References and Command Line Arguments

This week

- **Project 3 Code Skeleton:** due Wed**, Apr 12** (with free 24 hr extension)
- **Design meetings for those who are doing CYO**
  - **OR want extra help on the project!**
- **Quiz 8:** due Sunday
- **3-2-1:** due Friday

- **Practicum 3:** Apr 17th in class

References

What is a reference?

- A reference is special type of variable that is an *alias* or *alternate name* for another variable in the program
- It is denoted with an ampersand symbol - &

```
int i = 10;
int &r = i; // has to be a variable
int &w = 10;
double &q = i;
```

- `r` is an integer reference initialized to `i`

```
int i = 10;
int &r = i;
cout << i << endl;
cout << r << endl;

r = 20;
cout << i << endl;
cout << r << endl;

i = 30;
cout << i << endl;
cout << r << endl;
```

```
i = 10
r = i = 10
10
10

i = 20
r = 20
20
20

i = 30
r = 30
30
```

Pass by value

```
void withdraw(double balance, double amount)
{
    if(amount > balance)
    {
        cout << "Withdrawal amount exceeds current balance" << endl;
    }
    else
    {
        balance = balance - amount;
    }
}

int main()
{
    double balance = 1000;
    double amount = 60;
    withdraw(balance,amount);
    cout << balance << endl;
    return 0;
}
```

balance = 940

balance = 1000

Function parameters

- The parameters we've see so far are called ***value parameters***
- We return the variable if the updated information is required in main()

Pass by value – return the updated data

```
double withdraw(double balance, double amount)
{
    if(amount > balance)
    {
        cout << "Withdrawal amount exceeds current balance" << endl;
        return balance;
    }
    else
    {
        return balance - amount;
    }
}
```

Pass by reference

- Passing the actual variable instead of a copy
- References are sent as formal parameters
- The withdraw function will be modified from value parameters

```
void withdraw(double balance, double amount);
```

to now accept reference parameters

```
void withdraw(double& balance, double amount);
```

`double&` is called reference to a double or double ref

Pass by reference

```
double withdraw(double& balance, double amount)
{
    if(amount > balance)
    {
        cout << "Withdrawal amount exceeds current balance" << endl;
    }
    else
    {
        balance = balance - amount;
    }
}
int main()
{
    double balance = 1000;
    double amount = 60;
    withdraw(balance,amount);
    cout << balance << endl;
    return 0;
}
```

balance = 940

balance = 940

Functions with reference parameters

- If a function prototype lists a reference parameter then the arguments during function call have to be variables.

```
void withdraw(double& balance, double amount);
double balance = 1000;
double amount = 60;
withdraw(balance, amount);
```

- Constants like the example below will cause an error

```
withdraw(1000, 60);
withdraw(balance + 1000, 60);
```

Output parameters - references

- Use reference parameters to save computation result

```
double add(double n1, double n2)
{
        double sum = n1 + n2;
        return sum;
}
```

- Modified to a void function with a double ref output parameter

```
void add(double n1, double n2, double& sum)
{
        sum = n1 + n2;
}
```

Advantages of references

- References don’t consume extra memory
- Manipulating a reference directly, updates the variable being references
- Once initialized, a reference cannot to re-initialized to a new variable

```
int i = 10;
int &r = i;
int *p = &i;
int a = 20;
r = a; // a = 20; r = 20; i = 20
&r = a; // not allowed
```

Command Line Arguments

Running a program

Depending on the operating system (OS) and C++ development system used, have different options for running a program:

- Select “Run” in the compilation environment
- Click on an icon somewhere
- Type the program name in a prompt in the command shell window
  - This is the method we’ve been using for compiling multiple files with our classes:

```
g++ -std=c++17 Player.cpp Team.cpp gameDriver.cpp
```

Command Line

How to get the command shell window (terminal window) from your OS:

- **Windows:** type “cmd” in the Search box and click on cmd.exe
  - powershell, git bash etc.
- **Mac:** Search for “terminal” and open it
- **Linux:** Search for “terminal” and open it

How to get the command shell window (terminal window) in VS Code:

- Click on **Terminal** at the top toolbar,
- then click **New Terminal**

How is a cpp program executed?

- g++ -std=c++17 greeting .cpp
- ./a.out

- g++ -std=c++17 greeting .cpp –o prog
- ./prog

Command Line Arguments

- Your *execution* of the program from the command line is actually *calling* the `main()` function!

  It's a function…

  … so we can give that thing some input arguments!

Command Line Arguments

No matter how you run your program, you can pass some information into the program via command line arguments

- These arguments are passed to the main function the same way you pass arguments into any old function

Command Line Arguments: `main()`

- For our program to process command line arguments, we must make a few changes to our main function:

```
int main(int argc, char* argv[])
{
... do stuff ...
}
```

- **argc** = **arg**ument **c**ount. argc = 1 if the user typed nothing after the program name (1 arg)
- **argv** = **arg**ument **v**ector. Not a real vector, but just a bunch of character pointers (behaves like a bunch of strings for the arguments you give)

Command Line Arguments: example

The user might type into a command shell window for starting the program named **`prog`**:

```
./prog -v input.txt
```

- **`prog`** is the program name (your C++ program)
- "**`-v`**" and "**`input.txt`**" are command line arguments

The **`-`** in **`-v`** typically indicates an option.

- Strings that start with a "-" are processed as options
- Strings that do not start with a "-" are usually file names

Command Line Arguments: example

```
int main(int argc, char* argv[])
{
    ...
}
```

Example user input:

./prog -v input.txt

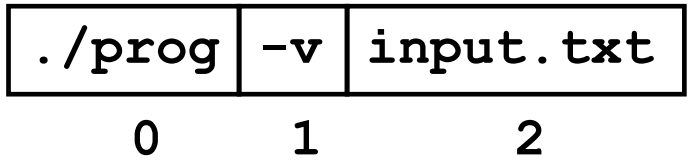
argc is **3**

argv contains these three strings:

argv[0]: "./prog"

argv[1]: "-v"

argv[2]: "input.txt"



Command Line Arguments: example

Example: Let's write a program that encrypts/decrypts a file using a caesar cypher. Take as input from the command line:

- an input file name (to encrypt/decrypt)
- an output file name (for the encrypted/decrypted file)
- an optional flag -d to denote we should decrypt the file instead of encrypt

Command Line Arguments: example

- So, our code will not prompt the user for file names! We will pass them in as arguments:

To encrypt:

```
./caesar.o input.txt encrypted.txt
```

To decrypt:

```
./caesar.o -d encrypted.txt decrypted.txt
```

Command Line Arguments: example

- Let's write a program that encrypts a file
  - scrambles it so that it is unreadable except to those who know the decryption method.
- Ignoring 2,000 years of progress in encryption, use a method familiar to Julius Caesar
  - replacing an A with a D, a B with an E, and so on.
  - each character c is replaced with c + 3

Plain text:

l	a	r	g	e		p	i	z	z	a

Encrypted text:

o	d	u	j	h		s	l	c	c	d

Examples

- See github