Intermediate Programming

Day 9

Outline

- Exercise 8
- Multidimensional arrays
- gdb
- Review questions

Exercise 8

- Implement the concat function

all_in_one.c

```
…
int concat( const char word1[] , const char word2[] , char result[] , int result_capacity )
{
	int len1 = strlen(word1) , len2 = strlen(word2);
	if( len1+len2>=result_capacity ) return 1;
	for( int i=0 ; i<len1 ; i++ ) result[i] = word1[i];
	for( int i=0 ; i<len2 ; i++ ) result[i+len1] = word2[i];
	result[len1+len2] = 0;
	return 0;
}
…
```

Exercise 8

- Separate into three files

string_functions.c

```
#include <string.h>
#include <string_functions.h> // Don't really need, but…
int concat( const char word1[] , const char word2[] , char result[] , int result_capacity )
{
	int len1 = strlen(word1) , len2 = strlen(word2);
	if( len1+len2>=result_capacity ) return 1;
	for( int i=0 ; i<len1 ; i++ ) result[i] = word1[i];
	for( int i=0 ; i<len2 ; i++ ) result[i+len1] = word2[i];
	result[len1+len2] = 0;
	return 0;
}
```

string_functions.h

```
#ifndef STRING_FUNCTIONS_H
#define STRING_FUNCTIONS_H
int concat( const char [] , const char [] , char [] , int );
#endif // STRING_FUNCTIONS_H
```

run_concat.c

```
#include <stdio.h>
#include <string.h>
#include "string_functions.h"

int main() {
	…
}
```

Exercise 8

- Separate into three files and compile manually

string_functions.c

```c
#include <string.h>
#include <string_functions.h> // Don't really need, but…
int concat( const char word1[] , const char word2[] , char result[] , int result_capacity )
{
	int len1 = strlen(word1) , len2 = strlen(word2);
	if( len1+len2>=result_capacity ) return 1;
	for( int i=0 ; i<len1 ; i++ ) result[i] = word1[i];
	for( int i=0 ; i<len2 ; i++ ) result[i+len1] = word2[i];
	result[len1+len2] = 0;
	return 0;
}
```

string_functions.h

```c
#ifndef STRING_FUNCTIONS_H
#define STRING_FUNCTIONS_H
int concat( const char [] , const char [] , char [] , int );
#endif // STRING_FUNCTIONS_H
```

run_concat.c

```c
#include <stdio.h>
#include <string.h>
#include "string_functions.h"
```

```
>> gcc -c string_functions.c -std=c99 -pedantic -Wall –Wextra
>> gcc -c run_concat.c -std=c99 -pedantic -Wall –Wextra
>> gcc -o run_concat run_concat.o string_functions.o
```

Exercise 8

- Create a `Makefile`

Makefile

```
CC=gcc
CFLAGS=-std=c99 -pedantic -Wall -Wextra

# Links together files needed to create executable
run:_concat run_concat.o string_functions.o
	$(CC) -o run_concat run_concat.o string_functions.o

# Compiles run_concat.c to create run_concat.o
# Note that we list string_functions.h here as a file that
# run_concat.c depends on, since run_concat.c #includes it
run_concat.o: run_concat.c string_functions.h
	$(CC) $(CFLAGS) -c run_concat.c

# Compiles functions.c to create functions.o
# Note that we list string_functions.h here as a file that
# string_functions.c depends on, since string_functions.c #includes it
string_functions.o: string_functions.c string_functions.h
	$(CC) $(CFLAGS) -c string_functions.c

# Removes all object files and the executable named run_concat,
# so we can start fresh
clean:
	rm -f *.o run_concat
```

Exercise 8

- Create a `Makefile` and try it out

Makefile

```
CC=gcc
CFLAGS=-std=c99 -pedantic -Wall -Wextra

# Links together files needed to create executable
run:_concat run_concat.o string_functions.o
	$(CC) -o run_concat run_concat.o string_functions.o

# Compiles run_concat.c to create run_concat.o
# Note that we list string_functions.h here as a file that
# run_concat.c depends on, since run_concat.c #includes it
run_concat.o: run_concat.c string_functions.h
	$(CC) $(CFLAGS) -c run_concat.c

# Compiles functions.c to create functions.o
# Note that we list string_functions.h here as a file that
# string_functions.c depends on, since string_functions.c #includes it
string_functions.o: string_functions.c string_functions.h
	$(CC) $(CFLAGS) -c string_functions.c

# Removes all object files and the executable named run_concat
```

```
>> make clean
rm -f *.o run_concat
>> make
gcc -std=c99 -pedantic -Wall -Wextra -c run_concat.c
gcc -std=c99 -pedantic -Wall -Wextra -c string_functions.c
gcc -o run_concat run_concat.o string_functions.o
```

Exercise 8

- Create a `Makefile` and try it out

Makefile

```
CC=gcc
CFLAGS=-std=c99 -pedantic -Wall -Wextra

# Links together files needed to create executable
run:_concat run_concat.o string_functions.o
	$(CC) -o run_concat run_concat.o string_functions.o

# Compiles run_concat.c to create run_concat.o
# Note that we list string_functions.h here as a file that
# run_concat.c depends on, since run_concat.c #includes it
run_concat.o: run_concat.c string_functions.h
	$(CC) $(CFLAGS) -c run_concat.c

# Compiles functions.c to create functions.o
# Note that we list string_functions.h here as a file that
# string_functions.c depends on, since string_functions.c #includes it
string_functions.o: string_functions.c string_functions.h
	$(CC) $(CFLAGS) -c string_functions.c

# Removes all object files and the executable named run_concat,
# so we can start fresh
```

```
>> touch string_functions.h
>> make
gcc -std=c99 -pedantic -Wall -Wextra -c run_concat.c
gcc -std=c99 -pedantic -Wall -Wextra -c string_functions.c
gcc -o run_concat run_concat.o string_functions.o
```

Exercise 8

- Create a `Makefile` and try it out

Makefile

```
CC=gcc
CFLAGS=-std=c99 -pedantic -Wall -Wextra

# Links together files needed to create executable
run:_concat run_concat.o string_functions.o
	$(CC) -o run_concat run_concat.o string_functions.o

# Compiles run_concat.c to create run_concat.o
# Note that we list string_functions.h here as a file that
# run_concat.c depends on, since run_concat.c #includes it
run_concat.o: run_concat.c string_functions.h
	$(CC) $(CFLAGS) -c run_concat.c

# Compiles functions.c to create functions.o
# Note that we list string_functions.h here as a file that
# string_functions.c depends on, since string_functions.c #includes it
string_functions.o: string_functions.c string_functions.h
	$(CC) $(CFLAGS) -c string_functions.c

# Removes all object files and the executable named run_concat,
# so we can start fresh
```

```
>> touch string_functions.c
>> make
gcc -std=c99 -pedantic -Wall -Wextra -c string_functions.c
gcc -o run_concat run_concat.o string_functions.o
```

Outline

- Exercise 8
- Multidimensional arrays
- gdb
- Review questions

2-dimensional arrays

Q: How do we declare a 2x3 grid of values?

Q: What is a 2x3 grid of values?

A: This is an array (of size 2) containing arrays (of size 3), storing a total of 6 values

2-dimensional arrays

Q: How do we declare a 2x3 grid of values?

A: We can declare a single array of integers and use row-major indexing

```
#include <stdio.h>
int main( void )
{
    int a2[2*3];
    for( int i=0 ; i<2 ; i++ )
        for( int j=0 ; j<3 ; j++ )
            a2[3*i+j] = i+j;
    return 0;
}
```

2-dimensional arrays

Q: How do we declare a 2x3 grid of values?

A: We can declare a single array of integers and use row-major indexing

- ✘ Indexing is ugly

```
#include <stdio.h>
int main( void )
{
    int a2[2*3];
    for( int i=0 ; i<2 ; i++ )
        for( int j=0 ; j<3 ; j++ )
            a2[3*i+j] = i+j;
    return 0;
}
```

2-dimensional arrays

Q: How do we declare a 2x3 grid of values?

A: We can declare a single array of integers and use row-major indexing

- ✗ Indexing is ugly

A: We can declare as a 2D array and let the compiler handle indexing

```
#include <stdio.h>
int main( void )
{
    int a2[2][3];
    for( int i=0 ; i<2 ; i++ )
        for( int j=0 ; j<3 ; j++ )
            a2[i][j] = i+j;
    return 0;
}
```

2-dimensional arrays

Q: How do we declare a 2x3 grid of values?

A: We can declare a single array of integers and use row-major indexing

- ✗ Indexing is ugly

A: We can declare as a 2D array and let the compiler handle indexing

- ✓ Indexing is clean

```
#include <stdio.h>
int main( void )
{
    int a2[2][3];
    for( int i=0 ; i<2 ; i++ )
        for( int j=0 ; j<3 ; j++ )
            a2[i][j] = i+j;
    return 0;
}
```

2-dimensional arrays

Initialization:

As with 1D arrays, we can use braced initialization to set the values of the array when we declare it.

More deeply nested braces correspond to dimensions further to the right.

```
#include <stdio.h>
int main( void )
{
    int a2[2][3]= { {1,2,3} , {4,5,6} };
    for( int i=0 ; i<2 ; i++ )
        for( int j=0 ; j<3 ; j++ )
            printf( "%d\n" , a2[i][j] );
    return 0;
}
```

2-dimensional arrays

Initialization:

As with 1D arrays, we can use braced initialization to set the values of the array when we declare it.

More deeply nested braces correspond to dimensions further to the right.

```
#include <stdio.h>
int main( void )
{
    int a2[2][3] = { {1,2,3} , {4,5,6} };
    for( int i=0 ; i<2 ; i++ )
        for( int j=0 ; j<3 ; j++ )
            printf( "%d\n" , a2[i][j] );
    return 0;
}
```

2-dimensional arrays

Initialization:

As with 1D arrays, we can use braced initialization to set the values of the array when we declare it.

More deeply nested braces correspond to dimensions further to the right.

```c
#include <stdio.h>
int main( void )
{
    int a2[2][3] = { {1,2,3} , {4,5,6} };
    for( int i=0 ; i<2 ; i++ )
        for( int j=0 ; j<3 ; j++ )
            printf( "%d\n" , a2[i][j] );
    return 0;
}
```

```
>> gcc foo.c
>> ./a.out
1
2
3
4
5
6
>>
```

2-dimensional arrays

Initialization:

As with 1D arrays, we can use braced initialization to set the values of the array when we declare it.

More deeply nested braces correspond to dimensions further to the right.

We can omit the first size, since the compiler can deduce the number of interior arrays.

```
#include <stdio.h>
int main( void )
{
    int a2[][3] = { {1,2,3} , {4,5,6} };
    for( int i=0 ; i<2 ; i++ )
        for( int j=0 ; j<3 ; j++ )
            printf( "%d\n" , a2[i][j] );
    return 0;
}
```

```
>> gcc foo.c
>> ./a.out
1
2
3
4
5
6
>>
```

2-dimensional arrays

```
#include <stdio.h>
int main( void )
{
    int a2[][] = { {1,2,3} , {4,5,6} };
    for( int i=0 ; i<2 ; i++ )
        for( int j=0 ; j<3 ; j++ )
            printf( "%d\n" , a2[i][j] );
    return 0;
```

Initialization:

As with 1D arrays, we can use braced

```
>> gcc foo.c
foo.c: In function ‘main’:
foo.c:4:13: error: array type has incomplete element type ‘int[]’
     4 |         int a2[][] = { {1,2,3} , {4,5,6} };
       |             ^~
foo.c:4:13: note: declaration of ‘a2’ as multidimensional array must have bounds for all dimensions except the
first
>>
```

We can omit the first size, since the compiler can deduce the number of interior arrays.

But we have to provide the other dimension(s).

2-dimensional arrays

- We saw that you can pass 1D arrays to functions, though `sizeof` won't work

⇒ We need to pass the size as an argument.

```
#include <stdio.h>
void print1( const int a1[] , int sz )
{
    for( int i=0 ; i<sz ; i++ ) printf( "%d\n" , a1[i] );
}
int main( void )
{
    int a1[3] = { 1 , 2 , 3 };
    print1( a1 , 3 );
    return 0;
}
```

```
>> ./a.out
1
2
3
>>
```

2-dimensional arrays

- We can also pass 2D arrays, but again, `sizeof` won't work:

⇒ We need to pass the size of the base as an argument.

```
#include <stdio.h>
void print2( const int a2[][3] , int sz )
{
    for( int i=0 ; i<sz ; i++ ) for( int j=0 ; j<3 ; j++ )
        printf( "%d\n" , a2[i][j] );
}
int main( void )
{
    int a2[][3] = { { 1 , 2 , 3 } , { 4 , 5 , 6 } };
    print2( a2 , 2 );
    return 0;
}
```

```
>> ./a.out
1
2
3
4
5
6
>>
```

2-dimensional arrays

- We can also pass 2D arrays, but again, **sizeof** won't work:

⇒ We need to pass the size of the base as an argument.

⇒ We also need to specify the other size(s) when we describe the multi-dimensional array.

```
#include <stdio.h>
void print2( const int a2[][3] , int sz )
{
    for( int i=0 ; i<sz ; i++ ) for( int j=0 ; j<3 ; j++ )
        printf( "%d\n" , a2[i][j] );
}
int main( void )
{
    int a2[][3] = { { 1 , 2 , 3 } , { 4 , 5 , 6 } };
    print2( a2 , 2 );
    return 0;
}
```

```
>> ./a.out
1
2
3
4
5
6
>>
```

2-dimensional arrays

- We can also pass 2D arrays, but again, `sizeof` won't work:

⇒ We need to pass the size of the base as an argument.

⇒ We also need to specify the other size(s) when we describe the multi-dimensional array.

If the function takes a 1D array as an argument, we can pass that in as well.

```
#include <stdio.h>
void print1( const int a1[] , int sz )
{
    for( int i=0 ; i<sz ; i++ ) printf( "%d\n" , a1[i] );
}
int main( void )
{
    int a2[][3] = { { 1 , 2 , 3 } , { 4 , 5 , 6 } };
    print1( a2[0] , 3 );
    print1( a2[1] , 3 );
    return 0;
}
```

```
>> ./a.out
1
2
3
4
5
6
>>
```

Multi-dim. arrays

Multi-dimensional arrays work similarly.

```
#include <stdio.h>
void print3( const int a3[][3][2] , int sz )
{
    for( int i=0 ; i<sz ; i++ )
        for( int j=0 ; j<3 ; j++ )
            for( int k=0 ; k<2 ; k++ )
                printf( "%d\n" , a3[i][j][k] );
}
int main( void )
{
    int a3[][3][2] =    // int a3[4][3][2]
    {
        { { 1 , 2 } , { 3 , 4 } , { 5, 6} } ,
        { { 7 , 8 } , { 9 , 10 } , { 11 , 12 } } ,
        { { 13 , 14 } , { 15 , 16 } , { 17 , 18 } } ,
        { { 19 , 20 } , { 21 , 22 } , { 23 , 24 } } ,
    };
    print3( a3 , 4 );
    return 0;
}
```

Multi-dim. arrays

Defining multi-dim. arrays:

- When we use brace initialization we still need to specify all but the first dimensions.
- More deeply nested braces correspond to dimensions further to the right.

```
#include <stdio.h>
void print3( const int a3[][3][2] , int sz )
{
    for( int i=0 ; i<sz ; i++ )
        for( int j=0 ; j<3 ; j++ )
            for( int k=0 ; k<2 ; k++ )
                printf( "%d\n" , a3[i][j][k] );
}
int main( void )
{
    int a3[][3][2] =    // int a3[4][3][2]
    {
        { { 1 , 2 } , { 3 , 4 } , { 5, 6} } ,
        { { 7 , 8 } , { 9 , 10 } , { 11 , 12 } } ,
        { { 13 , 14 } , { 15 , 16 } , { 17 , 18 } } ,
        { { 19 , 20 } , { 21 , 22 } , { 23 , 24 } } ,
    };
    print3( a3 , 4 );
    return 0;
}
```

Multi-dim. arrays

Defining/declaring functions:

- When we define/declare a function taking a multi-dim. array, we need to specify all but the first dimensions (and we need to pass in the number of sub-arrays).

```
#include <stdio.h>
void print3( const int a3[][3][2] , int sz )
{
    for( int i=0 ; i<sz ; i++ )
        for( int j=0 ; j<3 ; j++ )
            for( int k=0 ; k<2 ; k++ )
                printf( "%d\n" , a3[i][j][k] );
}
int main( void )
{
    int a3[][3][2] =    // int a3[4][3][2]
    {
        { { 1 , 2 } , { 3 , 4 } , { 5, 6} } ,
        { { 7 , 8 } , { 9 , 10 } , { 11 , 12 } } ,
        { { 13 , 14 } , { 15 , 16 } , { 17 , 18 } } ,
        { { 19 , 20 } , { 21 , 22 } , { 23 , 24 } } ,
    };
    print3( a3 , 4 );
    return 0;
}
```

Multi-dim. arrays

Defining/declaring functions:

- When we define/declare a function taking a multi-dim. array, we need to specify all but the first dimensions (and we need to pass in the number of sub-arrays).
- And we can pass in the sub-arrays if we want

```
#include <stdio.h>
void print1( const int a1[] , int sz )
{
    for( int i=0 ; i<sz ; i++ ) printf( "%d\n" , a1[i] );
}
void print2( const int a2[][2] , int sz )
{
    for( int i=0 ; i<sz ; i++ ) print1( a2[i] , 2 );
}
int main( void )
{
    int a3[][3][2] =
    {
        { { 1 , 2 } , { 3 , 4 } , { 5, 6} } ,
        { { 7 , 8 } ,  { 9 , 10 } , { 11 , 12 } }
    };
    for( int i=0 ; i<2 ; i++ ) print2( a3[i] , 3 );
    return 0;
}
```

Multi-dim. arrays

Defining/declaring functions:

- When we define/declare a function taking a multi-dim. array, we need to specify all but the first dimensions (and we need to pass in the number of sub-arrays).
- And we can pass in the sub-arrays if we want

```
#include <stdio.h>
void print1( const int a1[] , int sz )
{
    for( int i=0 ; i<sz ; i++ ) printf( "%d\n" , a1[i] );
}
void print2( const int a2[][2] , int sz )
{
    for( int i=0 ; i<sz ; i++ ) print1( a2[i] , 2 );
}
int main( void )
{
    int a3[][3][2] =
    {
        { { 1 , 2 } , { 3 , 4 } ,
        { { 7 , 8 } , { 9 , 10 }
    };
    for( int i=0 ; i<2 ; i++ ) p
    return 0;
}
```

```
>> ./a.out
1
2
3
4
5
6
7
8
9
10
11
12
>>
```

Outline

- Exercise 8
- Multidimensional arrays
- gdb
- Review questions

Getting started with **gdb**

- The function **string_reverse** should reverse the contents of its argument (in place)…

```
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
    const int len = strlen(str);
    for( int i=0 ; i<len ; i++ )
    {
        str[i] = str[len-i-1];
        str[len-i-1] = str[i];
    }
}
int main( void )
{
    char reverse_me[] = "AAABBB";
    string_reverse( reverse_me );
    printf( "%s\n“ , reverse_me );
    return 0;
}
```

Getting started with **gdb**

- The function **string_reverse** should reverse the contents of its argument (in place)… but it doesn't.

```
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
    const int len = strlen(str);
    for( int i=0 ; i<len ; i++ )
    {
        str[i] = str[len-i-1];
        str[len-i-1] = str[i];
    }
}
int main( void )
{
    char reverse_me[] = "AAABBB";
    string_reverse( reverse_me );
    printf( "%s\n“ , reverse_me );
    return 0;
}
```

```
>> ./a.out
BBBBBB
>>
```

Getting started with **gdb**

- Use the `-g` flag to compile an executable `a.out` which can be used by gdb

```
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
        const int len = strlen(str);
        for( int i=0 ; i<len ; i++ )
        {
                str[i] = str[len-i-1];
                str[len-i-1] = str[i];
        }
}
int main( void )
{
        char reverse_me[] = "AAABBB";
        string_reverse( reverse_me );
        printf( "%s\n“ , reverse_me );
        return 0;
}
```

```
>> gcc -std=c99 -pedantic -Wall -Wextra -g reverseString.c
>>
```

Getting started with **gdb**

- Use the `-g` flag to compile an executable `a.out` which can be used by gdb
- Launch gdb to debug the program

```c
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
        const int len = strlen(str);
        for( int i=0 ; i<len ; i++ )
        {
                str[i] = str[len-i-1];
                str[len-i-1] = str[i];
        }
}
int main( void )
{
        char reverse_me[] = "AAABBB";
        string_reverse( reverse_me );
        printf( "%s\n“ , reverse_me );
        return 0;
}
```

```
>> gcc -std=c99 -pedantic -Wall -Wextra -g reverseString.c
>> gdb ./a.out
...
```

Getting started with **gdb**

- Use the `-g` flag to compile an executable `a.out` which can be used by gdb
- Launch gdb to debug the program
- This will put you in the gdb environment and you will have a new prompt where you can enter debugging commands

```c
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
	const int len = strlen(str);
	for( int i=0 ; i<len ; i++ )
	{
		str[i] = str[len-i-1];
		str[len-i-1] = str[i];
	}
}
int main( void )
{
	char reverse_me[] = "AAABBB";
	string_reverse( reverse_me );
	printf( "%s\n" , reverse_me );
	return 0;
}
```

```
>> gcc -std=c99 -pedantic -Wall -Wextra -g reverseString.c
>> gdb ./a.out
...
(gdb)
```

Getting started with **gdb**

- Use the `-g` flag to compile an executable `a.out` which can be used by gdb
- Launch gdb to debug the program
- This will put you in the gdb environment and you will have a new prompt where you can enter debugging commands
- When you're finished using gdb, exit by typing "`quit` " or just "q"

```
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
	const int len = strlen(str);
	for( int i=0 ; i<len ; i++ )
	{
		str[i] = str[len-i-1];
		str[len-i-1] = str[i];
	}
}
int main( void )
{
	char reverse_me[] = "AAABBB";
	string_reverse( reverse_me );
	printf( "%s\n" , reverse_me );
	return 0;
}
```

```
>> gcc -std=c99 -pedantic -Wall -Wextra -g reverseString.c
>> gdb ./a.out
...
(gdb) quit
>>
```

Getting started with **gdb**

- Within the debugger:
  - `list`:
    display source code with line numbers

```
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
    const int len = strlen(str);
    for( int i=0 ; i<len ; i++ )
    {
        str[i] = str[len-i-1];
        str[len-i-1] = str[i];
    }
```

```
(gdb) list
1       #include <stdio.h>
2       #include <string.h>
3       void string_reverse( char str[] )
4       {
5         const int len = strlen(str);
6         for( int i=0 ; i<len ; i++ )
7           {
8             str[i] = str[len-i-1];
9             str[len-i-1] = str[i];
10          }
```

Getting started with **gdb**

- Within the debugger:
  - `list`:
    display source code with line numbers
    - Type `list` again to get the next lines

```
1.   #include <stdio.h>
2.   #include <string.h>
3.   void string_reverse( char str[] )
4.   {
5.       const int len = strlen(str);
6.       for( int i=0 ; i<len ; i++ )
7.       {
8.           str[i] = str[len-i-1];
9.           str[len-i-1] = str[i];
10.      }
```

```
...
10 }
(gdb) list
11      }
12      int main( void )
13      {
14        char reverse_me[] = "AAABBB";
15        string_reverse( reverse_me );
16        printf( "%s\n" , reverse_me );
17      return 0;
18      }
(gdb)
```

Getting started with **gdb**

- Within the debugger:
  - `break <line number>`:
    add a breakpoint at the specified line number

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.          const int len = strlen(str);
6.          for( int i=0 ; i<len ; i++ )
7.          {
8.                  str[i] = str[len-i-1];
9.                  str[len-i-1] = str[i];
10.         }
11. }
12. int main( void )
13. {
14.         char reverse_me[] = "AAABBB";
15.         string_reverse( reverse_me );
16.         printf( "%s\n" , reverse_me );
17.         return 0;
18. }
```

```
(gdb) break 12
Breakpoint 1 at 0x4005ad: file stringReverse.c, line 12.
(gdb)
```

Getting started with **gdb**

- Within the debugger:
  - `break <line number>`:
    add a breakpoint at the specified line number
  - `break <function name>`
    add a breakpoint at the beginning of the function call

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.          const int len = strlen(str);
6.          for( int i=0 ; i<len ; i++ )
7.          {
8.                  str[i] = str[len-i-1];
9.                  str[len-i-1] = str[i];
10.         }
11. }
12. int main( void )
13. {
14.         char reverse_me[] = "AAABBB";
15.         string_reverse( reverse_me );
16.         printf( "%s\n" , reverse_me );
17.         return 0;
18. }
```

```
(gdb) break main
Breakpoint 1 at 0x4005ad: file stringReverse.c, line 14
(gdb)
```

```
>> gdb ./a.out
...
(gdb) break 15
Breakpoint 1, main () at temp.c:15
(gdb) run
...
Breakpoint 1, main () at foo.c:15
15         string_reverse( reverse_me );
(gdb)
```

db

add a breakpoint at the specified line number

- `break <function name>`
  add a breakpoint at the beginning of the function call
- `run <command line args>`:
  execute the program, with specified arguments, until a breakpoint is hit
  (the line of the breakpoint is not executed)

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.      const int len = strlen(str);
6.      for( int i=0 ; i<len ; i++ )
7.      {
8.          str[i] = str[len-i-1];
9.          str[len-i-1] = str[i];
10.     }
11. }
12. int main( void )
13. {
14.     char reverse_me[] = "AAABBB";
15.     string_reverse( reverse_me );
16.     printf( "%s\n" , reverse_me );
17.     return 0;
18. }
```

```
>> gdb ./a.out
...
(gdb) break 15
Breakpoint 1, main () at temp.c:15
(gdb) run
...
Breakpoint 1, main () at foo.c:15
15          string_reverse( reverse_me );
(gdb) print reverse_me
$1 = "AAABBB"
(gdb)
```

db

ne

number

- `break <function name>`
  add a breakpoint at the beginning of the function call
- `print [<format>] <expr>`: outputs the [formatted] value of the expression (often a single variable)*

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.      const int len = strlen(str);
6.      for( int i=0 ; i<len ; i++ )
7.      {
8.          str[i] = str[len-i-1];
9.          str[len-i-1] = str[i];
10.     }
11. }
12. int main( void )
13. {
14.     char reverse_me[] = "AAABBB";
15.     string_reverse( reverse_me );
16.     printf( "%s\n“ , reverse_me );
17.     return 0;
18. }
```

See, e.g., https://sourceware.org/gdb/onlinedocs/gdb/Output-Formats.html for format options.

```
>> gdb ./a.out
...
(gdb) break 15
Breakpoint 1, main () at temp.c:15
(gdb) run
...
Breakpoint 1, main () at foo.c:15
15          string_reverse( reverse_me );
(gdb) print reverse_me
$1 = "AAABBB"
(gdb) continue
Continuing.
BBBBBB
[Inferior 1 (process 9504) exited normally]
(gdb)
```

db

ne

of

the function call

- `print [<format>] <expr>`: outputs the [formatted] value of the expression (often a single variable)
- `continue`: resume execution until next breakpoint is reached

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.          const int len = strlen(str);
6.          for( int i=0 ; i<len ; i++ )
7.          {
8.                  str[i] = str[len-i-1];
9.                  str[len-i-1] = str[i];
10.         }
11. }
12. int main( void )
13. {
14.         char reverse_me[] = "AAABBB";
15.         string_reverse( reverse_me );
16.         printf( "%s\n" , reverse_me );
17.         return 0;
18. }
```

Getting started with **gdb**

- Managing breakpoints
  - `break <line number>`
  - `break <function name>`
    add a breakpoint

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.      const int len = strlen(str);
6.      for( int i=0 ; i<len ; i++ )
7.      {
8.          str[i] = str[len-i-1];
9.          str[len-i-1] = str[i];
10.     }
11. }
12. int main( void )
```

```
>> gdb ./a.out
...
(gdb) break main
Breakpoint 1 at 0x4005ad: file stringReverse.c, line 14.
(gdb) break 16
Breakpoint 2 at 0x4005ca: file stringReverse.c, line 16.
```

Getting started with **gdb**

- Managing breakpoints
  - `break <...>`
  - `delete <breakpoint num>`
    remove specified breakpoint

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char *str )
4.  {
5.      const int len = strlen(str);
6.      for( int i=0 ; i<len ; i++ )
7.      {
8.          str[i] = str[len-i-1];
9.          str[len-i-1] = str[i];
10.     }
11. }
12. int main( void )
```

```
>> gdb ./a.out
...
(gdb) break main
Breakpoint 1 at 0x4005ad: file stringReverse.c, line 14.
(gdb) break 16
Breakpoint 2 at 0x4005ca: file stringReverse.c, line 16.
(gdb) delete 1
No breakpoint number 1.
(gdb)
```

Getting started with **gdb**

- Managing breakpoints
  - `break <...>`
  - `delete <breakpoint num>`
  - `info breakpoints`
    display current list of breakpoints

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char *str )
4.  {
5.      const int len = strlen(str);
6.      for( int i=0 ; i<len ; i++ )
7.      {
8.          str[i] = str[len-i-1];
9.          str[len-i-1] = str[i];
10.     }
11. }
12. int main( void )
```

```
>> gdb ./a.out
...
(gdb) break main
Breakpoint 1 at 0x4005ad: file stringReverse.c, line 14.
(gdb) break 16
Breakpoint 2 at 0x4005ca: file stringReverse.c, line 16.
(gdb) delete 1
No breakpoint number 1.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x00000000004005ca in main at stringReverse.c:16
(gdb)
```

Getting started with **gdb**

- Managing breakpoints
  - `break <...>`
    - Can also set conditional breakpoints e.g., if we want to pause execution at line 8 only if variable i is bigger than 5

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char *str )
4.  {
5.      const int len = strlen(str);
6.      for( int i=0 ; i<len ; i++ )
7.      {
8.          str[i] = str[len-i-1];
9.          str[len-i-1] = str[i];
10.     }
11. }
12. int main( void )
13. {
14.     char reverse_me[] = "AAABBB";
15.     string_reverse( reverse_me );
16.     printf( "%s\n" , reverse_me );
17.     return 0;
```

```
>> gdb ./a.out
...
(gdb) break 8 if i>5
Breakpoint 1 at 0x40054b: file stringReverse.c, line 8.
(gdb)
```

Getting started with **gdb**

- Advancing the debugger:
  - `continue`: resume execution until next breakpoint is reached
  - `step`: single-steps execution forward, one source line at a time
  - `next`: single-steps execution forward, but treats function call as a single item
  - `finish`: runs until end of the current function

Getting started with **gdb**

- Other commands
  - `watch <variable>`: pauses program whenever this variable's value changes, and outputs its old and new values
  - `backtrace`: produces a stack trace of function calls that lead to a segmentation fault

Tips:

- Command names can be shortened as long as there is no ambiguity, e.g. type "`b 5` " to set a breakpoint at line 5, or "`p x` " to print the value of `x`
- Hitting return re-executes the last command

Getting started with **gdb**

- Let’s try to debug

```c
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
        const int len = strlen(str);
        for( int i=0 ; i<len ; i++ )
        {
                str[i] = str[len-i-1];
                str[len-i-1] = str[i];
        }
}
int main( void )
{
        char reverse_me[] = "AAABBB";
        string_reverse( reverse_me );
        printf( "%s\n" , reverse_me );
        return 0;
}
```

```
>> gdb ./a.out
...
(gdb)
```

Getting started with **gdb**

- Let’s try to debug
  - Add a breakpoint at the start

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.      const int len = strlen(str);
6.      for( int i=0 ; i<len ; i++ )
7.      {
8.          str[i] = str[len-i-1];
9.          str[len-i-1] = str[i];
10.     }
11. }
12. int main( void )
13. {
14.     char reverse_me[] = "AAABBB";
15.     string_reverse( reverse_me );
16.     printf( "%s\n" , reverse_me );
17.     return 0;
18. }
```

```
>> gdb ./a.out
...
(gdb) break main
Breakpoint 1 at 0x4005ad: file string_reverse.c, line 14.
(gdb)
```

Getting started with **gdb**

- Let’s try to debug
  - Add a breakpoint at the start
  - Start the execution

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.          const int len = strlen(str);
6.          for( int i=0 ; i<len ; i++ )
7.          {
8.                  str[i] = str[len-i-1];
9.                  str[len-i-1] = str[i];
10.         }
11. }
12. int main( void )
13. {
14.         char reverse_me[] = "AAABBB";
15.         string_reverse( reverse_me );
16.         printf( "%s\n" , reverse_me );
17.         return 0;
18. }
```

```
>> gdb ./a.out
...
(gdb) b main
Breakpoint 1 at 0x4005ad: file string_reverse.c, line 14.
(gdb) r
...
Breakpoint 1, main () at string_reverse.c :14
14        char reverse_me[] = "AAABBB";
(gdb)
```

Getting started with **gdb**

- Let’s try to debug
  - ...
  - Start the execution
  - Advance to the next line

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.          const int len = strlen(str);
6.          for( int i=0 ; i<len ; i++ )
7.          {
8.                  str[i] = str[len-i-1];
9.                  str[len-i-1] = str[i];
10.         }
11. }
12. int main( void )
13. {
14.         char reverse_me[] = "AAABBB";
15.         string_reverse( reverse_me );
16.         printf( "%s\n" , reverse_me );
17.         return 0;
18. }
```

```
>> gdb ./a.out
...
(gdb) r
...
Breakpoint 1, main () at string_reverse.c :14
14          char reverse_me[] = "AAABBB";
(gdb) n
15 string_reverse( reverse_me );
(gdb)
```

Getting started with **gdb**

- Let’s try to debug
  - …
  - Advance to the next line
  - Step into the function

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.          const int len = strlen(str);
6.          for( int i=0 ; i<len ; i++ )
7.          {
8.                  str[i] = str[len-i-1];
9.                  str[len-i-1] = str[i];
10.         }
11. }
12. int main( void )
13. {
14.         char reverse_me[] = "AAABBB";
15.         string_reverse( reverse_me );
16.         printf( "%s\n" , reverse_me );
17.         return 0;
18. }
```

```
>> gdb ./a.out
...
(gdb) n
15 string_reverse( reverse_me );
(gdb) s
string_reverse (str=0x7fffffffdef9 "AAABBB") at string_reverse.c :5
5         const int len = strlen(str);
(gdb)
```

Getting started with **gdb**

- Let’s try to debug
  - ...
  - Step into the function
  - Advance to the next line

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.      const int len = strlen(str);
6.      for( int i=0 ; i<len ; i++ )
7.      {
8.          str[i] = str[len-i-1];
9.          str[len-i-1] = str[i];
10.     }
11. }
12. int main( void )
13. {
14.     char reverse_me[] = "AAABBB";
15.     string_reverse( reverse_me );
16.     printf( "%s\n" , reverse_me );
17.     return 0;
18. }
```

```
>> gdb ./a.out
...
(gdb) s
string_reverse (str=0x7fffffffdef9 "AAABBB") at string_reverse.c :5
5           const int len = strlen(str);
(gdb) n
6           for( int i=0 ; i<len ; i++ )
(gdb)
```

Getting started with **gdb**

- Let's try to debug
  - ...
  - Advance to the next line
  - Print out len

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.          const int len = strlen(str);
6.          for( int i=0 ; i<len ; i++ )
7.          {
8.                  str[i] = str[len-i-1];
9.                  str[len-i-1] = str[i];
10.         }
11. }
12. int main( void )
13. {
14.         char reverse_me[] = "AAABBB";
15.         string_reverse( reverse_me );
16.         printf( "%s\n" , reverse_me );
17.         return 0;
18. }
```

```
>> gdb ./a.out
...
(gdb) n
6           for( int i=0 ; i<len ; i++ )
(gdb) p len
$1 = 6
(gdb)
```

Getting started with **gdb**

- Let’s try to debug
  - …
  - Print out len
  - Advance to the next line

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.          const int len = strlen(str);
6.          for( int i=0 ; i<len ; i++ )
7.          {
8.                  str[i] = str[len-i-1];
9.                  str[len-i-1] = str[i];
10.         }
11. }
12. int main( void )
13. {
14.         char reverse_me[] = "AAABBB";
15.         string_reverse( reverse_me );
16.         printf( "%s\n" , reverse_me );
17.         return 0;
18. }
```

```
>> gdb ./a.out
...
(gdb) p len
$1 = 6
(gdb) n
8               str[i] = str[len-i-1];
(gdb)
```

Getting started with **gdb**

- Let's try to debug
  - …
  - Advance to the next line
  - Print out i

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.          const int len = strlen(str);
6.          for( int i=0 ; i<len ; i++ )
7.          {
8.                  str[i] = str[len-i-1];
9.                  str[len-i-1] = str[i];
10.         }
11. }
12. int main( void )
13. {
14.         char reverse_me[] = "AAABBB";
15.         string_reverse( reverse_me );
16.         printf( "%s\n" , reverse_me );
17.         return 0;
18. }
```

```
>> gdb ./a.out
...
(gdb) n
8               str[i] = str[len-i-1];
(gdb) p i
$2 = 0
(gdb)
```

Getting started with **gdb**

- Let’s try to debug
  - ...
  - Print out i
  - Print out str[i]

```c
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
        const int len = strlen(str);
        for( int i=0 ; i<len ; i++ )
        {
                str[i] = str[len-i-1];
                str[len-i-1] = str[i];
        }
}
int main( void )
{
        char reverse_me[] = "AAABBB";
        string_reverse( reverse_me );
        printf( "%s\n" , reverse_me );
        return 0;
}
```

```
>> gdb ./a.out
...
(gdb) p i
$2 = 0
(gdb) p str[i]
$3 = 65 'A'
(gdb)
```

Getting started with **gdb**

- Let's try to debug
  - ...
  - Print out `str[i]`
  - Print out `str[len-i-1]`

```c
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
        const int len = strlen(str);
        for( int i=0 ; i<len ; i++ )
        {
                str[i] = str[len-i-1];
                str[len-i-1] = str[i];
        }
}
int main( void )
{
        char reverse_me[] = "AAABBB";
        string_reverse( reverse_me );
        printf( "%s\n" , reverse_me );
        return 0;
}
```

```
>> gdb ./a.out
...
(gdb) p str[i]
$3 = 65 'A'
(gdb) p str[len-i-1]
$4 = 66 'B'
(gdb)
```

Getting started with **gdb**

- Let’s try to debug
  - ...
  - Print out `str[len-i-1]`
  - Print out `str`

```c
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
        const int len = strlen(str);
        for( int i=0 ; i<len ; i++ )
        {
                str[i] = str[len-i-1];
                str[len-i-1] = str[i];
        }
}
int main( void )
{
        char reverse_me[] = "AAABBB";
        string_reverse( reverse_me );
        printf( "%s\n" , reverse_me );
        return 0;
}
```

```
>> gdb ./a.out
...
(gdb) p str[len-i-1]
$4 = 66 'B'
(gdb) p str
$5 = 0x7fffffffded9 "AAABBB"
(gdb)
```

Getting started with **gdb**

- Let’s try to debug
  - ...
  - Print out **str**
  - Advance to the next line

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.      const int len = strlen(str);
6.      for( int i=0 ; i<len ; i++ )
7.      {
8.          str[i] = str[len-i-1];
9.          str[len-i-1] = str[i];
10.     }
11. }
12. int main( void )
13. {
14.     char reverse_me[] = "AAABBB";
15.     string_reverse( reverse_me );
16.     printf( "%s\n" , reverse_me );
17.     return 0;
18. }
```

```
>> gdb ./a.out
...
(gdb) p str
$5 = 0x7fffffffded9 "AAABBB"
(gdb) n
9               str[len-i-1] = str[i];
(gdb)
```

Getting started with **gdb**

- Let’s try to debug
  - ...
  - Advance to the next line
  - Print out `str`

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  void string_reverse( char str[] )
4.  {
5.          const int len = strlen(str);
6.          for( int i=0 ; i<len ; i++ )
7.          {
8.                  str[i] = str[len-i-1];
9.                  str[len-i-1] = str[i];
10.         }
11. }
12. int main( void )
13. {
14.         char reverse_me[] = "AAABBB";
15.         string_reverse( reverse_me );
16.         printf( "%s\n" , reverse_me );
17.         return 0;
18. }
```

```
>> gdb ./a.out
...
(gdb) n
9               str[len-i-1] = str[i];
(gdb) p str
$6 = 0x7fffffffded9 “BAABBB"
(gdb)
```

Getting started with **gdb**

- Let's try to debug
  - ...
  - Print out **str**
  - Advance to the next line

```
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
        const int len = strlen(str);
        for( int i=0 ; i<len ; i++ )
        {
                str[i] = str[len-i-1];
                str[len-i-1] = str[i];
        }
}
int main( void )
{
        char reverse_me[] = "AAABBB";
        string_reverse( reverse_me );
        printf( "%s\n" , reverse_me );
        return 0;
}
```

```
>> gdb ./a.out
...
(gdb) p str
$6 = 0x7fffffffded9 "BAABBB"
(gdb) n
9           for( int i=0 ; i<len ; i++ )
(gdb)
```

Getting started with **gdb**

- Let’s try to debug
  - Print out str

We have over-written `str[i]` before we used it to set `str[len-i-1]`!!!

```
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
        const int len = strlen(str);
        for( int i=0 ; i<len ; i++ )
        {
                str[i] = str[len-i-1];
                str[len-i-1] = str[i];
        }
}
int main( void )
{
        char reverse_me[] = "AAABBB";
        string_reverse( reverse_me );
        printf( "%s\n" , reverse_me );
        return 0;
}
```

```
>> gdb ./a.out
...
(gdb) n
9         for( int i=0 ; i<len ; i++
(gdb) p str
$6 = 0x7fffffffded9 “BAABBB"
(gdb)
```

Getting started with **gdb**

- Let's try to debug

We have over-written str[i] before we used it to set str[len-i-1]!!!

Use a temporary variable

Code still doesn't work!

Use gdb to figure out why

```c
#include <stdio.h>
#include <string.h>
void string_reverse( char str[] )
{
        const int len = strlen(str);
        for( int i=0 ; i<len ; i++ )
        {
                char temp = str[i];
                str[i] = str[len-i-1];
                str[len-i-1] = temp;
        }
}
int main( void )
{
        char reverse_me[] = "AAABBB";
        string_reverse( reverse_me );
        printf( "%s\n" , reverse_me );
        return 0;
}
```

```
>> ./a.out
AAABBB
>>
```

Outline

- Exercise 8
- Multidimensional arrays
- gdb
- Review questions

Review questions

1. How do you declare a multi-dimensional array and pass it to a function?

Specify the dimensions with multiple brackets.

If using braced initialization, need to specify all but the first dimension.

When passing to a function need to specify all but the first dimension (and pass the first dimension as an argument).

Review questions

2. How do you initialize a multi-dimensional array using array initialization?

Nested braces (with nesting matching dimension ordering from left to right)

Review questions

3. What is the compile flag needed to compile a program such that we can debug it using gdb?

```
-g
```

Review questions

4. How do you set a break point using gdb and check the call stack?

Set a break point: `b <line num / function name>`

Check the call stack: `backtrace`

Review questions

5. Check the gdb cheat sheet and find the command to print the content of a variable per step, instead of only printing it once using `print`?

```
display
```

Exercise 9

- Website -> Course Materials -> Exercise 9