_string2 << endl;
  cout << "Size: " << sizeof(short_string2) << endl;
}
```

char initialization (03_)


```
#include <iostream>

using namespace std;

int main()
{
  // Following code has errors
  char s1[10];
  char s2[10] = "";
  char s3[10] = {};
  char s4[10] = {'\0'};

  // Checking items of s
  cout << "s1" << endl;
  for(char c : s1)
    cout << int(c) << " ";
  cout << endl;

  cout << "s2" << endl;
  for(char c : s2)
    cout << int(c) << " ";
  cout << endl;

  cout << "s3" << endl;
  for(char c : s3)
    cout << int(c) << " ";
  cout << endl;

  cout << "s4" << endl;
  for(char c : s4)
    cout << int(c) << " ";
  cout << endl;

}
```

C-String Manipulation

- Can manipulate indexed variables

```
char happyString[7] = “DoBeDo";
happyString[6] = ‘Z’;
```

- “\0” (null) was overwritten by a “Z”!
  - NOT C-String anymore
    - Unpredictable Result

char


```
 1 #include <iostream>
 2 using namespace std;
 3
 4 int main(){
 5   char a[3] = "aa";
 6   char happyString[7] = "DoBeDo";
 7   char b[3] = "bb";
 8
 9   cout << "Before: " << happyString << endl;
10   cout << "Before: " << a << endl;
11   cout << "Before: " << b << endl;
12   happyString[6] = 'Z';
13   cout << "After: " << happyString << endl;
14   cout << "Before: " << a << endl;
15   cout << "Before: " << b << endl;
16   return 0;
17 }
```

char


```
#include <iostream>

using namespace std;

int main()
{
  char s[10] = "Hi Mom!";

  cout << s << endl;

  // Checking items of s
  for(char c : s)
    cout << c << ":" << int(c) << endl;
}
```

C-String Assignment

- C-strings not like other variables
  - Cannot assign

```
char aString[10];
aString = “Hello”;
```

- Must use library function for assignment:

```
strcpy(aString, “Hello”);
```

  - Built-in function (in <cstring>)
  - No checks for size!

C-String Comparison

- C-strings not like other variables
  - Cannot compare with == operator

```
char aString[10] = “Hello”;
aString == “Hello”;
```

- Must use library function for comparison:

```
if(strcmp(aString, “Hello”))
   cout << “not same”;
else
   cout << “same”;
```

C-String Length

- **Often useful to know string length**
  - Cannot compare with == operator

```
char myString[10] = “dobedo”;
cout << strlen(myString);
```

- **Note Result:**
  - 6 (Not including NULL)

C-String Concatenation

- String concatenate
  - strcat()

```
char stringVar[20] = “The rain”;
strcat(stringVar, “in Spain”);
```

- Note Result:
  - “The rainin Spain”
  - Incorporate spaces as needed!

Library

Declaring C-strings

- Requires no C++ library
- Built into standard C++

Manipulations

- Require library <string.h> or <cstring> (std::* versions)
- strcpy, strlen, str*, ... functions are included.

C++ STRING CLASS

Standard Class string

- **C++ has a data type of "string" to store sequences of characters**
  - Not a primitive data type
  - Should add #include <string> at the top of the program

```
#include <string>

std::string s; // with std namespace
```

```
#include <string>
using namespace std;

string s; // without std namespace
```

- **Many operators and functions defined for manipulation**
  - Same operations available as C-strings, and more!
  - Over 100 members of standard string class

Standard Class string

Example member functions:

- can assign, compare, add:
- `.length()`: returns length of string variable
- `.at(i)`: returns reference to char at position i `(s[i])`

```
string s1, s2, s3;
s3 = s1 + s2;     // concatenation (as strcat)
s3 = "Hello Mom!";   // assignment (as strcpy)
```

- Note c-string "Hello Mom!" automatically converted to string type!
- the “+” operator on strings concatenates two strings together

String reference


```
1 #include <iostream>
  2 using namespace std;
  3
  4 int main(){
  5   string str = "Mary";
  6
  7   cout << str[6] << endl;
  8  // cout << str.at(6) << endl;
  9
 10   return 0;
 11 }
```

C-string and string Object Conversions

From C-string to string object

- Automatic type conversions

```
char      aCString[] = "My C-string";
string    stringVar = aCString;
```

From string object to C-string

- string object cannot be assigned, since it's not a pointer must use explicit conversion:

```
strcpy( aCString, stringVar.c_str() );
aCString = stringVar; // illegal!
```

obj conversion


```
#include <iostream>
#include <string>

using namespace std;

int main() {
  char aCString[] = "My C-string";
#if 1
  string stringVar = aCString;

  cout << "C String: " << aCString << endl;
  cout << "string: " << stringVar << endl;

#else
  string stringVar = "C++";
  strcpy( aCString, stringVar.c_str() );
  //aCString = stringVar; // illegal!
  //aCString = stringVar.c_str(); // What happens with this code?

  cout << "C String: " << aCString << endl;
  cout << "string: " << stringVar << endl;
#endif
}
```

C-String Comparison

- Could compare just like primitive data type

```
//c-string comparison
if(strcmp(aString, “Hello”))
   cout << “not same”;
else
   cout << “same”;

//string class comparison
if(str1 == str2)
   cout << “same”
else
   cout << “not same”
```

Standard Class string

Display 9.4: Program Using the Class string

```
// demonstrates the standard class string.
#include <iostream>
#include <string>
using namespace std;

int main( )
{
   string phrase; //initialized to the empty string

   //two equivalent ways of initializing a string variable
   string adjective("fried"), noun("ants");
   string wish = "Bon appetite!";

   phrase = "I love " + adjective + " " + noun + "!";
   cout << phrase << endl << wish << endl;
   return 0;
}
```

It may work without including, BUT it is not guaranteed.

```
I love fried ants!
Bon appetite!
```

Conversion between string and numbers

String to numbers

- In C++11, it is simply a matter of calling
- stof, stod, stoi, or stol to convert a string to
- a float, double, int, or long, respectively.

```
int          i;
double       d;
string       s;
i = stoi("35");  // string "35" to an integer 35
d = stod("2.5"); // string "2.5" to the double 2.5
```

Numbers to string

```
string s = to_string(d*2);  // double 5.0 to a string "5.0000"
```

Member Functions

Display 9.7 Member Functions of the Standard Class string

Display 9.7 Member Functions of the Standard Class `string`

EXAMPLE	REMARKS
Constructors	
`string str;`	Default constructor; creates empty `string` object `str`.
`string str("string");`	Creates a `string` object with data `"string"`.
`string str(aString);`	Creates a `string` object `str` that is a copy of `aString`. `aString` is an object of the class `string`.
Element access	
`str[i]`	Returns read/write reference to character in `str` at index `i`.
`str.at(i)`	Returns read/write reference to character in `str` at index `i`.
`str.substr(position, length)`	Returns the substring of the calling object starting at `position` and having `length` characters.
Assignment/Modifiers	
`str1 = str2;`	Allocates space and initializes it to `str2`'s data, releases memory allocated for `str1`, and sets `str1`'s size to that of `str2`.
`str1 += str2;`	Character data of `str2` is concatenated to the end of `str1`; the size is set appropriately.
`str.empty( )`	Returns `true` if `str` is an empty `string`; returns `false` otherwise.

(continued)

Member Functions

Display 9.7 Member Functions of the Standard Class string

Display 9.7 Member Functions of the Standard Class `string`

EXAMPLE	REMARKS
`str1 + str2`	Returns a `string` that has `str2`'s data concatenated to the end of `str1`'s data. The size is set appropriately.
`str.insert(pos, str2)`	Inserts `str2` into `str` beginning at position `pos`.
`str.remove(pos, length)`	Removes substring of size `length`, starting at position `pos`.
Comparisons erase	
`str1 == str2  str1 != str2`	Compare for equality or inequality; returns a Boolean value.
`str1 < str2  str1 > str2`	Four comparisons. All are lexicographical comparisons.
`str1 <= str2  str1 >= str2`	
`str.find(str1)`	Returns index of the first occurrence of `str1` in `str`.
`str.find(str1, pos)`	Returns index of the first occurrence of string `str1` in `str`; the search starts at position `pos`.
`str.find_first_of(str1, pos)`	Returns the index of the first instance in `str` of any character in `str1`, starting the search at position `pos`.
`str.find_first_not_of (str1, pos)`	Returns the index of the first instance in `str` of any character *not* in `str1`, starting search at position `pos`.

str.length(): Returns the length of str

string example


```
#include <iostream>

using namespace std;

int main()
{
  string str("abcde");

  str.erase(0,1);
  cout << str << endl;
}
```

Summary

- **String class**
  - Primitive data type
- **Over 100 member functions**
- **length, stoi, stod, stol, stof**

MAKEFILE

Prof. Hyungjoon Koo's slides are adapted and modified for this lecture.

What does make do?

- **A tool that controls the generations of executables from sources**
  - Each source may require its own option
  - Compilation is a very complex process, often taking a long time
  - When recompiling a vast program, it is cumbersome to redo from the scratch
  - Just read a description file (Makefile) for generating an executable
  - It allows a developer to save a lot of time with an automated compilation process

Example

- **Suppose that we have four source code**
  - One header file: todo.h
  - Three C++ source code: main.cc, my_cal.cc, my_memo.cc
  - Target executable: todo

Source Code

todo.h

```
#include <iostream>
using namespace std;

int my_memo();
void my_cal();
```

main.cc

```
#include "todo.h"

int main(int argc, char** argv) {
   my_memo();
   my_cal();
   return 0;
}
```

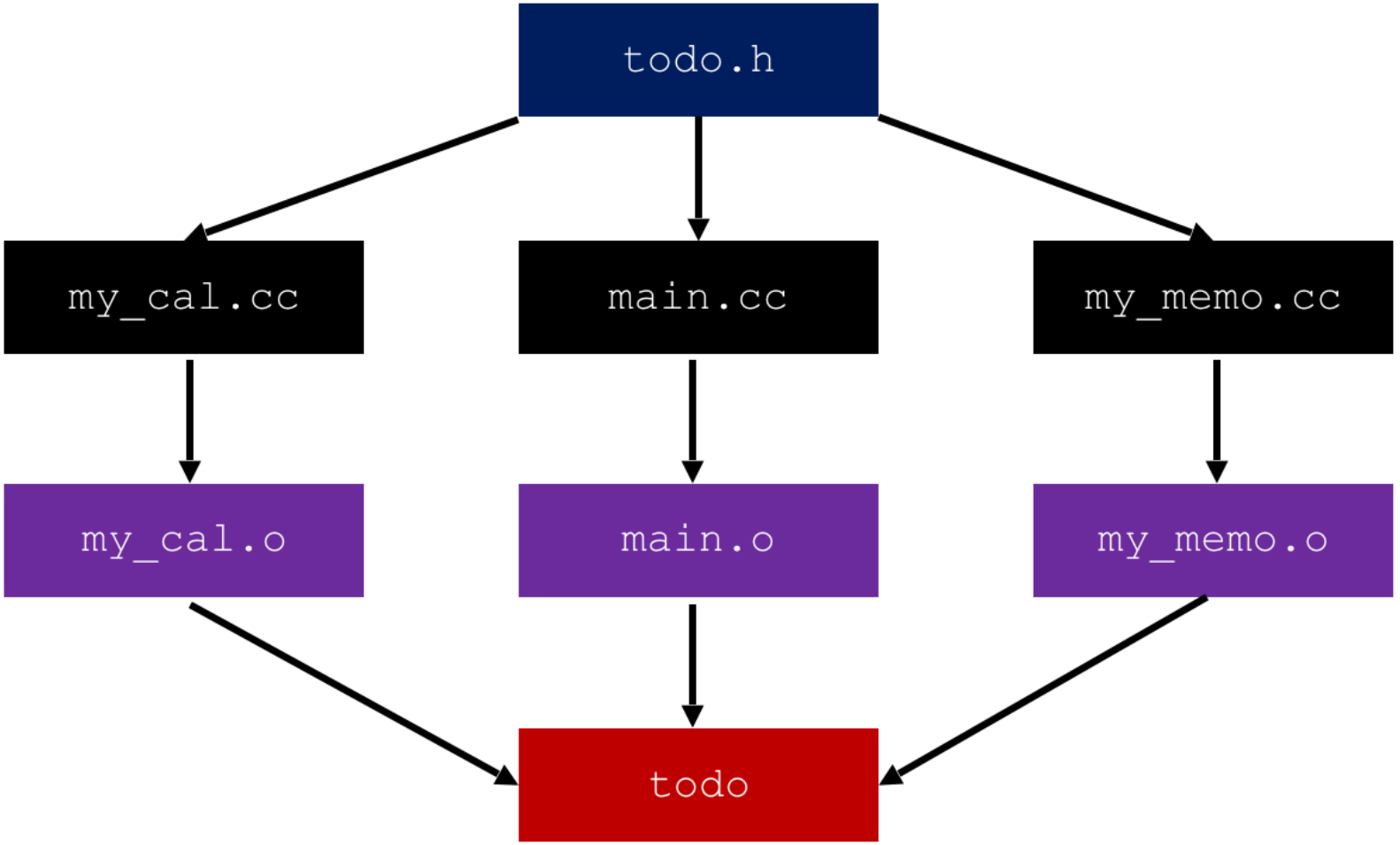
my_cal.cc

```
#include "todo.h"

void my_cal() {
  cout << "My calendar!\n";
}
```

my_memo.cc

```
#include "todo.h"

int my_memo() {
   cout << “My Memo” << endl;
   return 0;
}
```

Compilation



General Rules of Makefile

- **Consisting of (target:dependency), followed by commands**
  - A dependency can be omitted
  - The commands must begin with <TAB>
- **Line**
  - Empty lines will be ignored
  - When a line is too long, \ can be used at the end of the line to continue
- **# means a comment**

Makefile syntax (1)

Makefile Syntax

```
target(s) : dependency(ies)
<tab> command
<tab> command
```

Concrete Example

```
helloword: helloworld.cc
    g++ -o helloworld helloworld.cc

clean:
    rm helloworld
```

How make interprets Makefile

- **Find the description file such as Makefile**
- **Find a target**
- **Find a dependency**
- **If the dependency can be found in the directory**
  - Check if the creation time of the target is newer than the dependency
  - If true, do not proceed commands
  - If false, proceed commands

Makefile (ver 1)


```
all: todo

todo: my_memo.o my_cal.o main.o
	g++ -W -Wall -o todo my_memo.o my_cal.o main.o

my_memo.o: my_memo.cc
	g++ -W -Wall -c -o my_memo.o my_memo.cc

my_cal.o: my_cal.cc
	g++ -W -Wall -c -o my_cal.o my_cal.cc

main.o: main.c
	g++ -W -Wall -c -o main.o main.cc

clean:
	rm -rf *.o todo
```

Makefile syntax (2)

- Variables can be only string
  - Syntax: $(var) -> substitution

- Concrete Example

```
SOURCE = helloworld.c
TARGET = helloworld

helloword: $(SOURCE)
    g++ -o $(TARGET) $(SOURCE)
clean:
    rm $(TARGET)
```

Makefile (ver 2)


```
CC=g++
CXXFLAGS=-W -Wall
TARGET=todo
all: $(TARGET)

todo: my_memo.o my_cal.o main.o
	$(CC) $(CXXFLAGS) -o $(TARGET) my_memo.o my_cal.o main.o

my_memo.o: my_memo.cc
	$(CC) $(CXXFLAGS) -c -o my_memo.o my_memo.cc

my_cal.o: my_cal.cc
	$(CC) $(CXXFLAGS) -c -o my_cal.o my_cal.cc

main.o: main.cc
	$(CC) $(CXXFLAGS) -c -o main.o main.cc

clean:
	rm -rf *.o todo
```

Makefile syntax (3)

Special Symbols

- $@ : current target name
- $^ : current dependency list

Concrete Example

```
SOURCE = helloworld.c
TARGET = helloworld
helloword: $(SOURCE)
    g++ -o $@ $^
clean:
    rm $(TARGET)
```

Makefile (ver 3)


```
CC=g++
CXXFLAGS=-W -Wall
TARGET=todo
all: $(TARGET)

todo: my_memo.o my_cal.o main.o
	$(CC) $(CXXFLAGS) -o $@ $^

my_memo.o: my_memo.cc
	$(CC) $(CXXFLAGS) -c -o $@ $^

my_cal.o: my_cal.cc
	$(CC) $(CXXFLAGS) -c -o $@ $^

main.o: main.cc
	$(CC) $(CXXFLAGS) -c -o $@ $^

clean:
	rm -rf *.o todo
```

Makefile syntax (4)

Implicit Rules (magic rules)

- You don’t need to specify them
- e.x. compiling a c++ program
  - A.o is made automatically from a.cc or a.cpp

Concrete Example

```
…
OBJS = my_memo.o my_cal.o main.o

todo: $(OBJS)
        $(CC) $(CXXFLAGS) -o $@ $^
…
```

Makefile (ver 4)


```
CC=g++
CXXFLAGS=-W -Wall
TARGET=todo

all: $(TARGET)

OBJS = my_memo.o my_cal.o main.o

todo: $(OBJS)
        $(CC) $(CXXFLAGS) -o $@ $^

clean:
        rm -rf *.o todo
```

Programming Assignments

Weekly Assignments vs Programming Assignments

- Weekly Assignments: goorm & single source file
- Programming Assignments: terminal & multiple source files

Programming Assignments

- Do programming (edit *.cc source code)
- Configure build script (Makefile)
- Your makefile should produce runnable program
- Submit your solution (source codes) with Makefile

Reference

- https://makefiletutorial.com/