Intermediate Programming

Day 18

Outline

- Exercise 6-2
- Exercise 5-3
- Midterm project

Exercise 6-2 (part 2)

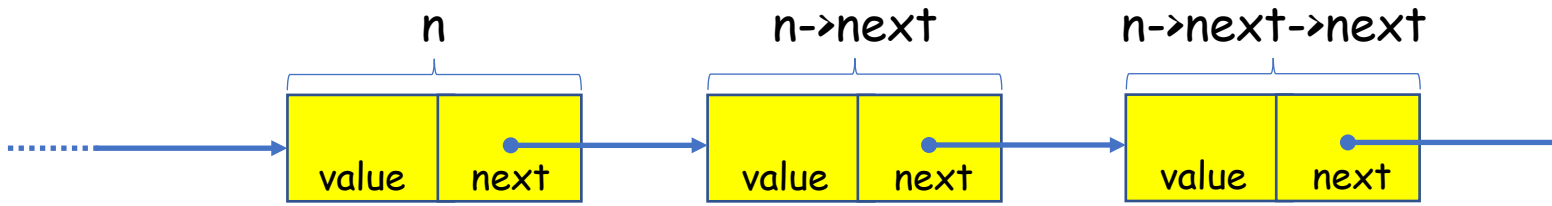
Implement the **remove_after** and **remove_front** functions.

```
char remove_after( Node *node )
{




}
```

Exercise 6-2 (part 2)

Implement the **remove_after** and **remove_front** functions.

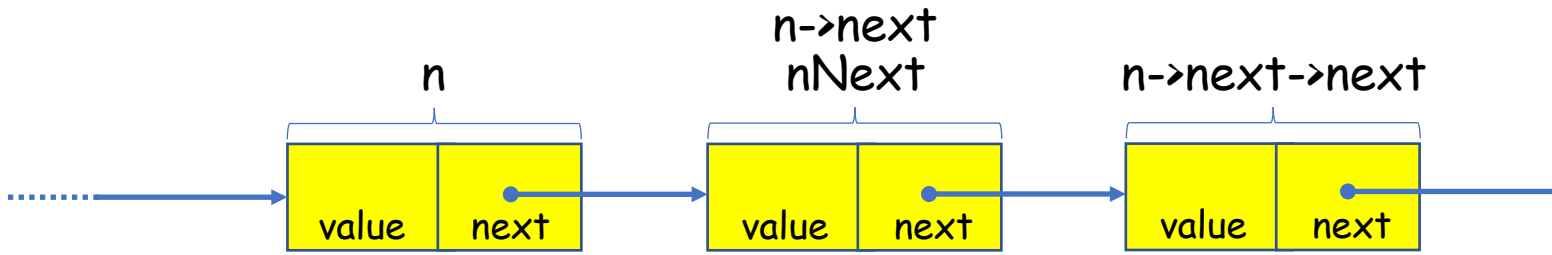
```
char remove_after( Node *node )
{




}
```



Exercise 6-2 (part 2)

Implement the **remove_after** and **remove_front** functions.

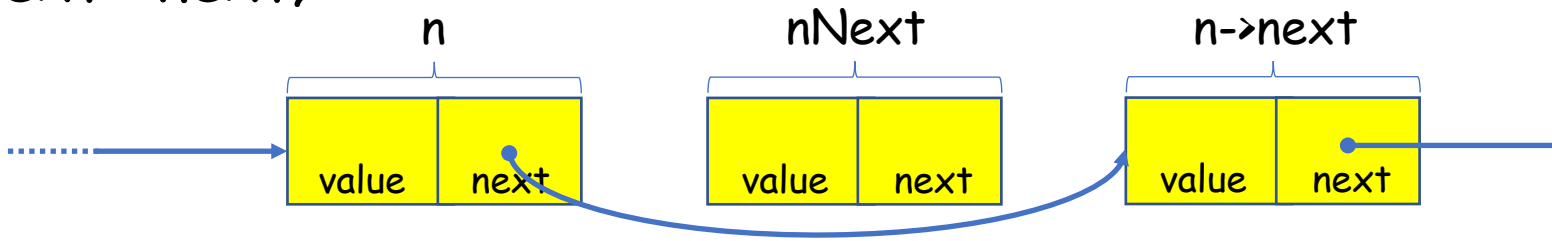
```
char remove_after( Node *node )
{
    Node *nNext = node->next;




}
```



Exercise 6-2 (part 2)

Implement the **remove_after** and **remove_front** functions.

```
char remove_after( Node *node )
{
    Node *nNext = node->next;
    if( !nNext ) return '?';
    char data = nNext->data;
    node->next = node->next->next;


}
```



Exercise 6-2 (part 2)

Implement the **remove_after** and **remove_front** functions.

```
char remove_after( Node *node )
{
    Node *nNext = node->next;
    if( !nNext ) return '?';
    char data = nNext->data;
    node->next = node->next->next;
    free( nNext );

}
```

Exercise 6-2 (part 2)

Implement the **remove_after** and **remove_front** functions.

```
char remove_after( Node *node )
{
    Node *nNext = node->next;
    if( !nNext ) return '?';
    char data = nNext->data;
    node->next = node->next->next;
    free( nNext );
    return data;
}
```

Exercise 6-2 (part 2)

Implement the **remove_after** and **remove_front** functions.

```
char remove_front( Node **list_ptr )
{




}
```

Exercise 6-2 (part 2)

Implement the **remove_after** and **remove_front** functions.

```
char remove_front( Node **list_ptr )
{




}
```

Exercise 6-2 (part 2)

Implement the **remove_after** and **remove_front** functions.

```
char remove_front( Node **list_ptr )
{
    Node* n = (*list_ptr);




}
```

Exercise 6-2 (part 2)

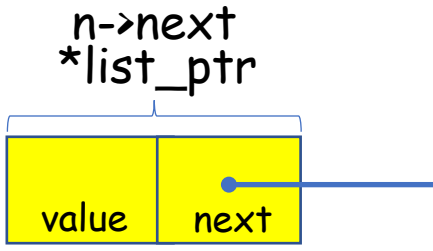
Implement the **remove_after** and **remove_front** functions.

```
char remove_front( Node **list_ptr )
{
    Node* n = (*list_ptr);
    if( !n ) return '?';
    char data = n->data;
    *list_ptr = n->next;



}
```

Exercise 6-2 (part 2)

Implement the **remove_after** and **remove_front** functions.

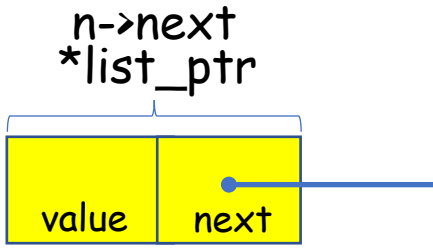
```
char remove_front( Node **list_ptr )
{
    Node* n = (*list_ptr);
    if( !n ) return '?';
    char data = n->data;
    *list_ptr = n->next;
    free( n );

}
```



Exercise 6-2 (part 2)

Implement the **remove_after** and **remove_front** functions.

```
char remove_front( Node **list_ptr )
{
    Node* n = (*list_ptr);
    if( !n ) return '?';
    char data = n->data;
    *list_ptr = n->next;
    free( n );
    return data;
}
```



Exercise 6-2 (part 4)

Implement the **remove_all** function.

```
void remove_all( Node **list_ptr , char val )
{
    while( (*list_ptr)->data==val ) remove_front( list_ptr );
    for( Node *n=*list_ptr ; n ; n=n->next )
        if( n->next && n->next->data==val )
            remove_after( n );
}
```

Exercise 6-2 (part 4)

Implement the **remove_all** function.

```
void remove_all( Node **list_ptr , char val )
{
    while( (*list_ptr)->data==val ) remove_front( list_ptr );
    for( Node *n=*list_ptr ; n ; n=n->next )
            while( n->next && n->next->data==val )
                    remove_after( n );
}
```

Exercise 6-2 (part 6)

Implement the **insert** function.

```
Node *insert( Node **list_ptr , char val )
{
    if( !*list_ptr )
    {
            *list_ptr  = create_node( val );
            return *list_ptr;
    }
    else if( val<(*list_ptr)->data )
    {
            add_front( list_ptr , val );
            return *list_ptr;
    }
    else
    {
            Node *n;
            for( n=*list_ptr ; n->next!=NULL && val>=n->next->data ; n=n->next ) ;
            add_after( n , val );
            return n->next;
    }
}
```

Outline

- Exercise 6-2
- Exercise 5-3
- Midterm project

Exercise 5-3 (part 3)

Implement the `int_magnitude` function… It should return an `unsigned int` value representing the *magnitude* of the argument value.

Recall:

- In binary, if a number is negative, it's most significant bit will be one.
- Two's complement: To get the expression for the negative of a number:
  - Flip the bits
  - Add one

Exercise 5-3 (part 3)

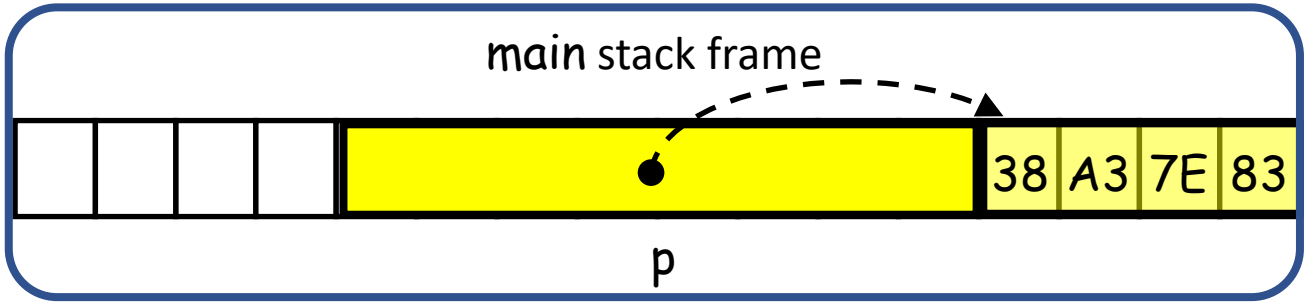
Implement the **int_magnitude** function… It should return an **unsigned int** value representing the *magnitude* of the argument value.

```
unsigned int magnitude( unsigned int value )
{
    unsigned int mask = 1<<31;
    if( !(value & mask ) ) return value;
    else                   return (~value) + 1;
}
```

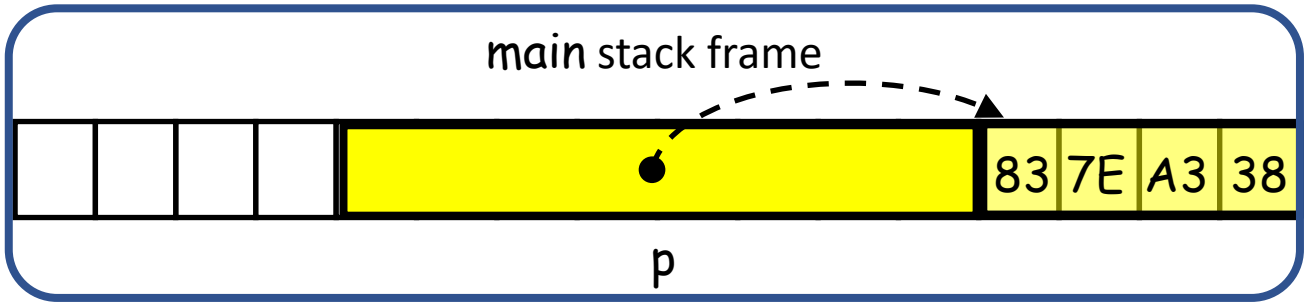
Exercise 5-3 (part 4)

```
#include <stdio.h>
int main( void )
{
    ...
    unsigned int val = 950238851u;
    unsigned int *p = &val;
    printf( "%u\n" , *p );
    return 0;
}
```

val … in hexadecimal is 38A37E83… in gdb … print each byte … is the computer you are running on … big endian or little endian?



or

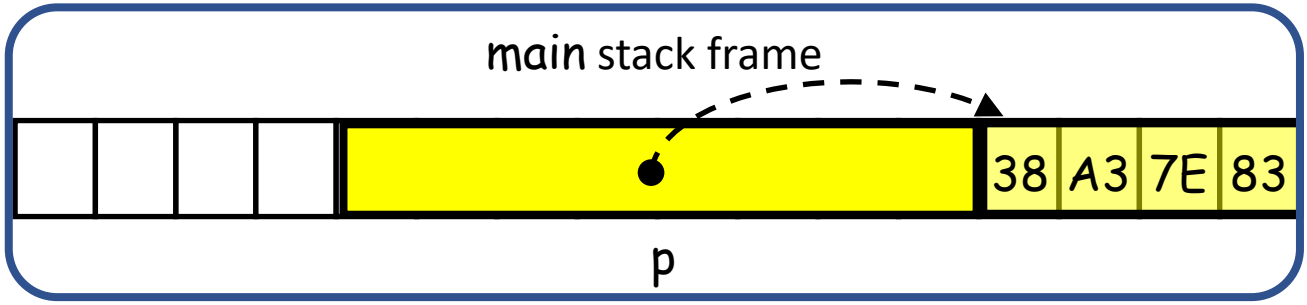


Exercise 5-3 (part 4)

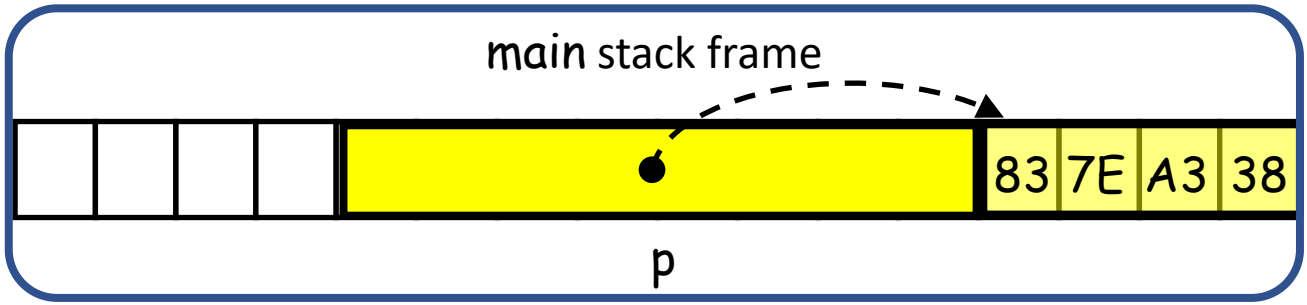
```
#include <stdio.h>
int main( void )
{
    ...
    unsigned int val = 950238851u;
    unsigned int *p = &val;
    printf( "%u\n" , *p );
    return 0;
}
```

val … in hexadecimal is 38A37E83… in gdb … print each byte … is the computer you are running on … big endian or little endian?

```
(gdb) b main
...
(gdb) r
...
(gdb) n
...
(gdb) n
21        printf("%u\n", *p);
```



or

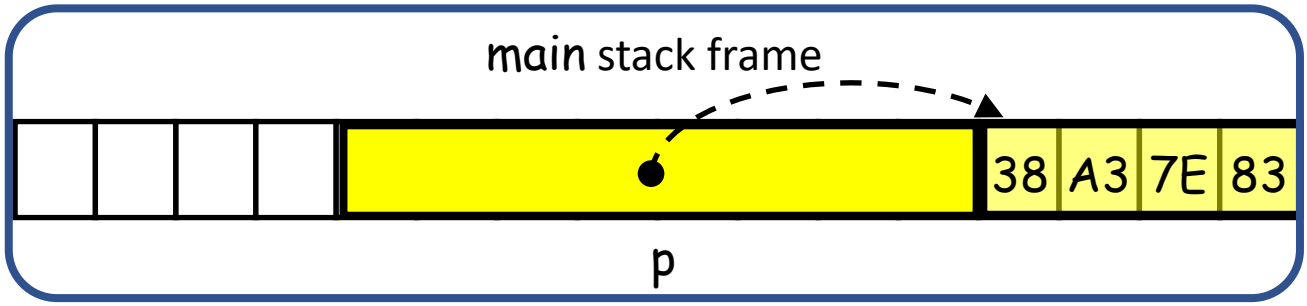


Exercise 5-3 (part 4)

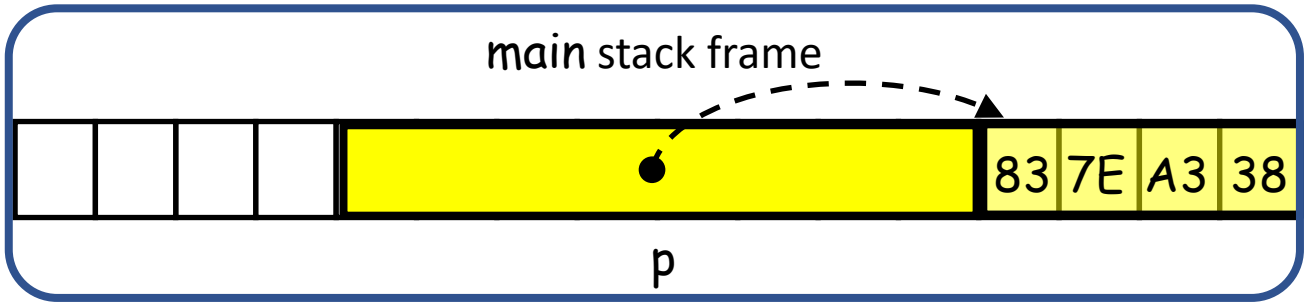
```
#include <stdio.h>
int main( void )
{
    ...
    unsigned int val = 950238851u;
    unsigned int *p = &val;
    printf( "%u\n" , *p );
    return 0;
}
```

val … in hexadecimal is 38A37E83… in gdb … print each byte … is the computer you are running on … big endian or little endian?

```
(gdb) p /x ((unsigned char*)p)[0]
$1 = 0x83
(gdb) p /x ((unsigned char*)p)[1]
$2 = 0x7e
(gdb) p /x ((unsigned char*)p)[2]
$3 = 0xa3
(gdb) p /x ((unsigned char*)p)[3]
$4 = 0x38
```



or



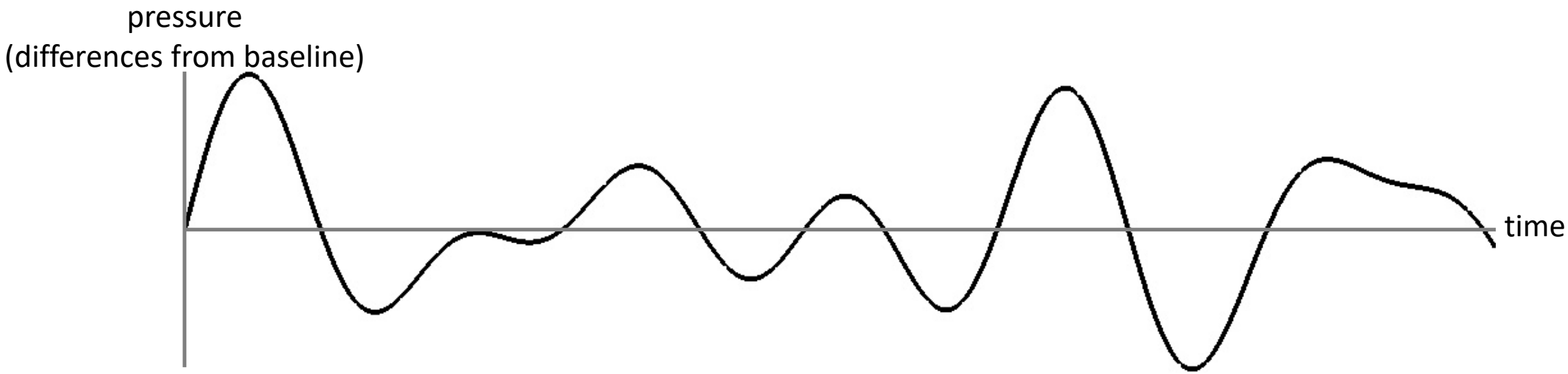
Outline

- Exercise 6-2
- Exercise 5-3
- Midterm project

Midterm project

Audio is represented as an oscillating function of time (seconds).

- This describes the disturbance in the ambient pressure

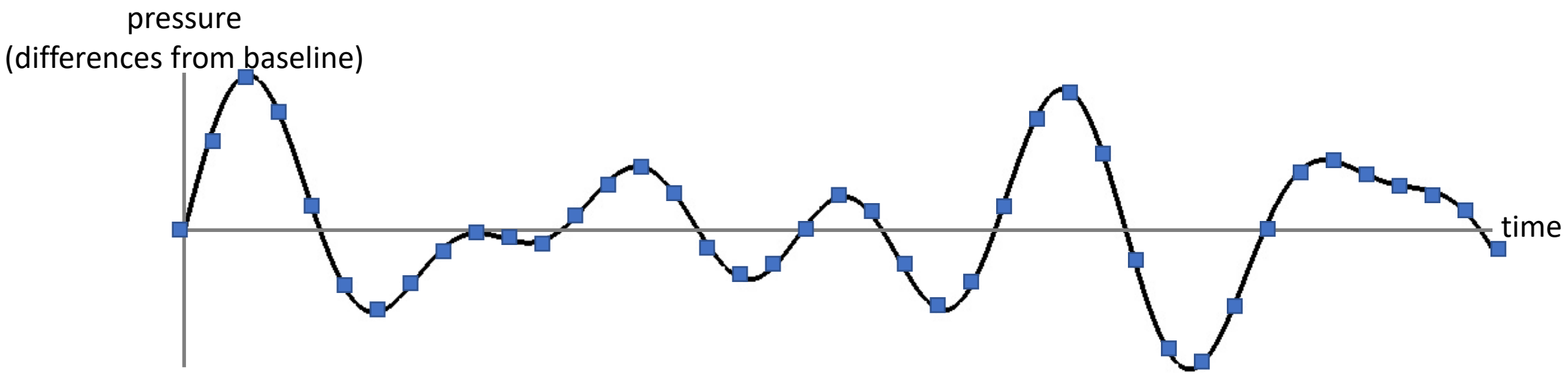


Midterm project

Audio is represented as an oscillating function of time (seconds).

To represent it on a computer, we represent it by discrete samples.

- The number of samples/second is Hertz
- In the assignment this is fixed at 44.1KHz



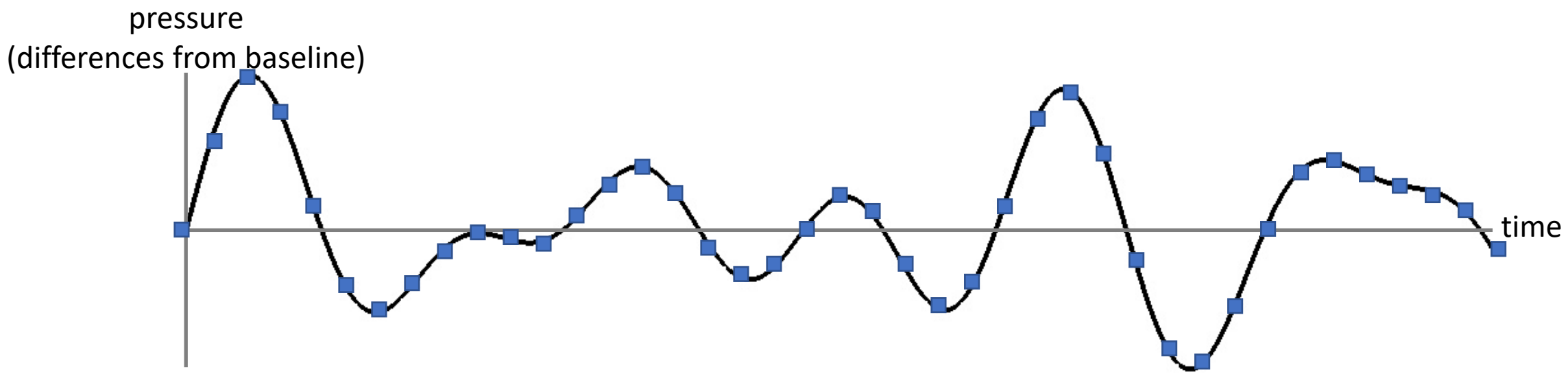
Midterm project

Audio is represented as an oscillating function of time (seconds).

To represent it on a computer, we represent it by discrete samples.

When we represent an audio signal in stereo, we represent (and sample) two signals, one for the left ear and one for the right.

- We need twice as many samples



Midterm project

Write out audio signals to `.wav` files, which includes:

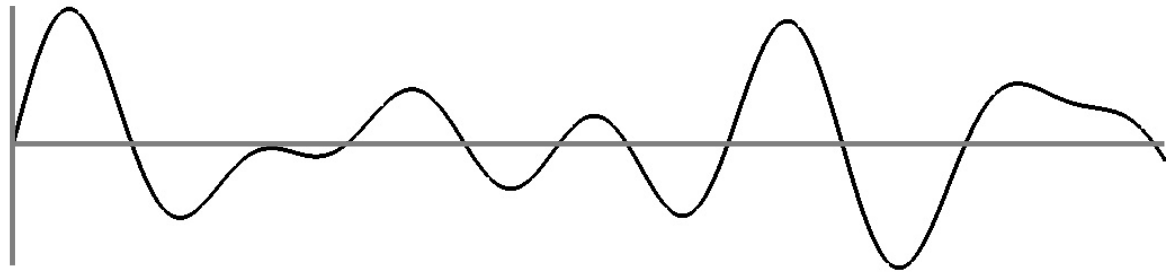
- Header:
  Describes the audio signal (number of samples, sampling rate=44.1KHz, number of channels=2, etc.)

```
duration = num_samples /441000;
```

- Audio content:
  The interleaved samples of the left and right signals (in binary)
  - Total number of samples is `2*num_samples`, with the even-indices sampling the signal for the left ear and the odd-indices sampling the right.
  - Each sample is represented with a 16-bit `int` type (`int16_t`).

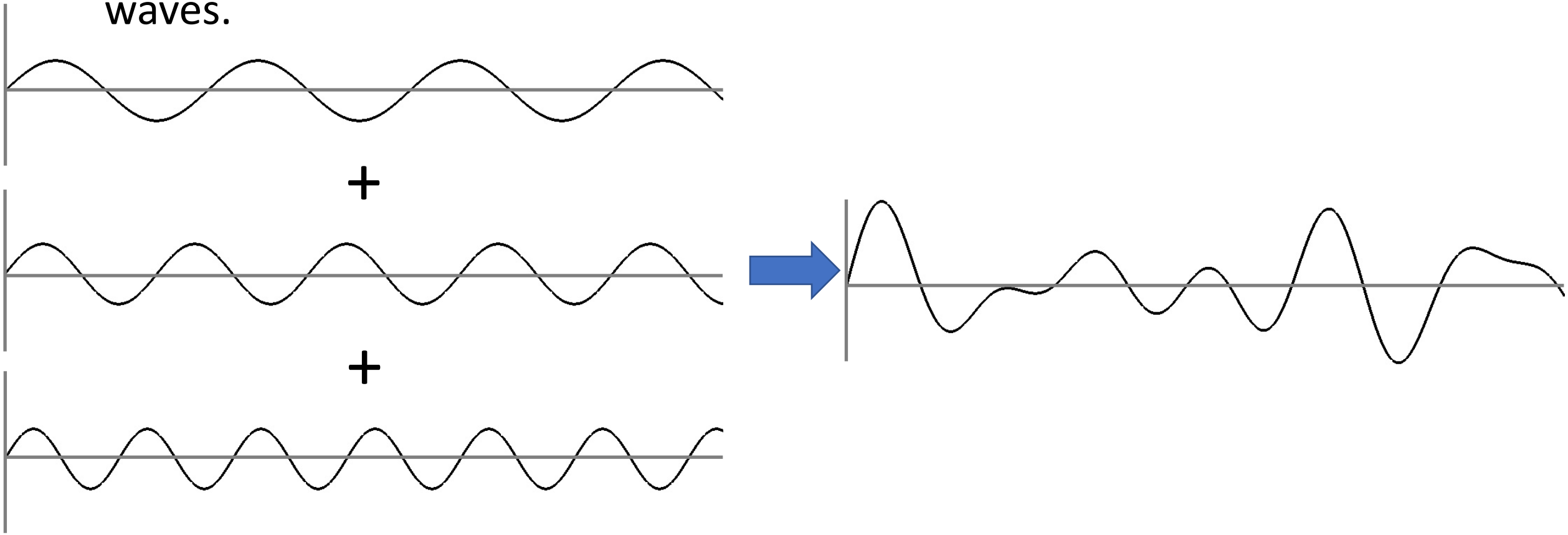
Midterm project

As the signal is (locally) oscillatory,



Midterm project

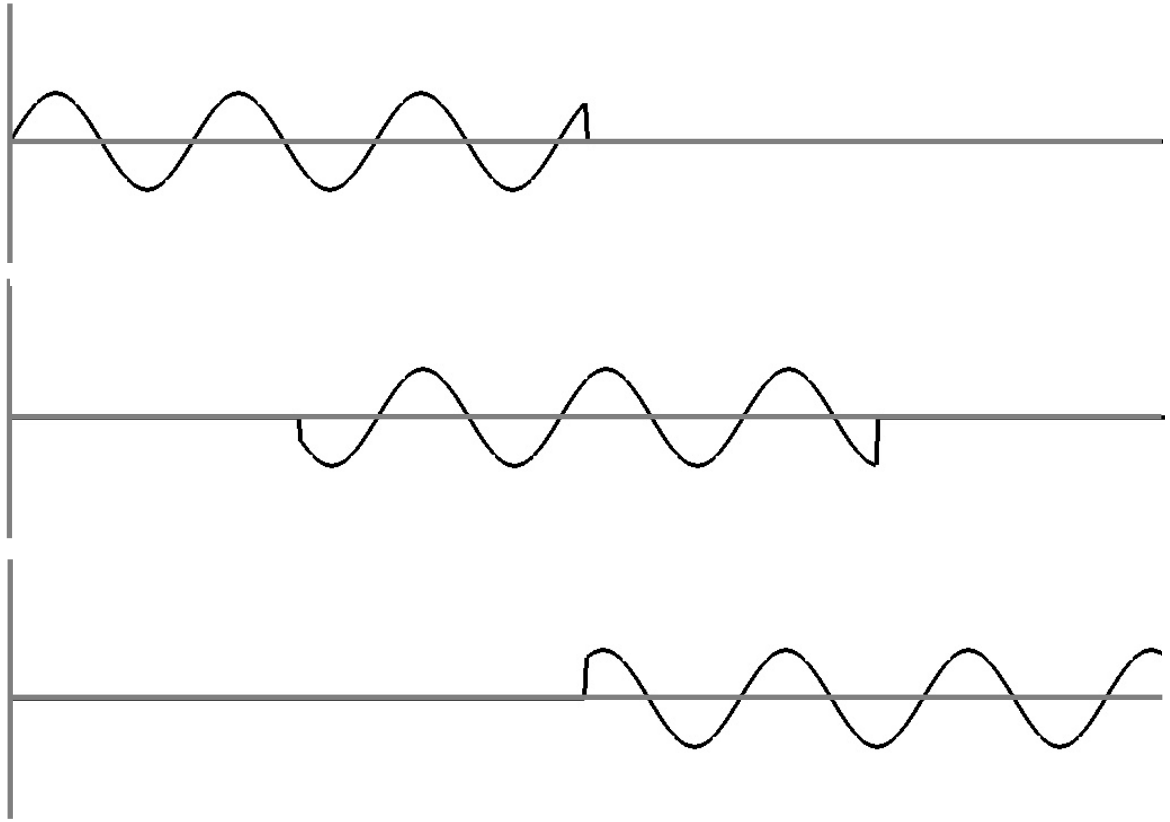
As the signal is (locally) oscillatory, we represent it as a sum of different waves.



Midterm project

The individual waves are represented in terms of their:

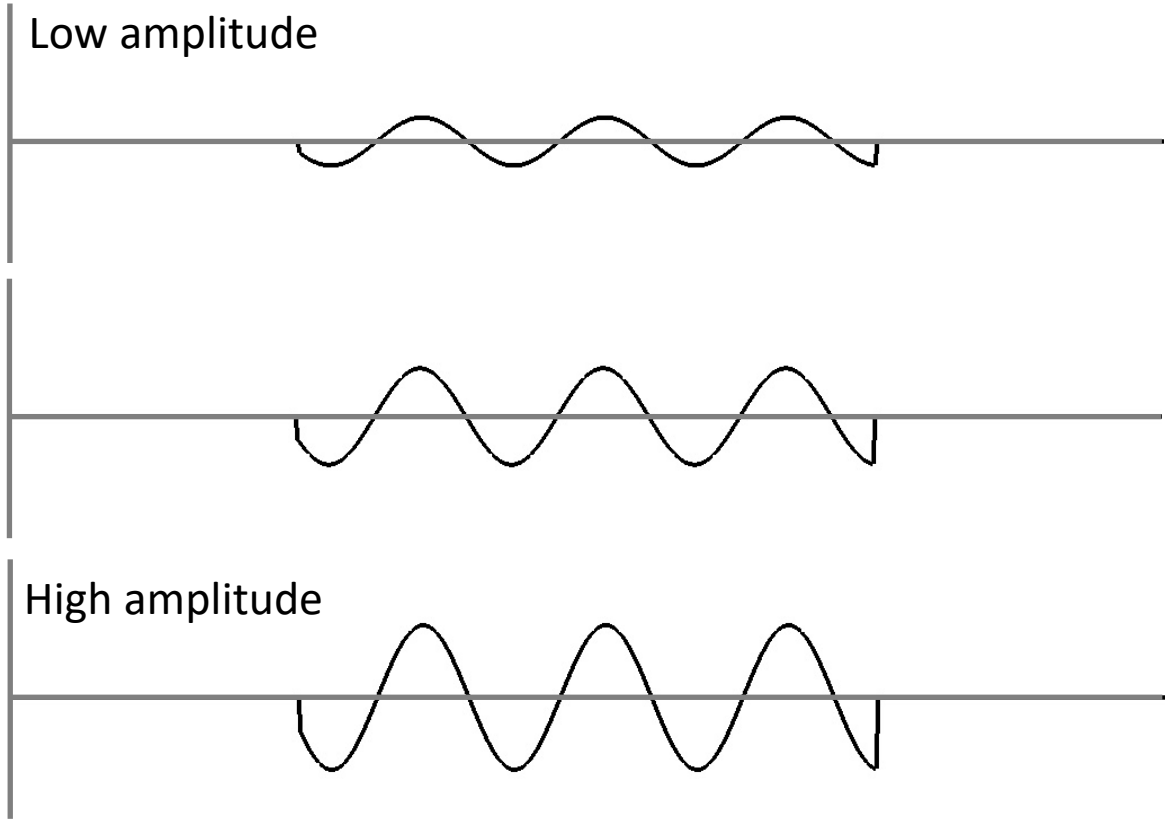
- **Temporal span**
  When is the sound played?



Midterm project

The individual waves are represented in terms of their:

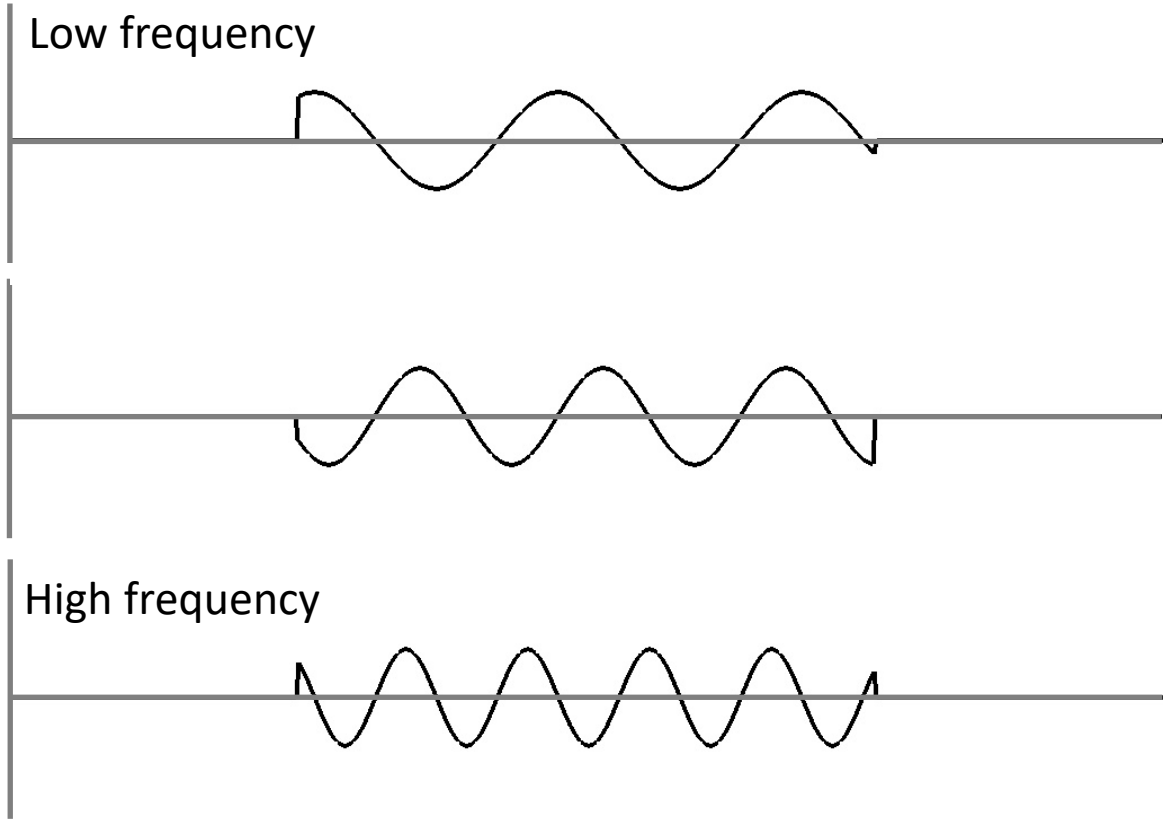
- Temporal span
- **Amplitude/gain**
  How loud is the sound?



Midterm project

The individual waves are represented in terms of their:

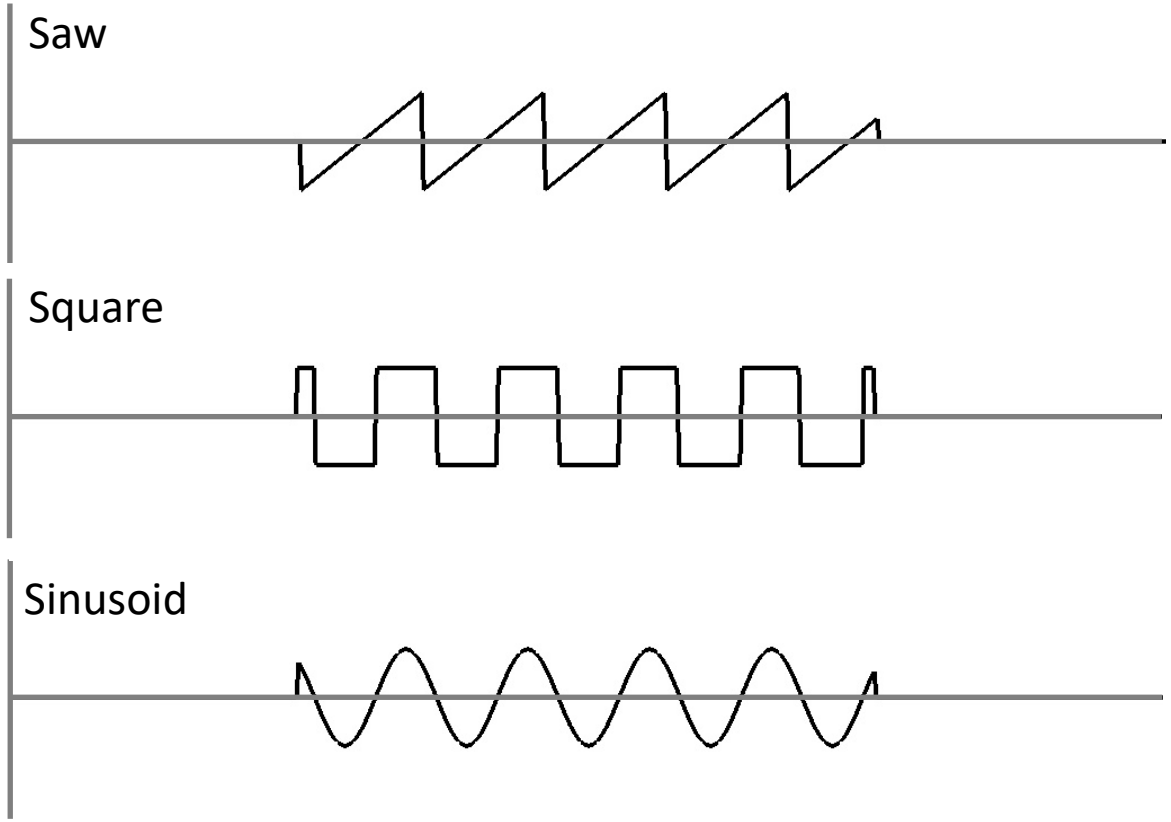
- Temporal span
- Amplitude/gain
- **Frequency**
  What is the pitch of the sound?



Midterm project

The individual waves are represented in terms of their:

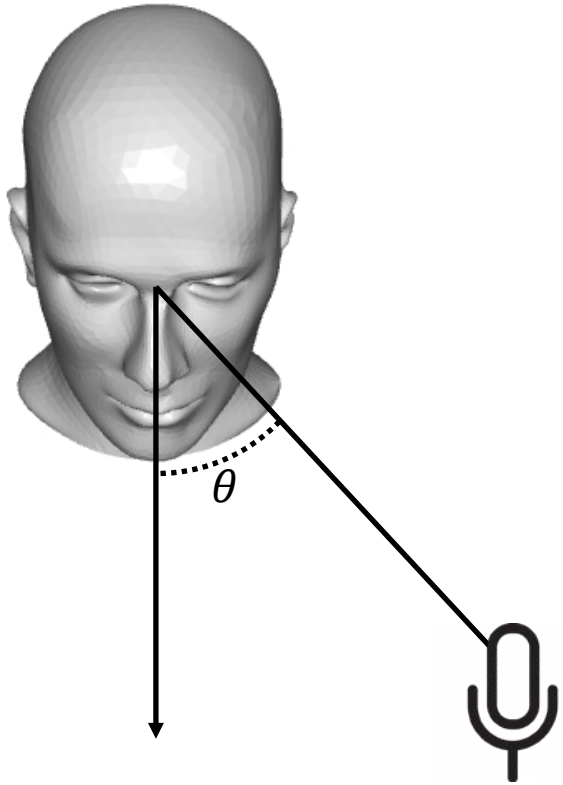
- Temporal span
- Amplitude/gain
- Frequency
- **Shape**
  What is the shape of the repeating part of the wave?

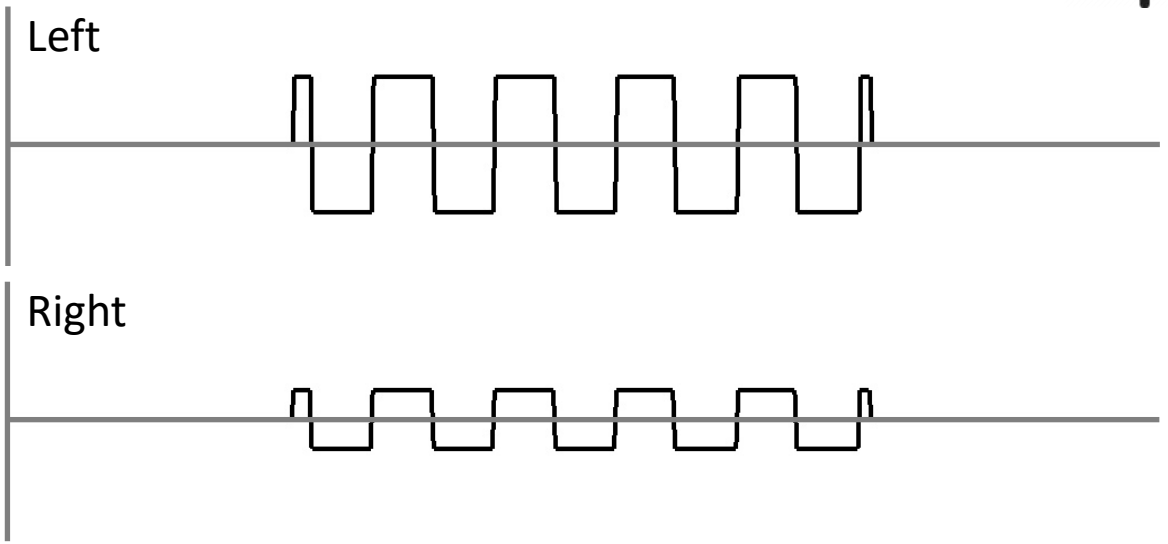


Midterm project

The individual waves are represented in terms of their:

- Temporal span
- Amplitude/gain
- Frequency
- Shape
- **Angle**
  How frontally aligned is the sound?



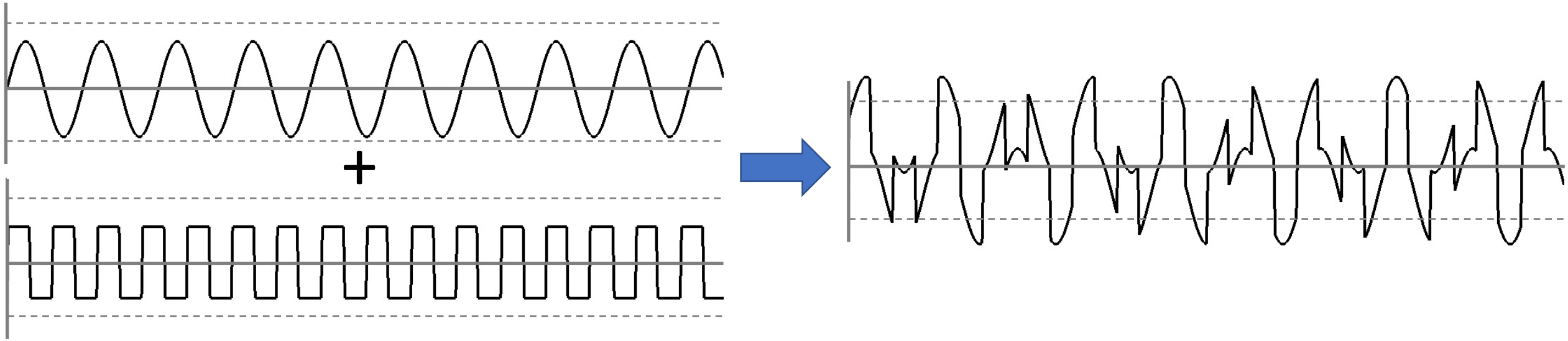


Midterm project

[WARNING]

When writing the signal to a `.wav` file, each sample of the signal is represented with a 16-bit int type (`int16_t`).

⇒ Even if the signal samples are within bounds, their sum may not be.



Midterm project

[WARNING]

When writing the signal to a `.wav` file, each sample of the signal is represented with a 16-bit int type (`int16_t`).

⇒ Even if the signal samples are within bounds, their sum may not be.

⇒ Be sure to clamp to avoid overflow happens!!!