CS601: Software Development for Scientific Computing

Autumn 2022

Week8: Intermediate C++ (templates), tools: gdb

Course progress..

- **Last Week:** Object Orientation
  - Example classes – `Vector, Fruit`
  - Encapsulation, overloading
  - Inheritance and polymorphism, overriding
  - Const and References
  - STL (e.g. `vector`), Exception handling via `try-catch`

- **Next:**
  - STL (other types), Template programming
  - gdb

Recap: Returning References-Example1

- How can we assign one object to another?

```
Apple a1(“Apple”, 1.2); //constructor Apple::Apple(string, float)
                         //is invoked
Apple a2; //constructor Apple::Apple() is invoked.
a2  = a1 //object a1 is assigned to a2;assignment operator is invoked
```

```
Apple& Apple::operator=(const Apple& rhs)
```

Called Copy Assignment Operator

```
Apple& Apple::operator=(const Apple& rhs) {
commonName = rhs.commonName;
weight = rhs.weight;
energyPerUnitWeight = rhs.energyPerUnitWeight;
constituents = rhs.constituents;
return *this;
}
```

What is Move Assignment Operator?

`this`

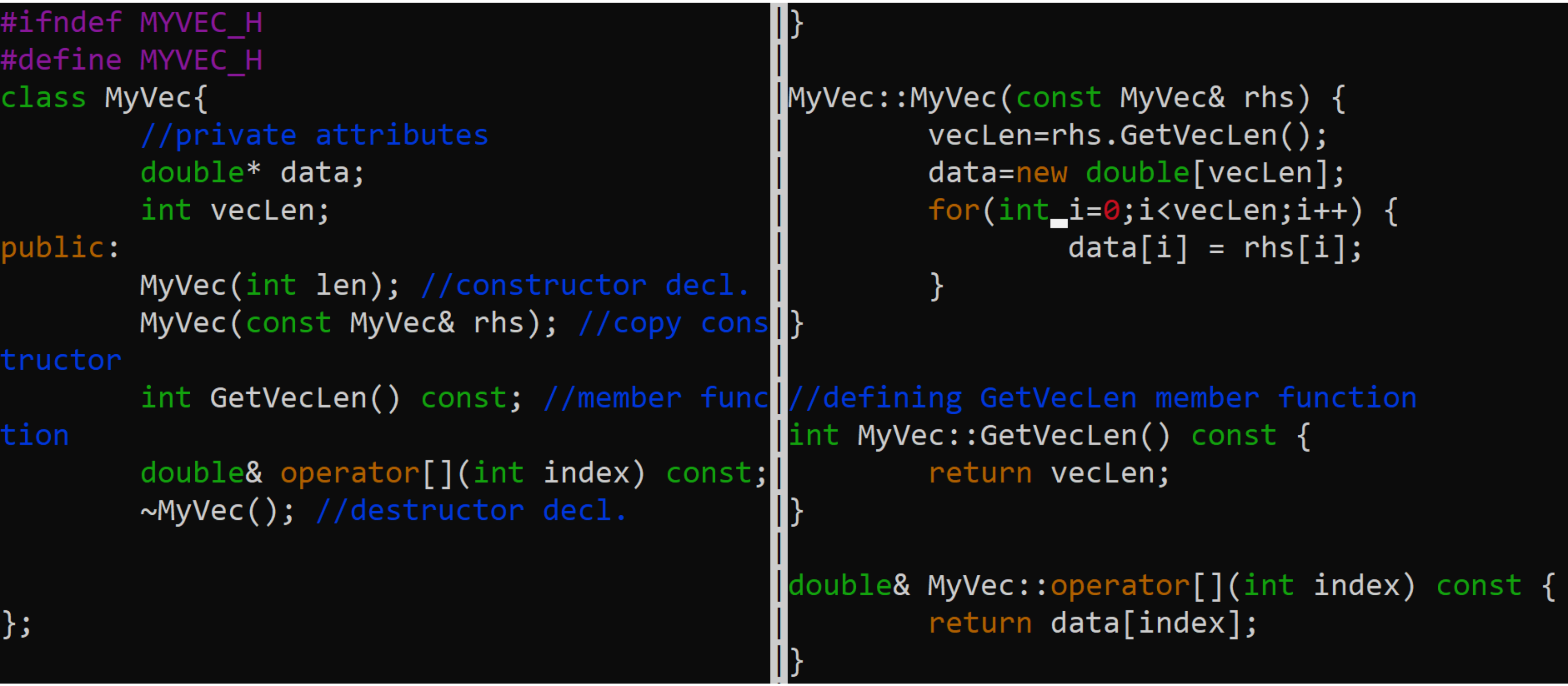
- Implicit variable defined by the compiler for every class
  - E.g. `MyVec *this;`
- All member functions have `this` as an implicit first argument
  - E.g.

```
int MyVec::GetVecLen() const;
```

would actually be:

```
int MyVec::GetVecLen(MyVec* this) const;
```

Returning References – Example2



```cpp
#ifndef MYVEC_H
#define MYVEC_H
class MyVec{
        //private attributes
        double* data;
        int vecLen;
public:
        MyVec(int len); //constructor decl.
        MyVec(const MyVec& rhs); //copy constructor
        int GetVecLen() const; //member function
        double& operator[](int index) const;
        ~MyVec(); //destructor decl.


};
```

```cpp
}

MyVec::MyVec(const MyVec& rhs) {
        vecLen=rhs.GetVecLen();
        data=new double[vecLen];
        for(int i=0;i<vecLen;i++) {
                data[i] = rhs[i];
        }
}

//defining GetVecLen member function
int MyVec::GetVecLen() const {
        return vecLen;
}

double& MyVec::operator[](int index) const {
        return data[index];
}
```

```cpp
MyVec v1;
v1[0]=100;
```

Overloading +=

- `MyVec v1;`

  `v1+=3;`
- `MyVec& MyVec::operator+=(double)`

Overloading +=

- `MyVec v1;`

  `v1+=3;`

  - `MyVec& MyVec::operator+=(double)`

- `MyVec v2;`

  `v2+=v1;`

  - `MyVec& MyVec::operator+=(const MyVec& rhs)`
  - What if you make the return value above `const`?

    Disallow: `(v2+=v1)+=3;`

Overloading +

- `v1=v1+3;` ***Single-argument constructors:** allow implicit conversion from a particular type to initialize an object.*
  - `const MyVec MyVec::operator+(double val)`
- `v3=v1+v2;`

```
1. const MyVec MyVec::operator+(const MyVec&
vec2) const;
```

OR

```
2. friend const MyVec operator+(const MyVec&
lhs, const MyVec& rhs);
```

v1=3+v1 is compiler error! Why?

Operator Overloading - Guidelines

- If a binary operator accepts operands of different types and is commutative, both orders should be overloaded
- Consistency:
  - If a class has ==, it should also have !=
  - += and + should result in identical values
  - define your copy assignment operator if you have defined a copy constructor

Exercise

- What member functions does class `MyVec` should define to support:

```
MyVec v2;
v2=-v1; //v1 is of type MyVec
```

- Bonus: How to define pre-increment (`++obj`) and post-increment (`obj++`) operations?

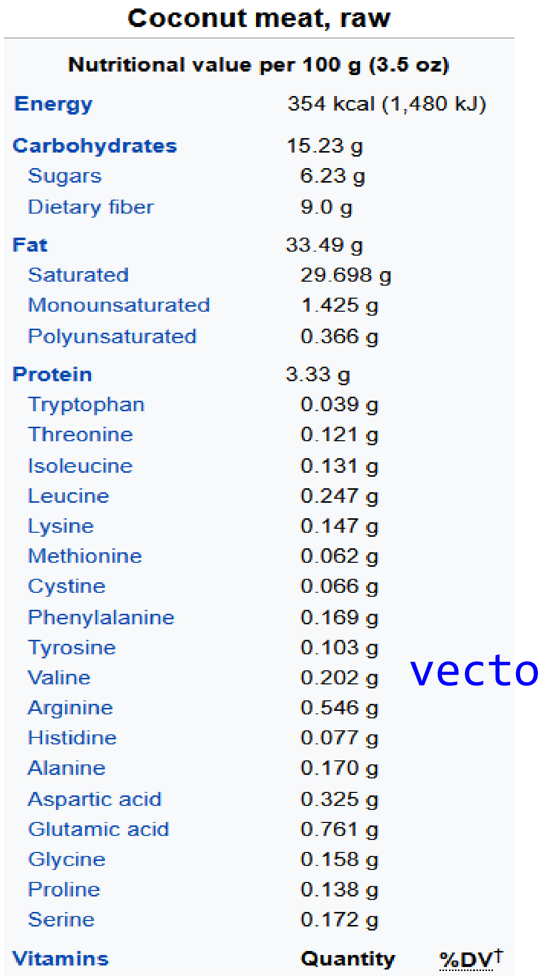
Standard Template Library (STL)

- Large set of frequently used data structures and algorithms
  - Defined as *parametrized* data types and functions
  - Types to represent complex numbers and strings, algorithms to sort, get random numbers etc.
- Convenient and bug free to use these libraries
- E.g. `vector, map, queue, pair, sort` etc.
- Use your own type only for efficiency considerations - *only if you are sure!*

STL - Motivation

Real-world view

source:wikipedia



Coconut meat, raw

Nutritional value per 100 g (3.5 oz)		
Energy	354 kcal (1,480 kJ)	
Carbohydrates	15.23 g	
Sugars	6.23 g	
Dietary fiber	9.0 g	
Fat	33.49 g	
Saturated	29.698 g	
Monounsaturated	1.425 g	
Polyunsaturated	0.366 g	
Protein	3.33 g	
Tryptophan	0.039 g	
Threonine	0.121 g	
Isoleucine	0.131 g	
Leucine	0.247 g	
Lysine	0.147 g	
Methionine	0.062 g	
Cystine	0.066 g	
Phenylalanine	0.169 g	
Tyrosine	0.103 g	
Valine	0.202 g	
Arginine	0.546 g	
Histidine	0.077 g	
Alanine	0.170 g	
Aspartic acid	0.325 g	
Glutamic acid	0.761 g	
Glycine	0.158 g	
Proline	0.138 g	
Serine	0.172 g	
Vitamins	**Quantity**	**%DV**†

Consider the nutrients (constituents) present in edible part of coconut.

How would you capture the Real-world view in a Program?

```
vector<pair<string, float> > constituents;
```

Container

- Holder of a collection of objects
- Is an object itself
- Different types:
  - sequence container
  - associative container (ordered/unordered)
  - container adapter

Sequence Container

- Provide fast sequential access to elements
- Factors to consider:
  - Cost to add/delete an element
  - Cost to perform non-sequential access to elements

container name	**comments**
`vector`	Flexible array, fast random access
`string`	Like `vector.` Meant for sequence of characters
`list/slist`	doubly/singly linked list. Sequential access to elements (bidirectional/unidirectional).
`deque`	Double-ended queue. Fast random access, Fast append
`array`	Intended as replacement for ‘C’-style arrays. Fixed-sized.

Container Adapter

- Provide an interface to sequence containers
  - `stack, queue, priority_queue`

Associative Container

- Implement sorted data structures for efficient searching (O(log n)) complexity.
  - Set, map, multiset, multimap

container name	**comments**
`set`	Collection of unique sorted keys. Implemented as class template
`map`	Collection of key-value pairs sorted by unique keys. Implemented as class template

Unordered Associative Container

- Implement hashed data structures for efficient searching (O(1) best-case, O(n) worst-case complexity).
  - `unordered_set, unordered_map, unordered_multiset, unordered_multimap`

Templating Functions

- Provide a recipe for generating multiple versions of the function based on the data type of the data on which the function operates

Function Templates - Goal

```
double scprod(int len,
         double* vec1,
         double* vec2)
{
    double result;
    //compute result
    //return result
}
```

```
int scprod(int len,
         int* vec1,
         int* vec2)
{
    int result;
    //compute result
    //return result
}
```

How can you avoid multiple implementations of the same functionality but with different types?

Function Templates – Implementation and Invocation

```
template<typename T>
double scprod(int len,
         T* vec1,
         T* vec2)
{
    T result;
    //compute result
    //return result
}
```

Add this template definition in .h file! why .h and not .cpp?

Called template parameter. Can choose any name other than T. the keyword 'typename' can be replaced with 'class'

```
int main() {
//define vec1-vec4
scprod<double>(10,vec1, vec2); //explicit instantiation
scprod<int>(100,vec3,vec4); //explicit instantiation
scprod(100, vec3,vec4); //implicit instantiation
```

Class Templates

- Like function templates but for templating classes

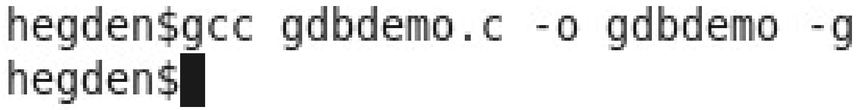
Refer to demo example for class and function templates

GDB

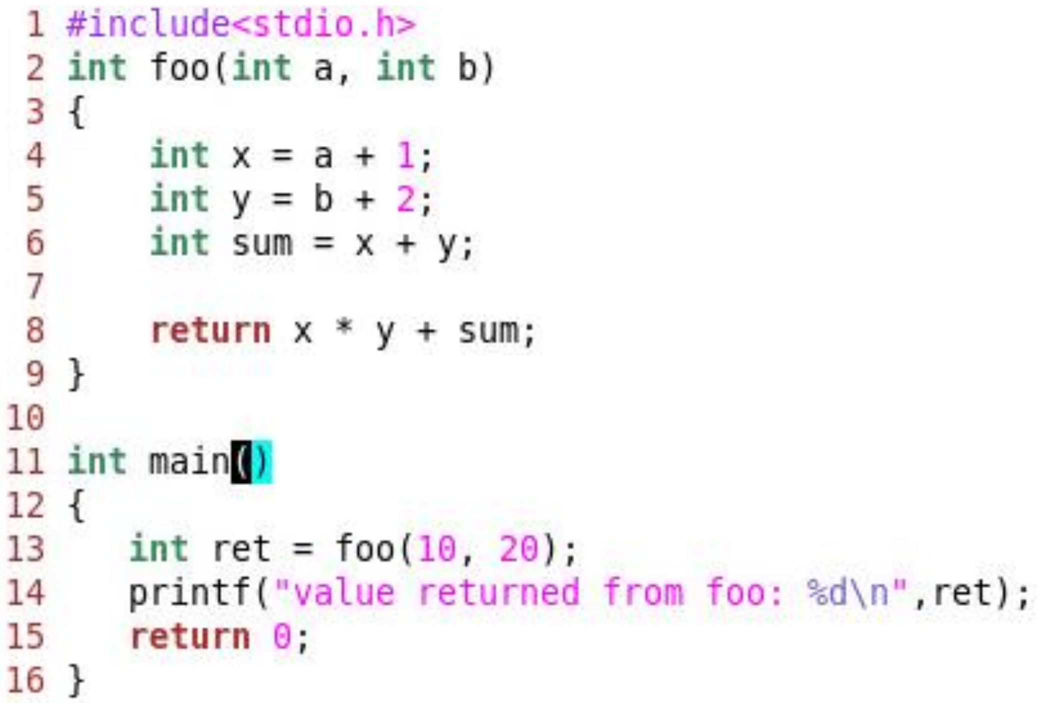
- GNU Debugger – A tool for inspecting your C/C++ programs
  - How to begin inspecting a program using gdb?
  - How to control the execution?
  - How to display, interpret, and alter memory contents of a program using gdb?
  - Misc – displaying stack frames, visualizing assembler code.

GDB

– Compile your programs with –g option



```
hegden$gcc gdbdemo.c -o gdbdemo -g
hegden$
```



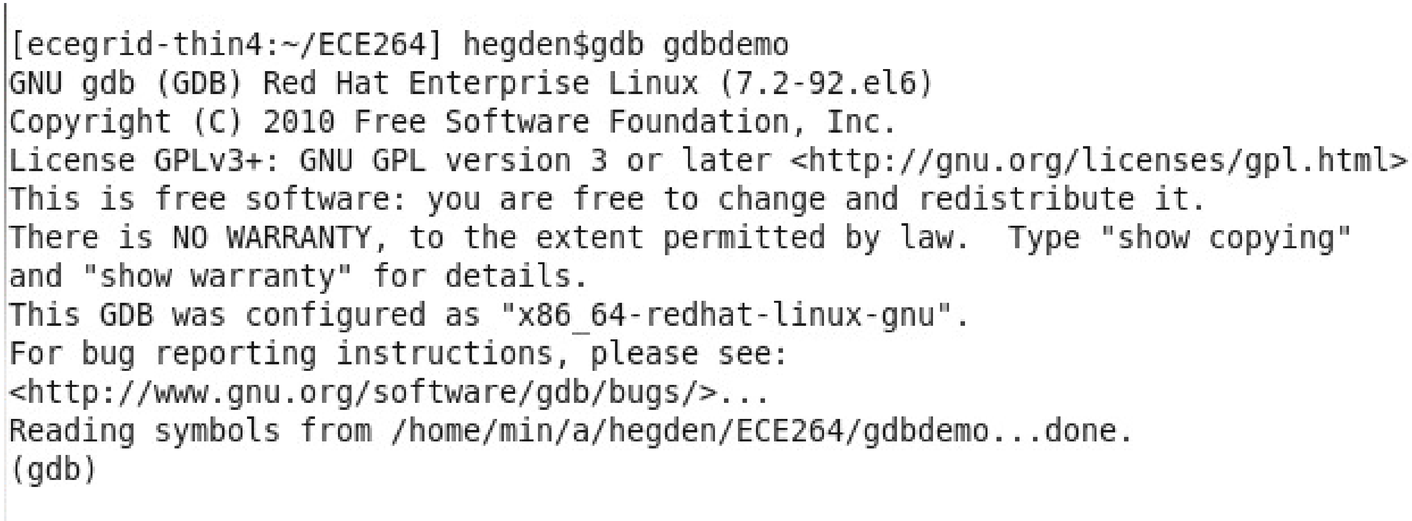
```
#include<stdio.h>
int foo(int a, int b)
{
    int x = a + 1;
    int y = b + 2;
    int sum = x + y;

    return x * y + sum;
}

int main()
{
   int ret = foo(10, 20);
   printf("value returned from foo: %d\n",ret);
   return 0;
}
```

GDB – Start Debug

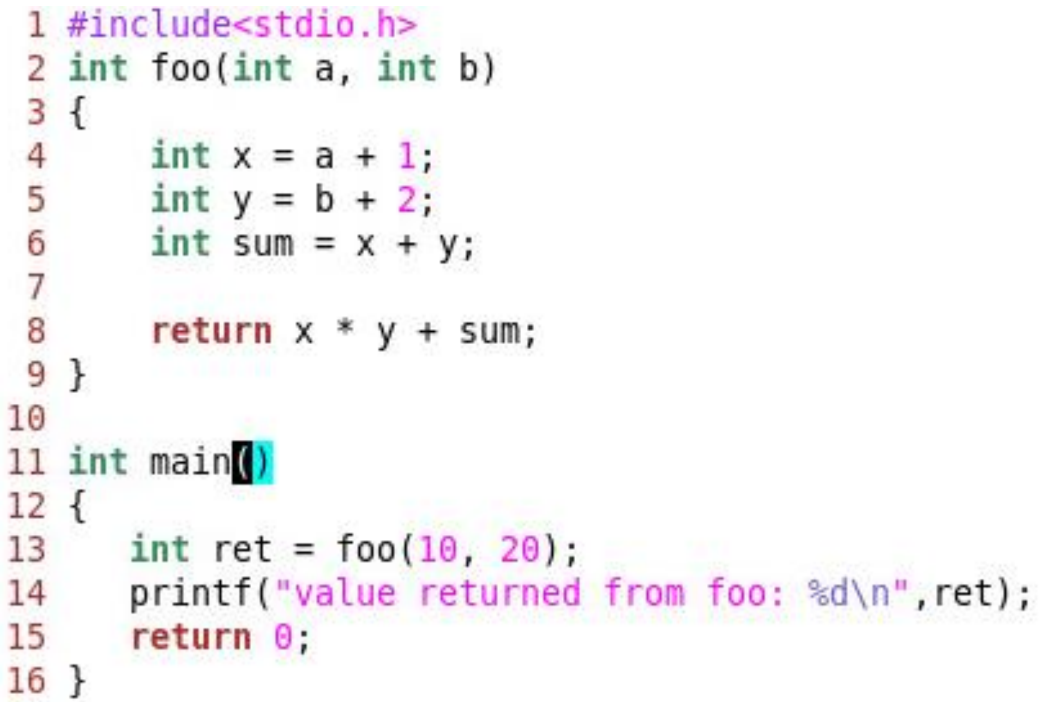
- Start debug mode (`gdb gdbdemo`)
  - Note the executable on first line (not .c files)
  - Note the last line before `(gdb)` prompt:
    - if –g option is not used while compiling, you will see “`(no debugging symbols found)`”



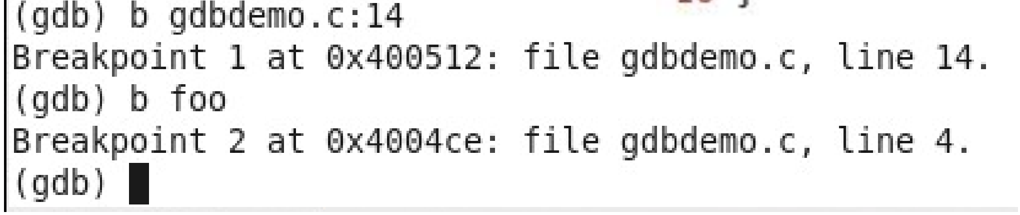
```
[ecegrid-thin4:~/ECE264] hegden$gdb gdbdemo
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-92.el6)
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/min/a/hegden/ECE264/gdbdemo...done.
(gdb)
```

GDB – Set breakpoints

- Set breakpoints (b)
  - At line 14
  - Beginning of `foo`



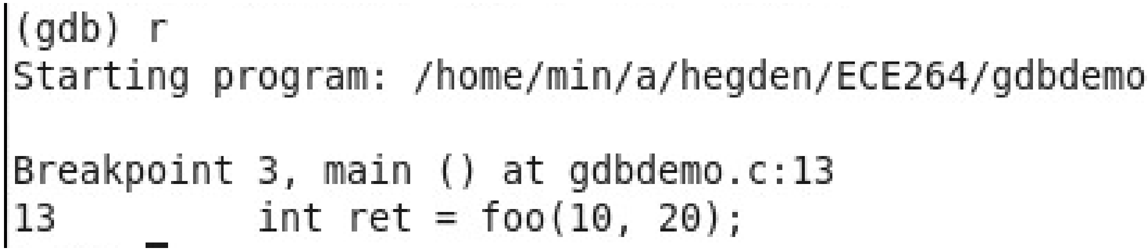
```
 1 #include<stdio.h>
 2 int foo(int a, int b)
 3 {
 4     int x = a + 1;
 5     int y = b + 2;
 6     int sum = x + y;
 7
 8     return x * y + sum;
 9 }
10
11 int main()
12 {
13    int ret = foo(10, 20);
14    printf("value returned from foo: %d\n",ret);
15    return 0;
16 }
```



```
(gdb) b gdbdemo.c:14
Breakpoint 1 at 0x400512: file gdbdemo.c, line 14.
(gdb) b foo
Breakpoint 2 at 0x4004ce: file gdbdemo.c, line 4.
(gdb) 
```

GDB – Start execution

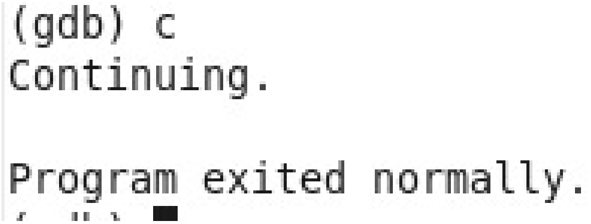
- Start execution (`r <command-line arguments>`)
  - Execution stops at the first breakpoint encountered



```
(gdb) r
Starting program: /home/min/a/hegden/ECE264/gdbdemo

Breakpoint 3, main () at gdbdemo.c:13
13	    int ret = foo(10, 20);
```

  - Continue execution (`c`)

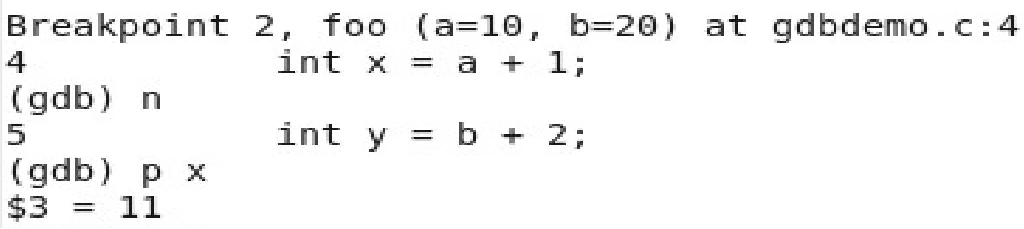


```
(gdb) c
Continuing.

Program exited normally.
(gdb) █
```

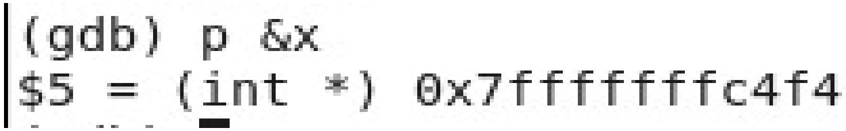
GDB – Printing

– Printing variable values (`p <variable_name>`)



```
Breakpoint 2, foo (a=10, b=20) at gdbdemo.c:4
4             int x = a + 1;
(gdb) n
5             int y = b + 2;
(gdb) p x
$3 = 11
```

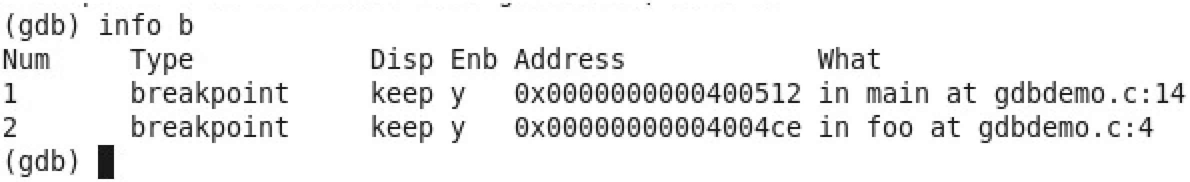
– Printing addresses (`p &<variable_name>`)



```
(gdb) p &x
$5 = (int *) 0x7fffffffc4f4
```

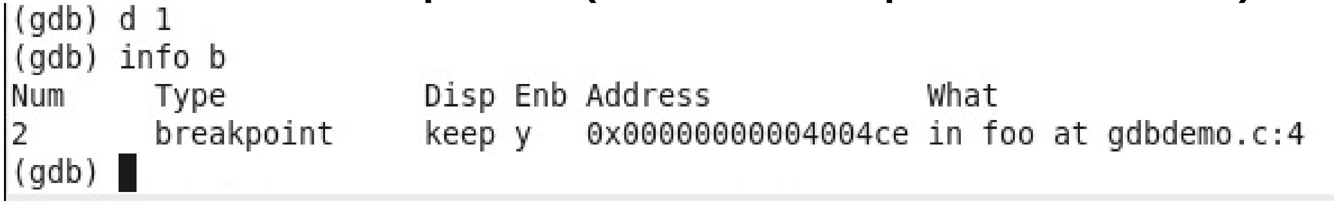
GDB – Manage breakpoints

- Display all breakpoints set (`info b`)



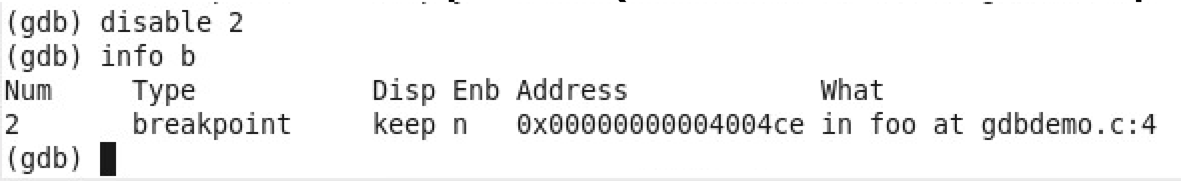
```
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000400512 in main at gdbdemo.c:14
2       breakpoint     keep y   0x00000000004004ce in foo at gdbdemo.c:4
(gdb) █
```

- Delete a breakpoint (`d <breakpoint num>`)



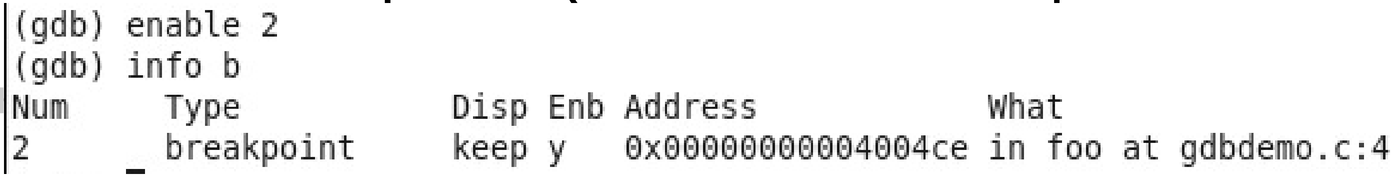
```
(gdb) d 1
(gdb) info b
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x00000000004004ce in foo at gdbdemo.c:4
(gdb) █
```

- Disable a breakpoint (`disable <breakpoint num>`)



```
(gdb) disable 2
(gdb) info b
Num     Type           Disp Enb Address            What
2       breakpoint     keep n   0x00000000004004ce in foo at gdbdemo.c:4
(gdb) █
```

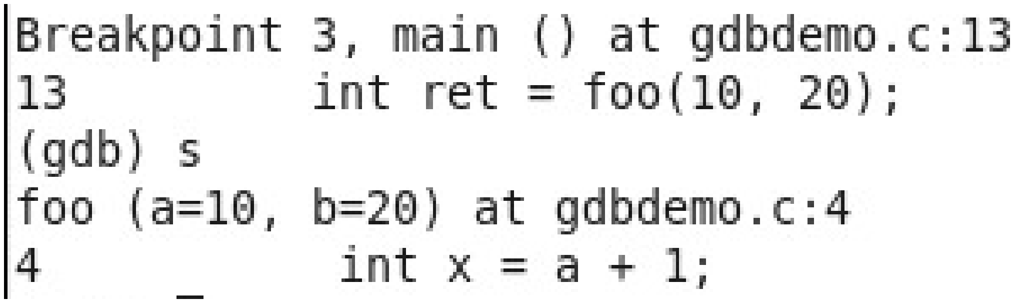
- Enable breakpoint (`enable <breakpoint num>`)



```
(gdb) enable 2
(gdb) info b
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x00000000004004ce in foo at gdbdemo.c:4
```

GDB – Step in

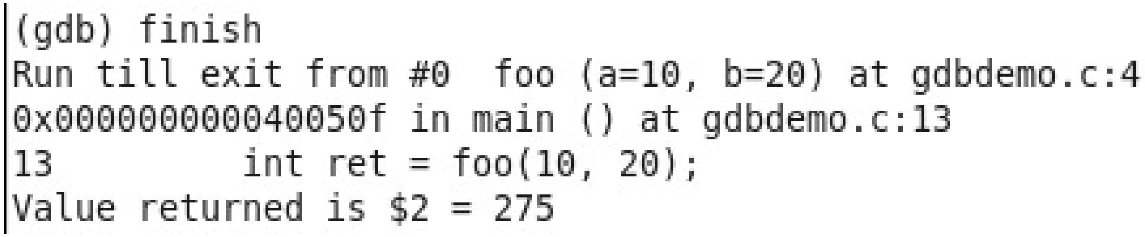
– Steps inside a function call (`s`)



```
Breakpoint 3, main () at gdbdemo.c:13
13         int ret = foo(10, 20);
(gdb) s
foo (a=10, b=20) at gdbdemo.c:4
4           int x = a + 1;
```

GDB – Step out

– Jump to return address (`finish`)

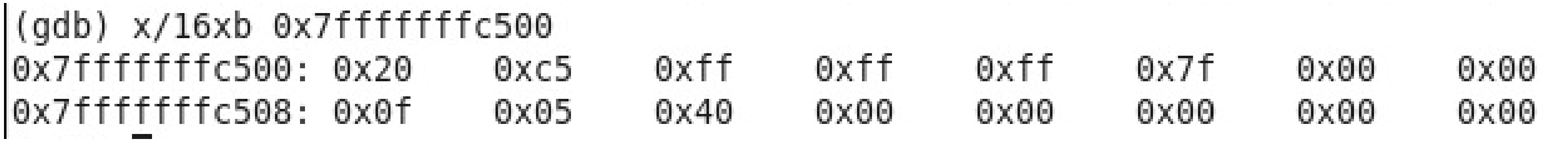


```
(gdb) finish
Run till exit from #0  foo (a=10, b=20) at gdbdemo.c:4
0x000000000040050f in main () at gdbdemo.c:13
13          int ret = foo(10, 20);
Value returned is $2 = 275
```

GDB – Memory dump

– Printing memory content (`x/nfu <address>`)

- n = repetition (number of bytes to display)
- f = format (‘x’ – hexadecimal, ‘d’-decimal, etc.)
- u = unit (‘b’ – byte, ‘h’ – halfword/2 bytes, ‘w’ – word/4 bytes, ‘g’ – giga word/8 bytes)
- `E.g. x/16xb 0x7fffffffc500` (display the values of 16 bytes stored from starting address

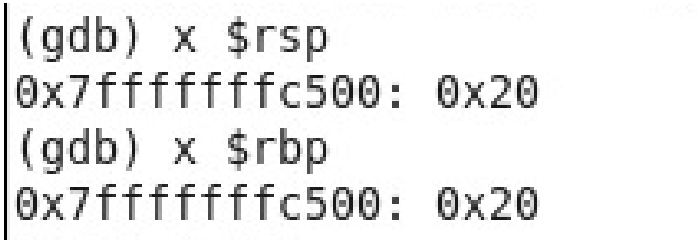


```
(gdb) x/16xb 0x7fffffffc500
0x7fffffffc500: 0x20    0xc5    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffc508: 0x0f    0x05    0x40    0x00    0x00    0x00    0x00    0x00
```

GDB – Printing addresses

– Registers ($rsp, $rbp)

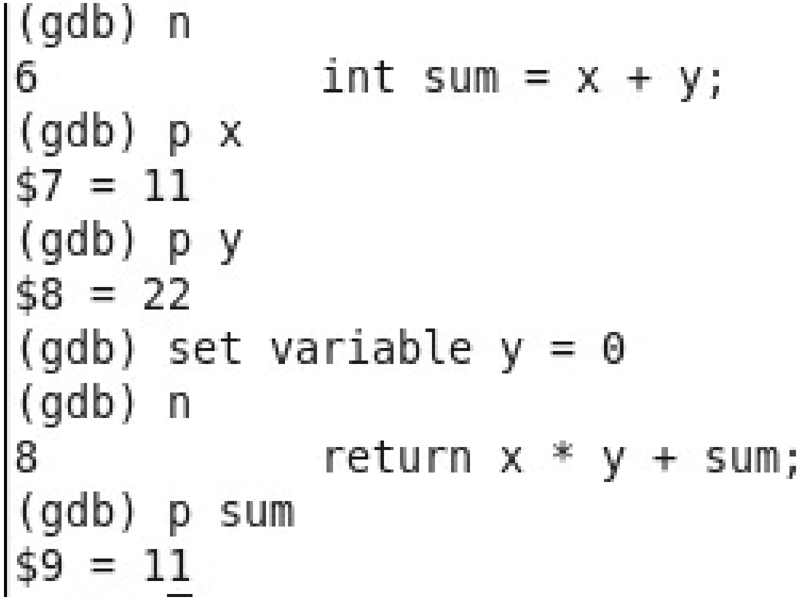
- Note that we use the ‘x’ command and not the ‘p’ command.



```
(gdb) x $rsp
0x7fffffffc500: 0x20
(gdb) x $rbp
0x7fffffffc500: 0x20
```

GDB – Altering memory content

- Set command (`set variable <name> = value`)



```
(gdb) n
6           int sum = x + y;
(gdb) p x
$7 = 11
(gdb) p y
$8 = 22
(gdb) set variable y = 0
(gdb) n
8           return x * y + sum;
(gdb) p sum
$9 = 11
```

- Set command (`set *(<type *>addr) = value`)

GDB Demo

- Refer to the demo example

GNU gprof

Valgrind