Functions

Sai-Keung Wong

National Yang Ming Chiao Tung University

Hsinchu, Taiwan

Intended Learning Outcomes

- List some properties of functions
- Describe how to use vectors, characters, and strings
- List some math functions
- Describe the mechanism to write to a file or read from a file in the text mode.
- Implement functions to generate random numbers

Functions

Examples

Functions

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n)
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
}
```

Functions

- A function is a A1 code.

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n)
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
}
```

Functions

- A function is a block of code.
- It runs when A2

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n)
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
}
```

Functions

- A function is a block of code.
- It runs when it is called.
- A function may have A3

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n)
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
}
```

Functions

- A function is a block of code.
- It runs when it is called.
- A function may have parameters.
- It may return A4

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n)
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
}
```

Functions

- A function is a block of code.
- It runs when it is called.
- A function may have parameters.
- It may return a value.
- A function is designed to perform a A5

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n)
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
}
```

Functions

- A function is a block of code.
- It runs when it is called.
- A function may have parameters.
- It may return a value.
- A function is designed to perform a specific task.

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n)
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
}
```

Functions

- A function is **a block of code.**
- It runs when it is called.
- A function may have parameters.
- It may return a value.
- A function is designed to perform a specific task.

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n)
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
}
```

Functions

- A function is a block of code.
- It runs when it is called.
- A function may have parameters.
- It may return a value.
- A function is designed to perform a specific task.

```
void test( )
{
        cout << fac( 5 ) << endl;
        printf_information( );
}
```

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n)
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
}
```

Functions

- A function is a block of code.
- It runs when it is called.
- A function may have parameters.
- It may return a value.
- A function is designed to perform a specific task.

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n )
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
}
```

Functions

- A function is a block of code.
- It runs when it is called.
- A function may have parameters.
- It may return a value.
- A function is designed to perform a specific task.

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n )
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
        return; // no value returned
}
```

Functions

- A function is a block of code.
- It runs when it is called.
- A function may have parameters.
- It may return a value.
- A function is designed to perform **a specific task**.

```
//Compute factorial. For n >=1, 1*2*…*n
int fac( int n )
{
        if (n==0) return 1;
        return n*fac(n-1);
}
```

```
void printf_information( )
{
        cout << student_name << endl;
        cout << student_ID << endl;
        return; // no value returned
}
```

Example: System requirement

Enter a letter x. Show the next y letters after that letter x.

If x + i is not a letter, do not show it, where i is inside [1, y] and y > 0

a) The y letters are shown in one row. A space character is between two consecutive letters.

b) The y letters are shown in one row. A space character is between two consecutive letters. A double quote is shown before and after the y letters.

c) The y letters are shown in one row. A tab character is between two consecutive letters.

d) Each of the y letters is shown in one row.

Example: System requirement

Enter a letter x. Show the next y letters after that letter x.

If x + i is not a letter, do not show it, where i is inside [1, y] and y > 0

a) The y letters are shown in one row. A space character is between two consecutive letters.

b) The y letters are shown in one row. A space character is between two consecutive letters. A double quote is shown before and after the y letters.

c) The y letters are shown in one row. A tab character is between two consecutive letters.

d) Each of the y letters is shown in one row.

Bits $b_7 b_6 b_5$ →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Row ↓ / Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Bits $b_7 b_6 b_5$ →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

ASCII (American Standard Code for Information Interchange)

https://en.wikipedia.org/wiki/ASCII#/media/File:USASCII_code_chart.png

ASCII Code

The ASCII for lowercase letters are consecutive integers starting from the code for 'a', then for 'b', 'c', ..., and 'z'.

The same is true for the uppercase letters.

The ASCII code for 'a' is greater than the code for 'A'.
So 'a' - 'A' is the same as 'b' - 'B', 'c' - ‘C',…

For a lowercase letter *ch*, its corresponding uppercase letter is

'A' + (ch - 'a').

100	101	110	111
4	5	6	7
@	P	`	p
A	Q	a	q
B	R	b	r
C	S	c	s
D	T	d	t
E	U	e	u
F	V	f	v
G	W	g	w
H	X	h	x
I	Y	i	y
J	Z	j	z
K	[	k	{
L	\	l	|
M	]	m	}
N	^	n	~
O	_	o	DEL

ASCII Code

The ASCII for lowercase letters are consecutive integers starting from the code for 'a', then for 'b', 'c', ..., and 'z'.

The same is true for the uppercase letters.

The ASCII code for 'a' is greater than the code for 'A'.
So 'a' - 'A' is the same as 'b' - 'B', 'c' - ‘C',…

For a lowercase letter *ch*, its corresponding uppercase letter is

'A' + (ch - 'a').

1 0 0	1 0 1	1 1 0	1 1 1
4	5	6	7
@	P	`	p
A	Q	a	q
B	R	b	r
C	S	c	s
D	T	d	t
E	U	e	u
F	V	f	v
G	W	g	w
H	X	h	x
I	Y	i	y
J	Z	j	z
K	[	k	{
L	\	l	|
M	]	m	}
N	^	n	~
O	_	o	DEL

ASCII Code

The ASCII for lowercase letters are consecutive integers starting from the code for 'a', then for 'b', 'c', ..., and 'z'.

The same is true for the uppercase letters.

The ASCII code for 'a' is greater than the code for 'A'.
So 'a' - 'A' is the same as 'b' - 'B', 'c' - ‘C',…

For a lowercase letter *ch*, its corresponding uppercase letter is

'A' + (ch - 'a').

100	101	110	111
4	5	6	7
@	P	`	p
A	Q	a	q
B	R	b	r
C	S	c	s
D	T	d	t
E	U	e	u
F	V	f	v
G	W	g	w
H	X	h	x
I	Y	i	y
J	Z	j	z
K	[	k	{
L	\	l	|
M	]	m	}
N	^	n	~
O	_	o	DEL

ASCII Code

The ASCII for lowercase letters are consecutive integers starting from the code for 'a', then for 'b', 'c', ..., and 'z'.

The same is true for the uppercase letters.

The ASCII code for 'a' is greater than the code for 'A'.
So 'a' - 'A' is the same as 'b' - 'B', 'c' - ‘C',…

For a lowercase letter *ch*, its corresponding uppercase letter is

'A' + (ch - 'a').

1 0 0	1 0 1	1 1 0	1 1 1
4	5	6	7
@	P	`	p
A	Q	a	q
B	R	b	r
C	S	c	s
D	T	d	t
E	U	e	u
F	V	f	v
G	W	g	w
H	X	h	x
I	Y	i	y
J	Z	j	z
K	[	k	{
L	\	l	|
M	]	m	}
N	^	n	~
O	_	o	DEL

ASCII Code

The ASCII for lowercase letters are consecutive integers starting from the code for 'a', then for 'b', 'c', ..., and 'z'.

The same is true for the uppercase letters.

The ASCII code for **'a' is greater than the code for 'A'**.
So 'a' - 'A' is the same as 'b' - 'B', 'c' - ‘C',…

For a lowercase letter *ch*, its corresponding uppercase letter is

'A' + (ch - 'a').

100	101	110	111
4	5	6	7
@	P	`	p
A	Q	a	q
B	R	b	r
C	S	c	s
D	T	d	t
E	U	e	u
F	V	f	v
G	W	g	w
H	X	h	x
I	Y	i	y
J	Z	j	z
K	[	k	{
L	\	l	|
M	]	m	}
N	^	n	~
O	_	o	DEL

ASCII Code

The ASCII for lowercase letters are consecutive integers starting from the code for 'a', then for 'b', 'c', ..., and 'z'.

The same is true for the uppercase letters.

The ASCII code for **'a' is greater than the code for 'A'**.
So 'a' - 'A' is the same as 'b' - 'B', 'c' - ‘C',…

For a lowercase letter *ch*, its corresponding uppercase letter is

'A' + (ch - 'a').

100	101	110	111
4	5	6	7
@	P	`	p
A	Q	a	q
B	R	b	r
C	S	c	s
D	T	d	t
E	U	e	u
F	V	f	v
G	W	g	w
H	X	h	x
I	Y	i	y
J	Z	j	z
K	[	k	{
L	\	l	|
M	]	m	}
N	^	n	~
O	_	o	DEL

ASCII Code

The ASCII for lowercase letters are consecutive integers starting from the code for 'a', then for 'b', 'c', ..., and 'z'.

The same is true for the uppercase letters.

The ASCII code for **'a' is greater than the code for 'A'**.
So 'a' - 'A' is the same as 'b' - 'B', 'c' - 'C',…

For a lowercase letter *ch*, its corresponding uppercase letter is

'A' + (ch - 'a').

100	101	110	111
4	5	6	7
@	P	`	p
A	Q	a	q
B	R	b	r
C	S	c	s
D	T	d	t
E	U	e	u
F	V	f	v
G	W	g	w
H	X	h	x
I	Y	i	y
J	Z	j	z
K	[	k	{
L	\	l	|
M	]	m	}
N	^	n	~
O	_	o	DEL

ASCII Code

The ASCII for lowercase letters are consecutive integers starting from the code for 'a', then for 'b', 'c', ..., and 'z'.

The same is true for the uppercase letters.

The ASCII code for **'a' is greater than the code for 'A'**.
So 'a' - 'A' is the same as 'b' - 'B', 'c' - ‘C',…

For a lowercase letter *ch*, its corresponding uppercase letter is

'A' + (ch - 'a').

100	101	110	111
4	5	6	7
@	P	`	p
A	Q	a	q
B	R	b	r
C	S	c	s
D	T	d	t
E	U	e	u
F	V	f	v
G	W	g	w
H	X	h	x
I	Y	i	y
J	Z	j	z
K	[	k	{
L	\	l	|
M	]	m	}
N	^	n	~
O	_	o	DEL

Example: System requirement

Enter a letter x. Show the next y letters after that letter x.

If x + i is not a letter, do not show it, where i is inside [1, y] and y > 0

a) The y letters are shown in one row. A space character is between two consecutive letters.

b) The y letters are shown in one row. A space character is between two consecutive letters. A double quote is shown before and after the y letters.

c) The y letters are shown in one row. A tab character is between two consecutive letters.

d) Each of the y letters is shown in one row.

x = ‘A’
y = 3

‘B’
‘C’
‘D’

x = ‘X’
y = 3

‘Y’
‘Z’

Example: System requirement

Enter a letter x. Show the next y letters after that letter x.

If x + i is not a letter, do not show it, where i is inside [1, y] and y > 0

a) The y letters are shown in one row. A space character is between two consecutive letters.

b) The y letters are shown in one row. A space character is between two consecutive letters. A double quote is shown before and after the y letters.

c) The y letters are shown in one row. A tab character is between two consecutive letters.

d) Each of the y letters is shown in one row.

x = ‘A’
y = 3

‘B’
‘C’
‘D’

x = ‘X’
y = 3

‘Y’
‘Z’

```
void generateLetters( ) {
   char x;
   int y;
   cout << "Input x:";
   cin >>  x;
   char nc;
   for (int i = 1; i <= y; ++i ) {
            nc = x + i;
            if ( (nc >= 'a' && nc <= 'z')
                ||
                (nc >= 'A' && nc <= 'Z')
            ) {
              cout << nc << endl;
            }
   } // end for
}
```

Example: System requirement

Enter a letter x. Show the next y letters after that letter x.

If x + i is not a letter, do not show it, where i is inside [1, y] and y > 0

a) The y letters are shown in one row. A space character is between two consecutive letters.

b) The y letters are shown in one row. A space character is between two consecutive letters. A double quote is shown before and after the y letters.

c) The y letters are shown in one row. A tab character is between two consecutive letters.

d) Each of the y letters is shown in one row.

x = ‘A’
y = 3

‘B’
‘C’
‘D’

x = ‘X’
y = 3

‘Y’
‘Z’

```
void generateLetters( ) {
   char x;
   int y;
   cout << "Input x:";
   cin >>  x;
   char nc;
   for (int i = 1; i <= y; ++i ) {
            nc = x + i;
            if ( (nc >= 'a' && nc <= 'z')
                ||
                (nc >= 'A' && nc <= 'Z')
            ) {
              cout << nc << endl;
            }
   } // end for
}
```

Is there any BUG?

Example: System requirement

Enter a letter x. Show the next y letters after that letter x.

If x + i is not a letter, do not show it, where i is inside [1, y] and y > 0

a) The y letters are shown in one row. A space character is between two consecutive letters.

b) The y letters are shown in one row. A space character is between two consecutive letters. A double quote is shown before and after the y letters.

c) The y letters are shown in one row. A tab character is between two consecutive letters.

d) Each of the y letters is shown in one row.

x = ‘A’
y = 3

‘B’
‘C’
‘D’

x = ‘X’
y = 3

‘Y’
‘Z’

```
void generateLetters( ) {
   char x;
   int y;
   cout << "Input x:";
   cin >>  x;
   cout << "Input y:";
   cin >>  y;
   char nc;
   for (int i = 1; i <= y; ++i ) {
            nc = x + i;
            if ( (nc >= 'a' && nc <= 'z')
                ||
                (nc >= 'A' && nc <= 'Z')
            ) {
              cout << nc << endl;
            }
   } // end for
}
```

Example: System requirement

- Input a string. Show each of the letters of the string in one row.

Input:

Abcd

Output:

A

b

c

d

Example: System requirement

- Input a string. Show each of the letters of the string in one row.

Input:

Abcd

Output:

A

b

c

d

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

void showElementsOfString( ) {
   string str;
   cout << "Input a string:";
   cin >>  str;
   cout << “Output:“ << endl;
   for (int i = 0; i < str.size( ); ++i ) {
            cout << str[ i ] << endl;
    } // end for
}
```

Example: System requirement

- Ask to input integers until a non-positive integer is input.
- Compute the average of all the input positive integers.

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

double computeAverage( ) {
   vector<int> myArray;
   while ( true )  {
            int a;
            cin >> a;
            if ( a <= 0 ) break;
            myArray.push_back( a );
   }
   int total = 0;
   for ( int i =0; i < myArray.size( ); ++i )
            total += myArray[ i ];
   double average = total / myArray.size( );
   return average;
}
```

Example: System requirement

- Ask to input integers until a non-positive integer is input.
- Compute the average of all the input positive integers.

BUG????

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

double computeAverage( ) {
   vector<int> myArray;
   while ( true )  {
            int a;
            cin >> a;
            if ( a <= 0 ) break;
            myArray.push_back( a );
   }
   int total = 0;
   for ( int i =0; i < myArray.size( ); ++i )
            total += myArray[ i ];
   double average = total / myArray.size( );
   return average;
}
```

Example: System requirement

- Ask to input integers until a non-positive integer is input.
- Compute the average of all the input positive integers.

Division by zero if no input

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

double computeAverage( ) {
   vector<int> myArray;
   while ( true )  {
            int a;
            cin >> a;
            if ( a <= 0 ) break;
            myArray.push_back( a );
   }
   int total = 0;
   for ( int i =0; i < myArray.size( ); ++i )
            total += myArray[ i ];
   double average = total / myArray.size( );
   return average;
}
```

Example: System requirement

- Ask to input integers until a non-positive integer is input.
- Compute the average of all the input positive integers.

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

double computeAverage( ) {
   vector<int> myArray;
   while ( true )  {
            int a;
            cin >> a;
            if ( a <= 0 ) break;
            myArray.push_back( a );
   }
   if ( myArray.size( ) == 0 ) return 0;
   int total = 0;
   for ( int i =0; i < myArray.size( ); ++i )
            total += myArray[ i ];
   double average = total / myArray.size( );
   return average;
}
```

Example: System requirement

- Ask to input integers until a non-positive integer is input.
- Compute the average of all the input positive integers.

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

double computeAverage( ) {
   vector<int> myArray;
   while ( true )  {
            int a;
            cin >> a;
            if ( a <= 0 ) break;
            myArray.push_back( a );
   }
   if ( myArray.size( ) == 0 ) return 0; // -1?
   int total = 0;
   for ( int i =0; i < myArray.size( ); ++i )
            total += myArray[ i ];
   double average = total / myArray.size( );
   return average;
}
```

Example: System requirement

- Ask to input integers until a non-positive integer is input.
- Compute the average of all the input positive integers.

BUG????

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

double computeAverage( ) {
   vector<int> myArray;
   while ( true )  {
            int a;
            cin >> a;
            if ( a <= 0 ) break;
            myArray.push_back( a );
   }
   if ( myArray.size( ) == 0 ) return 0;
   int total = 0;
   for ( int i =0; i < myArray.size( ); ++i )
            total += myArray[ i ];
   double average = total / myArray.size( );
   return average;
}
```

Example: System requirement

- Ask to input integers until a non-positive integer is input.
- Compute the average of all the input positive integers.

Integer division
5/2 = 2 instead of 2.5

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

double computeAverage( ) {
   vector<int> myArray;
   while ( true )  {
            int a;
            cin >> a;
            if ( a <= 0 ) break;
            myArray.push_back( a );
   }
   if ( myArray.size( ) == 0 ) return 0;
   int total = 0;
   for ( int i =0; i < myArray.size( ); ++i )
            total += myArray[ i ];
   double average = total / myArray.size( );
   return average;
}
```

Example: System requirement

- Ask to input integers until a non-positive integer is input.
- Compute the average of all the input positive integers.

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

double computeAverage( ) {
   vector<int> myArray;
   while ( true )  {
            int a;
            cin >> a;
            if ( a <= 0 ) break;
            myArray.push_back( a );
   }
   if ( myArray.size( ) == 0 ) return 0;
   int total = 0;
   for ( int i =0; i < myArray.size( ); ++i )
            total += myArray[ i ];
   double average = total / (double) myArray.size( );
   return average;
}
```

Example: Ask the user to input the number of unsigned integers. Input the unsigned integers. Compute their average.

```
Input the number of unsigned integers: 4
Input unsigned integer [1]: 5
Input unsigned integer [2]: 7
Input unsigned integer [3]: 2
Input unsigned integer [4]: 1

Total: 15
Average: 3.75
```

Example: Ask the user to input the number of unsigned integers. Input the unsigned integers. Compute their average.

```
int num;
cin >> num;
vector<unsigned int> numArr;
numArr.resize(num);
int i = 0; unsigned int tmp;
while ( i < num ) {
        cin >> tmp;
        numArr.push_back( tmp );
        ++i;
}
```

```
unsigned int total = 0;
for (int i  = 0; i < num; ++i ) {
        total += numArr[i];
}

cout << "Total:"
        << total << endl;
cout << "Average:"
        << total/num << endl;
```

```
Input the number of unsigned integers: 4
Input unsigned integer [1]: 5
Input unsigned integer [2]: 7
Input unsigned integer [3]: 2
Input unsigned integer [4]: 1

Total: 15
Average: 3.75
```

Example: Ask the user to input the number of unsigned integers. Input the unsigned integers. Compute their average.

```
int num;
cin >> num;
vector<unsigned int> numArr;
numArr.resize(num);
int i = 0; unsigned int tmp;
while ( i < num ) {
        cin >> tmp;
        numArr.push_back( tmp );
        ++i;
}
```

```
unsigned int total = 0;
for (int i  = 0; i < num; ++i ) {
        total += numArr[i];
}

cout << "Total:"
        << total << endl;
cout << "Average:"
        << total/num << endl;
```

```
Input the number of unsigned integers: 4
Input unsigned integer [1]: 5
Input unsigned integer [2]: 7
Input unsigned integer [3]: 2
Input unsigned integer [4]: 1

Total: 15
Average: 3.75
```

Are there any bugs? If there are bugs, fix all of them.

Example: Ask the user to input the number of unsigned integers. Input the unsigned integers. Compute their average.

```
int num;
cin >> num;
vector<unsigned int> numArr;
numArr.resize(num);
int i = 0; unsigned int tmp;
while ( i < num ) {
        cin >> tmp;
        numArr.push_back( tmp );
        ++i;
}
```

```
unsigned int total = 0;
for (int i  = 0; i < num; ++i ) {
        total += numArr[i];
}

cout << "Total:"
        << total << endl;
cout << "Average:"
        << total/num << endl;
```

```
Input the number of unsigned integers: 4
Input unsigned integer [1]: 5
Input unsigned integer [2]: 7
Input unsigned integer [3]: 2
Input unsigned integer [4]: 1

Total: 15
Average: 3.75
```

numArr.push_back(tmp) stores the elements at position starting from num instead of 0.

Example: Ask the user to input the number of unsigned integers. Input the unsigned integers. Compute their average.

```
int num;
cin >> num;
vector<unsigned int> numArr;
numArr.resize(num);
int i = 0; unsigned int tmp;
while ( i < num ) {
        cin >> tmp;
        numArr.push_back( tmp );
        ++i;
}
```

```
unsigned int total = 0;
for (int i  = 0; i < num; ++i ) {
        total += numArr[i];
}

cout << "Total:"
        << total << endl;
cout << "Average:"
        << total/num << endl;
```

```
Input the number of unsigned integers: 4
Input unsigned integer [1]: 5
Input unsigned integer [2]: 7
Input unsigned integer [3]: 2
Input unsigned integer [4]: 1

Total: 15
Average: 3.75
```

				5	7	2	1

numArr.push_back(tmp) stores the elements at position starting from num instead of 0.

Example: Ask the user to input the number of unsigned integers. Input the unsigned integers. Compute their average.

```
int num;
cin >> num;
vector<unsigned int> numArr;
numArr.resize(num);
int i = 0; unsigned int tmp;
while ( i < num ) {
        cin >> tmp;
        numArr.push_back( tmp );
        ++i;
}
```

```
unsigned int total = 0;
for (int i  = 0; i < num; ++i ) {
        total += numArr[i];
}

cout << "Total:"
        << total << endl;
cout << "Average:"
        << total/num << endl;
```

```
Input the number of unsigned integers: 4
Input unsigned integer [1]: 5
Input unsigned integer [2]: 7
Input unsigned integer [3]: 2
Input unsigned integer [4]: 1

Total: 15
Average: 3.75
```

				5	7	2	1

Integer division. Do you want floating point division?

Example: Ask the user to input the number of unsigned integers. Input the unsigned integers. Compute their average.

```
int num;
cin >> num;
vector<unsigned int> numArr;
numArr.resize(num);
int i = 0; unsigned int tmp;
while ( i < num ) {
        cin >> tmp;
        numArr[ i ] = tmp;
        ++i;
}
```

~~numArr.push_back(tmp);~~

```
unsigned double total = 0;
for (int i  = 0; i < num; ++i ) {
        total += numArr[i];
}

cout << "Total:"
        << total << endl;
cout << "Average:"
        << total/num << endl;
```

```
Input the number of unsigned integers: 4
Input unsigned integer [1]: 5
Input unsigned integer [2]: 7
Input unsigned integer [3]: 2
Input unsigned integer [4]: 1

Total: 15
Average: 3.75
```

0	0	0	0

Integer division. Do you want floating point division?

Example: Ask the user to input the number of unsigned integers. Input the unsigned integers. Compute their average.

```
int num;
cin >> num;
vector<unsigned int> numArr;
numArr.resize(num);
int i = 0; unsigned int tmp;
while ( i < num ) {
        cin >> tmp;
        numArr[ i ] = tmp;
        ++i;
}
```

~~numArr.push_back(tmp);~~

```
unsigned double total = 0;
for (int i  = 0; i < num; ++i ) {
        total += numArr[i];
}

cout << "Total:"
        << total << endl;
cout << "Average:"
        << total/num << endl;
```

```
Input the number of unsigned integers: 4
Input unsigned integer [1]: 5
Input unsigned integer [2]: 7
Input unsigned integer [3]: 2
Input unsigned integer [4]: 1

Total: 15
Average: 3.75
```

5	7	2	1

Integer division. Do you want floating point division?

Example: Show the intermediate results to avoid mistakes. Use extra variables when necessary (to improve readability)

```
int num;
cin >> num;
vector<unsigned int> numArr;

numArr.resize(num);



int i = 0; unsigned int tmp;
while ( i < num ) {
   cin >> tmp;
   numArr[ i ] = tmp;
   ++i;

}
```

```
unsigned double total = 0;

for (int i  = 0; i < num; ++i ) {

   total += numArr[ i ];
}



cout << “Total:” << total << endl;
cout << “Average:” << total/num << endl;
```

Example: Show the intermediate results to avoid mistakes. Use extra variables when necessary (to improve readability)

```
int num;
cin >> num;
vector<unsigned int> numArr;

numArr.resize(num);
cout << "size:" << numArr.size( ) << endl;

int i = 0; unsigned int tmp;
while ( i < num ) {
   cin >> tmp;
   numArr[ i ] = tmp;
   ++i;
   cout << "size:" << numArr.size( ) << endl;
}
```

```
unsigned double total = 0;

for (int i  = 0; i < num; ++i ) {
   cout << "element [" <<  i << " ]:" << numArr[i] << endl;
   total += numArr[ i ];
}

double average = total/num;
cout << "Total:" << total << endl;
cout << "Average:" << average << endl;
```

Intended Learning Outcomes

- List some properties of functions
- Describe how to use vectors

Rounding Functions

`ceil(x)`	x is rounded up to its nearest integer. This integer is returned as a double value.
`floor(x)`	x is rounded down to its nearest integer. This integer is returned as a double value.

```
x = 4.5;
a = ceil ( x );

b = floor ( x );
```

```
x = 4.1;
c = ceil ( x );

d = floor ( x );
```

The min, max, and abs Functions

- max(**2**, **3**) returns **3**
- max(**2.5**, **4.0**) returns **4.0**
- min(**2.5**, **4.6**) returns **2.5**
- abs(**-2**) returns **2**
- abs(**-2.5**) returns **2.5**

Character Data Type

An ordinal data type

Character Data Type

```
char letter = ‘B';          (ASCII)
char numChar = ‘5';         (ASCII)

char ch = 'a';
cout << ++ch;               // increment ch by one and display the new character

char ch2 = ‘y';
cout << --ch2;              // decrement ch2 by one and display the new character
cout << ch2++;
cout << ++ch2;
```

Character Data Type

```
char letter = ‘B';          (ASCII)
char numChar = ‘5';         (ASCII)

char ch = 'a';
cout << ++ch;               // increment ch by one and display the new character

char ch2 = ‘y';
cout << --ch2;              // decrement ch2 by one and display the new character
cout << ch2++;
cout << ++ch2;
```

What are the output?

Character Data Type

```
char letter = 'B';          (ASCII)
char numChar = '5';         (ASCII)

char ch = 'a';
cout << ++ch;      A1   // increment ch by one and display the new character

char ch2 = 'y';
cout << --ch2;     A2   // decrement ch2 by one and display the new character
cout << ch2++;     A3
cout << ++ch2;     A4
```

What are the output?

Character Data Type

```
char letter = 'B';          (ASCII)
char numChar = '5';         (ASCII)

char ch = 'a';
cout << ++ch;     b         // increment ch by one and display the new character

char ch2 = 'y';
cout << --ch2;    x         // decrement ch2 by one and display the new character
cout << ch2++;    x
cout << ++ch2;    z
```

What are the output?

Casting between char and numeric types

```
int i = 'b';
```

```
int i = static_cast<A1>('b');
```

```
char c = 98;
```

```
char c = A2<char>(98);
```

Numeric Operators on Characters

The char type is treated as if it is an integer of the byte size.
All numeric operators can be applied to char operands.

```
int i = '1' + '3';                    // (int) '1' is 49 and (int)'3' is 51

cout << "i is " << i << endl;         // i is decimal 100

int j = 2 + 'b';                      // (int) ' b' is 98
```

000	001	110	111
4	5	6	7
@	P	`	p
A	Q	a	q
B	R	b	r
C	S	c	s
D	T	d	t
E	U	e	u
F	V	f	v
G	W	g	w
H	X	h	x
I	Y	i	y
J	Z	j	z
K	[	k	{
L	\	l	|
M	]	m	}
N	^	n	~
O	_	o	DEL

Numeric Operators on Characters

```
int i = ‘1' + '3';                    // (int) '1' is 49 and (int)'3' is 51
cout << "i is " << i << endl;         // i is decimal 100

int j = 2 + ‘b';     // (int) ' b' is 98
cout << "j is " << j  << endl;



cout << j
        << " is the ASCII code for character "
        << static_cast<char>(j)
        << endl;
```

Numeric Operators on Characters

```
int i = '1' + '3';                    // (int) '1' is 49 and (int)'3' is 51
cout << "i is " << i << endl;         // i is decimal 100

int j = 2 + 'b';      // (int) ' b' is 98
cout << "j is " << j  << endl;

cout << j
        << " is the ASCII code for character "
        << static_cast<char>(j)
        << endl;
```

Output:

i is A1
j is A2
100 is the ASCII code for character A3

Comparing and Testing Characters

Two characters can be compared using the **comparison operators** just like comparing two numbers.

'a' < 'b' is true because the ASCII code for **'a'** (**97**) is less than the ASCII code for **'b'** (**98**).

'a' < 'A' is true because the ASCII code for **'a'** (**97**) is less than the ASCII code for **'A'** (**65**).

'1' < '8' is true because the ASCII code for **'1'** (**49**) is less than the ASCII code for **'8'** (**56**).

String

Concepts

The string Type

```
string s;

string message = "Programming is fun";

message.length( )          // return the number of characters
message.size();            // return as length()
message.at(index)          // return the character at the index from the string
```

String Subscript Operator

```
string message = “Hello!”;
cout << message[3] << endl;
cout << message[5] << endl;
```

0	1	2	3	4	5
H	e	l	l	o	!

stringName[index]: the subscript operator for accessing the character at a specified index in a string.

Retrieve and modify the character in a string.

String Subscript Operator

```
string message = “Hello!”;
cout << message[3] << endl;
cout << message[5] << endl;
```

0	1	2	3	4	5
H	e	l	l	o	!

stringName[index]: the subscript operator for accessing the character at a specified index in a string.

message.length() is A1

Retrieve and modify the character in a string.

String Subscript Operator

```
string s = "ABCD";
s[0] = 'P';
cout << s[0] << endl;
cout << s << endl;
```

Concatenating Strings

```
string s1, s2;
s1…
s2…
string s3 = s1 + s2;

message += " and programming is fun";
```

Comparing Strings

Use the relational operators to compare two strings: **==**, **!=**, **<**, **<=**, **>**, **>=**

Compare their corresponding characters one by one from left to right.

```
string s1 = "ABD";
string s2 = "ABF";
cout << (s1 == s2) << endl;      // Output   0 (means false)
cout << (s1 != s2) << endl;      // Output   1 (means true)
cout << (s1 > s2) << endl;       // Output   0 (means false)
cout << (s1 >= s2) << endl;      // Output   0 (means false)
cout << (s1 < s2) << endl;       // Output   1 (means true)
cout << (s1 <= s2) << endl;      // Output   1 (means true)
```

Reading Strings

```
string place;
cout << "Enter a place: ";
cin >> place;                          // Read to array place
cout << "You entered " << place << endl;
```

```
string place;
cout << "Enter a place: ";
getline(cin, place, '\n');             // Same as getline(cin, place)
cout << "You entered " << place << endl;
```

Simple file output/input

Concepts

Simple File Output : Text mode

Write data to a file:

```
ofstream output;                    // declare a variable of the ofstream type
output.open("numbers.txt");         // open the file
```

An alternative to open a file to output:

```
ofstream output("numbers.txt");     // create file and open it
```

To write data, use the stream insertion operator (**<<**).

```
output << 95 << " " << 56 << " " << 34 << endl;

// quite similar to cout
```

Simple File Input : Text mode

Read data from a file

```
ifstream input;                    // declare a variable of the ifstream type
input.open("numbers.txt"); // specify a file. Invoke the open function
```

An alternative to open a file (or create the file at the same time):

```
ifstream input("numbers.txt");
```

To read data, use the stream extraction operator (**>>**).
For example,

```
input >> score1 >> score2 >> score3;                    (Good)

// quite similar to cin
```

Remark

- Remember to close the file after using it.

- myFile.A1
- myFile.A2

Random numbers

Concepts

Generating Random Characters

```
const int seed = 7;
srand( seed );                    // set the seed
unsigned int c = rand()%128;      // generate a number randomly in [0,127]
```

Always save the seed for testing programs

Generating Random Characters

```
long seed = time(NULL)
srand(seed)
```

srand(0) set the seed

srand(time(NULL)) use current time as the seed. Must save the seed

Generating Random Characters

rand( )%10	returns a random integer between 0 and 9.
50 + (rand( )% 50)	returns a random integer between 50 and 99
a + (rand( ) % b)	returns a random number between a and a + b, excluding a + b. Assume a and b are positive integer.
RAND_MAX	the maximum number that can be generated

Generate a random floating point number between [0, 1]?

Generate a random floating point number between [0, 1]?

rand( ) / A1 (RAND_MAX)

Generate a random floating point number between [0, 1]?

rand( ) / (double) (RAND_MAX)

Generate a random floating point number between [u0, u1]?

```
rand( ) / (double) (RAND_MAX)

double generateRandomNumber( double u0, double u1) {
  double u;
  double f = rand( ) / (double) (RAND_MAX);
  u = u0 + f*(u1 - u0);
  return u;
}
```

Generate a random floating point number between [u0, u1]?

rand( ) / (double) (RAND_MAX)

```
double generateRandomNumber( double u0, double u1) {
  double u;
  double f = rand( ) / (double) (RAND_MAX);
  u = u0 + f*(u1 - u0);
  return u;
}
```

Prove the correctness of the function

Generate a random floating point number between [u0, u1]?

rand( ) / (double) (RAND_MAX)

```
double generateRandomNumber( double u0, double u1) {
  double u;
  double f = rand( ) / (double) (RAND_MAX);
  u = u0 + f*(u1 - u0);
  return u;
}
```

Prove the correctness of the function

f = 0

f = 1

Generate a random floating point number between [u0, u1]?

rand( ) / (double) (RAND_MAX)

```
double generateRandomNumber( double u0, double u1) {
  double u;
  double f = rand( ) / (double) (RAND_MAX);
  u = u0 + f*(u1 - u0);
  return u;
}
```

Prove the correctness of the function

f = 0 => u = u0 + 0*(u1 – u0) => u =

Generate a random floating point number between [u0, u1]?

rand( ) / (double) (RAND_MAX)

```
double generateRandomNumber( double u0, double u1) {
  double u;
  double f = rand( ) / (double) (RAND_MAX);
  u = u0 + f*(u1 - u0);
  return u;
}
```

Prove the correctness of the function

f = 0 => u = u0 + 0*(u1 – u0) => u =

f = 1 => u = u0 + 1*(u1 – u0) => u =

Generate a random floating point number between [u0, u1]?

rand( ) / (double) (RAND_MAX)

```
double generateRandomNumber( double u0, double u1) {
   double u;
   double f = rand( ) / (double) (RAND_MAX);
   u = u0 + f*(u1 - u0);
   return u;
}
```

Prove the correctness of the function

f = 0 => u = u0 + 0*(u1 – u0) => u = A1

f = 1 => u = u0 + 1*(u1 – u0) => u = A2

Generate a random floating point number between [0, 1)? (not including 1)

rand( ) / (double) (A1 A2)

Generate a random floating point number between [0, 1)? (not including 1)

rand( ) / (double) (RAND_MAX+1)

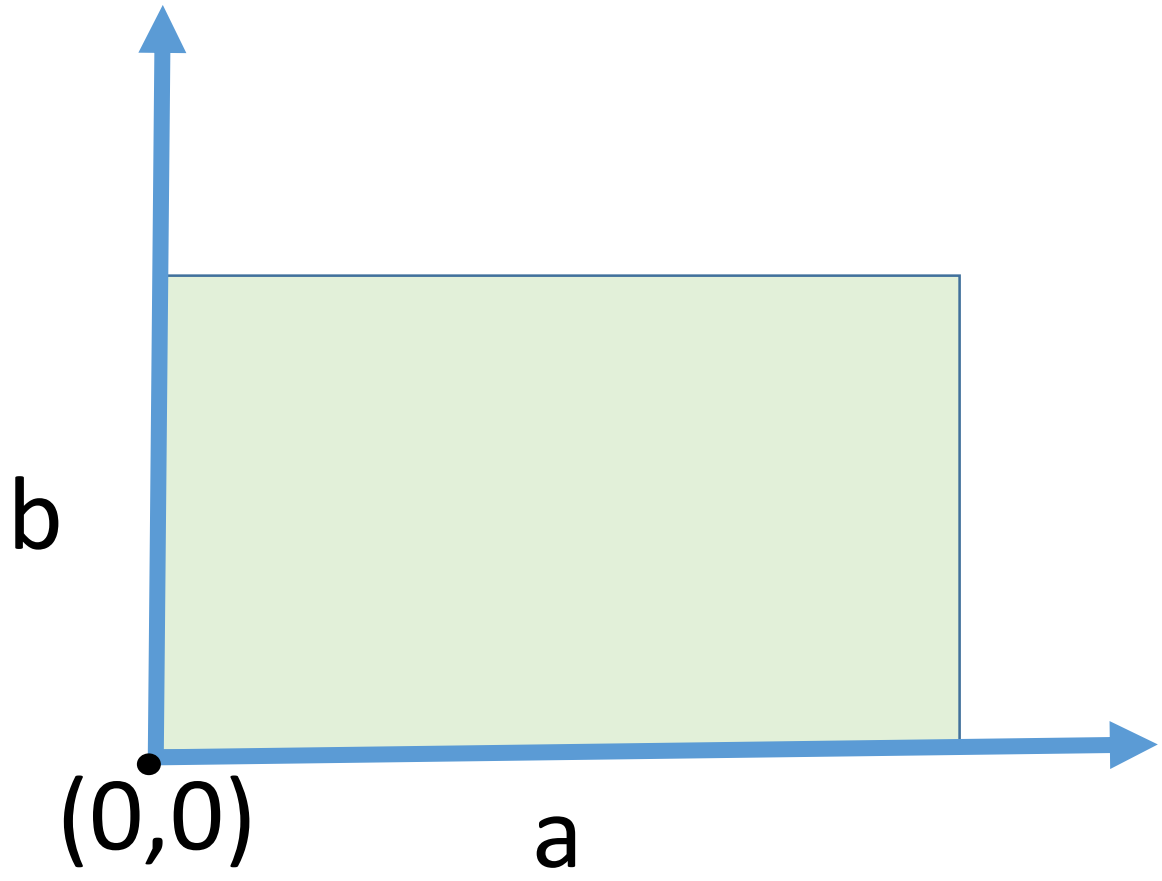
Generate a random floating point number between (0, 1)? (not including 0 and 1)

(A1) / (double) (A2)

Generate a random floating point number between (0, 1)? (not including 0 and 1)

(rand( )+1) / (double) (RAND_MAX+2)

Exercise: Generate a random point inside (inclusive) a rectangle whose side lengths are a and b.
The lower left corner is at the origin.

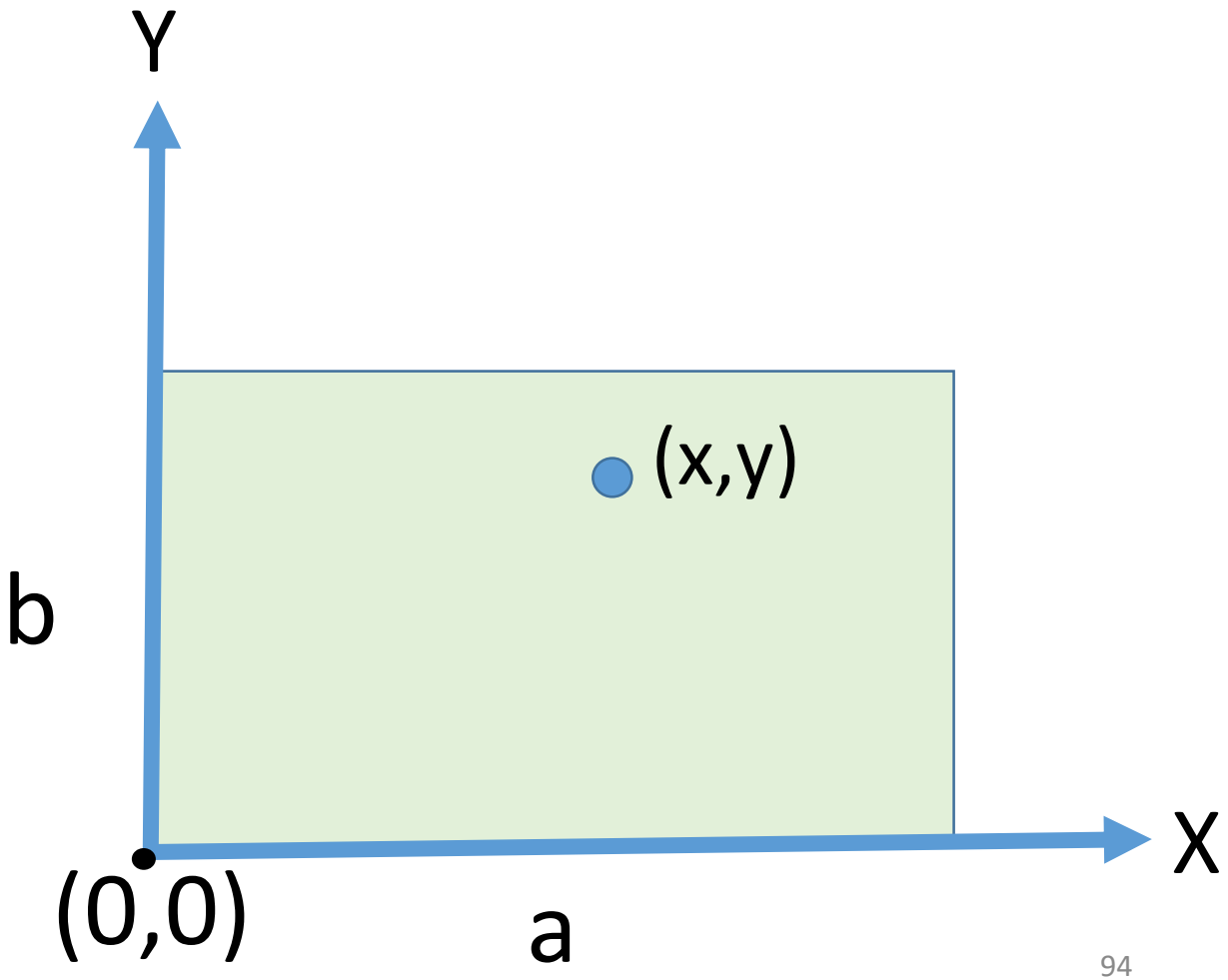


Exercise: Generate a random point inside (inclusive) a rectangle whose side lengths are a and b.
The lower left corner is at the origin.

```
v = generateRandomNumber(0, 1);

x = a*v;
y = b*v;
```

Does it work?

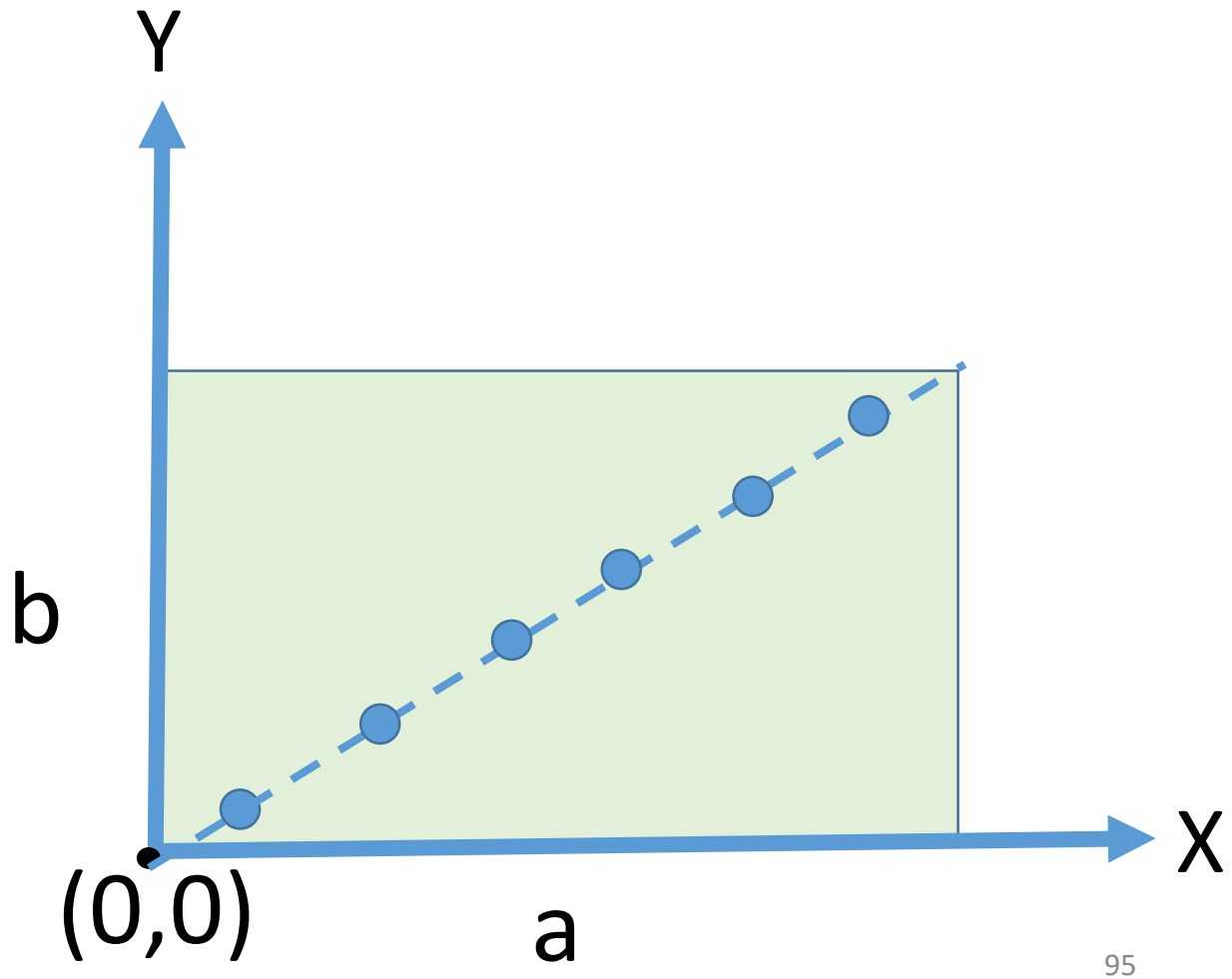


Exercise: Generate a random point inside (inclusive) a rectangle whose side lengths are a and b. The lower left corner is at the origin.

```
v = generateRandomNumber(0, 1);

x = a*v;
y = b*v;
```

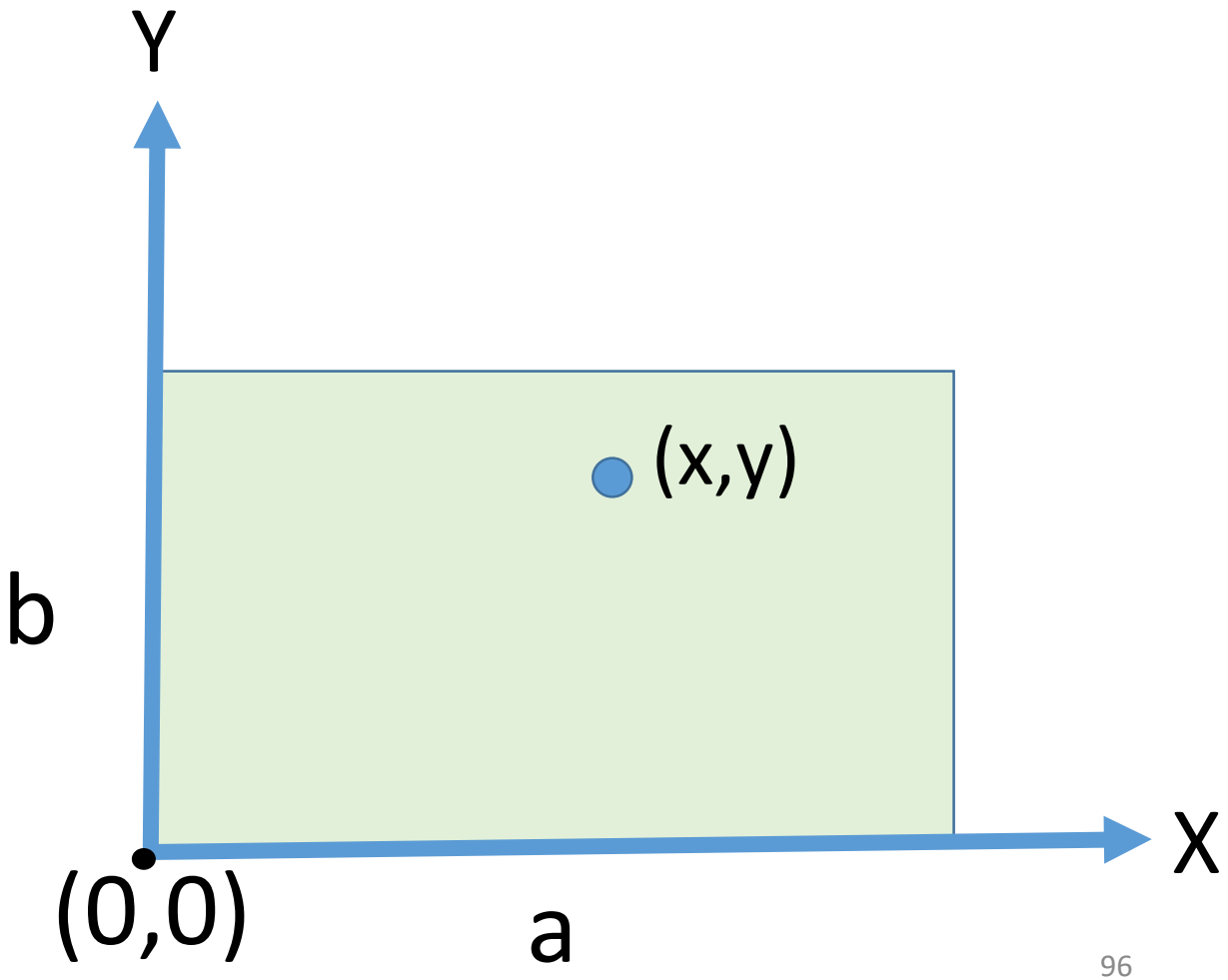
Generate the sample points uniformly along the diagonal line of the rectangle.



Exercise: Generate a random point inside (inclusive) a rectangle whose side lengths are a and b. The lower left corner is at the origin.

x = generateRandomNumber(0, a);

y = A1 A2 ;



Exercise: Generate a random point inside (inclusive) a rectangle whose side lengths are a and b. The lower left corner is at the origin.

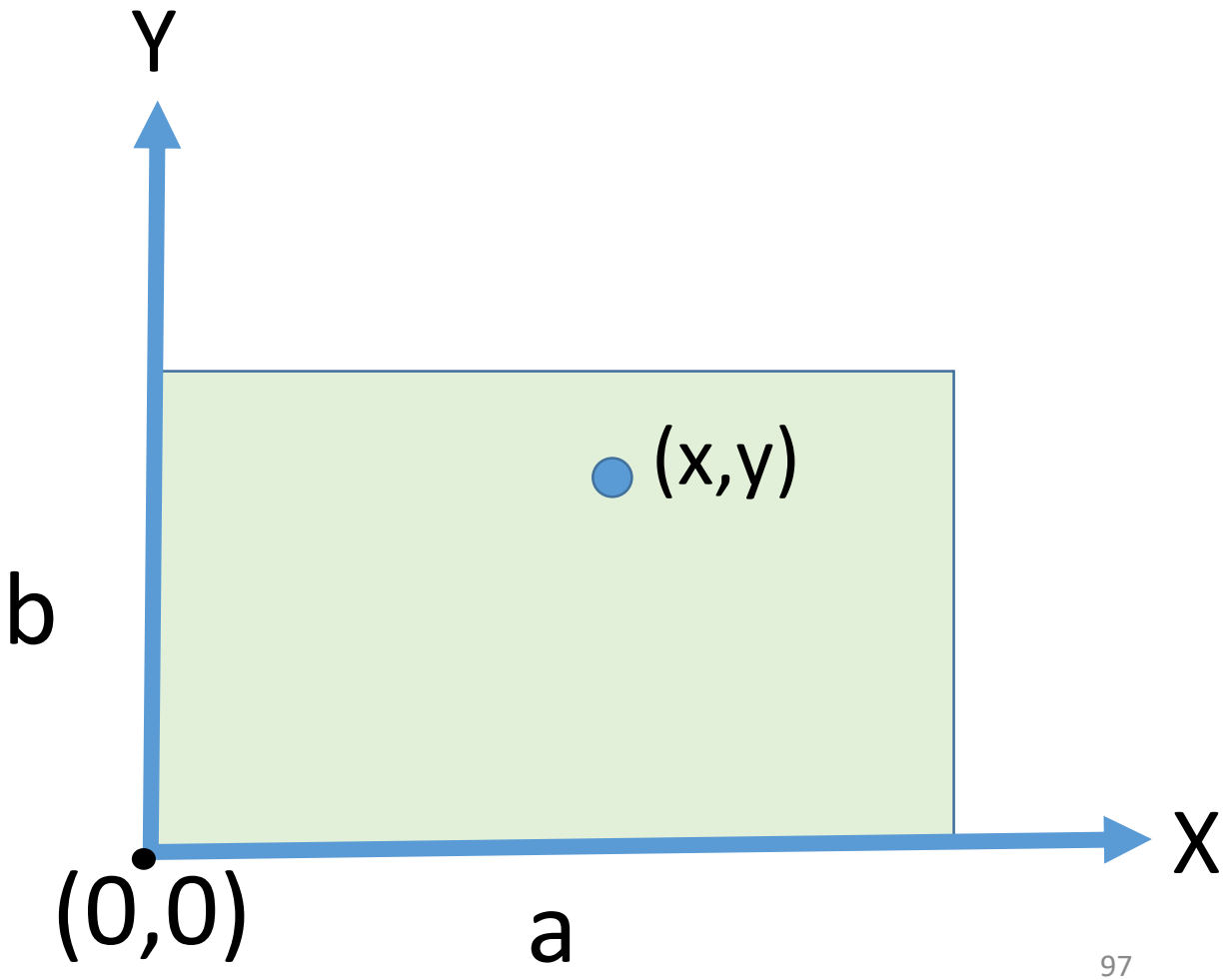
```
x = generateRandomNumber(0, a);

y = generateRandomNumber(0, b);
```

The coordinates x and y are

A1 . So **generate** them

A2 .

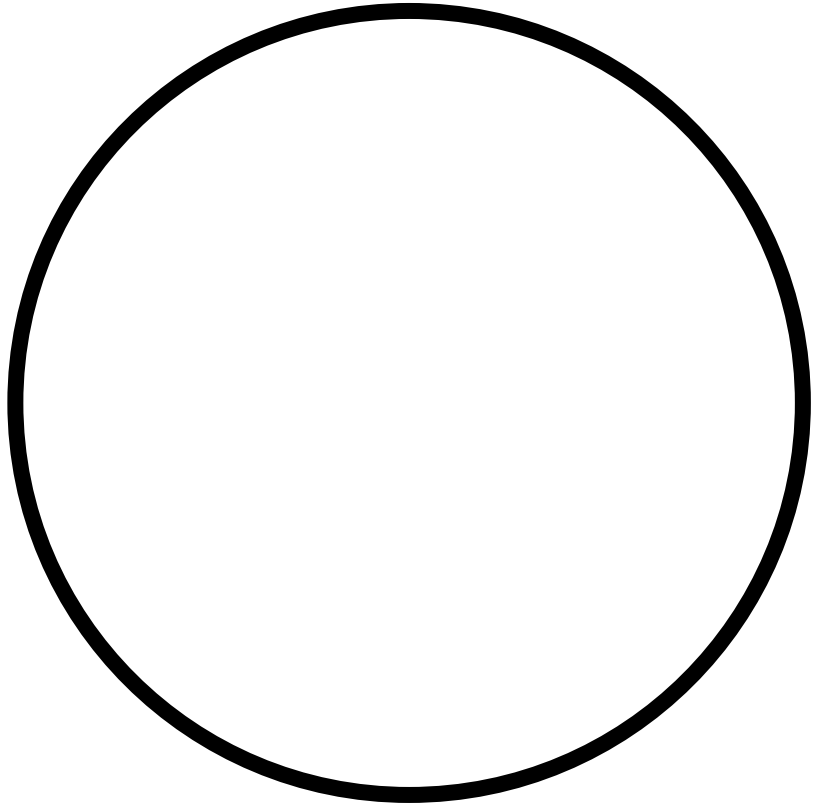


Generate a random point on the boundary of a circle with radius r in the two dimensional space

- Uniformly distributed?

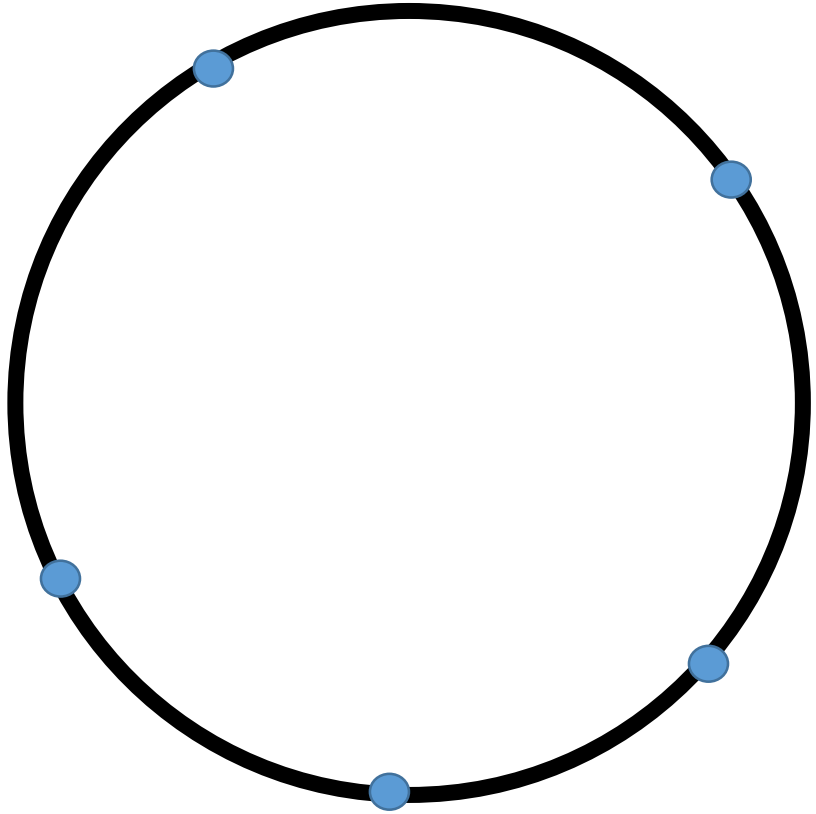
Generate a random point on the boundary of a circle with radius r in the two dimensional space

- Uniformly distributed?



Generate a random point on the boundary of a circle with radius r in the two dimensional space

- Uniformly distributed?



Generate a random point on the boundary of a circle with radius r in the two dimensional space

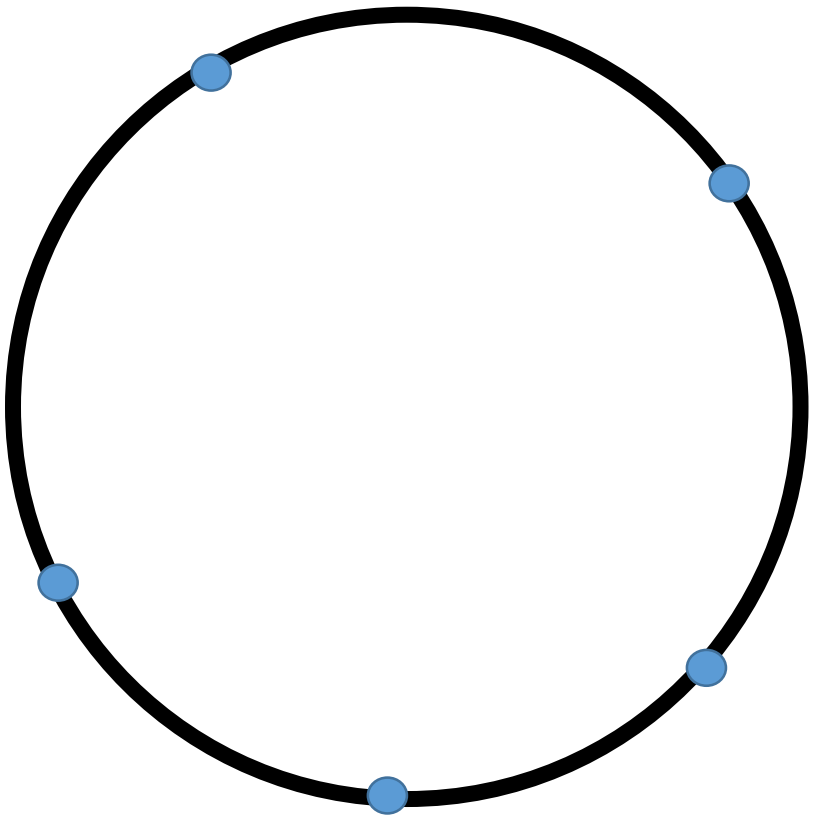
The circle parametric form:

- $x = r \cos(\theta)$
- $y = r \sin(\theta)$

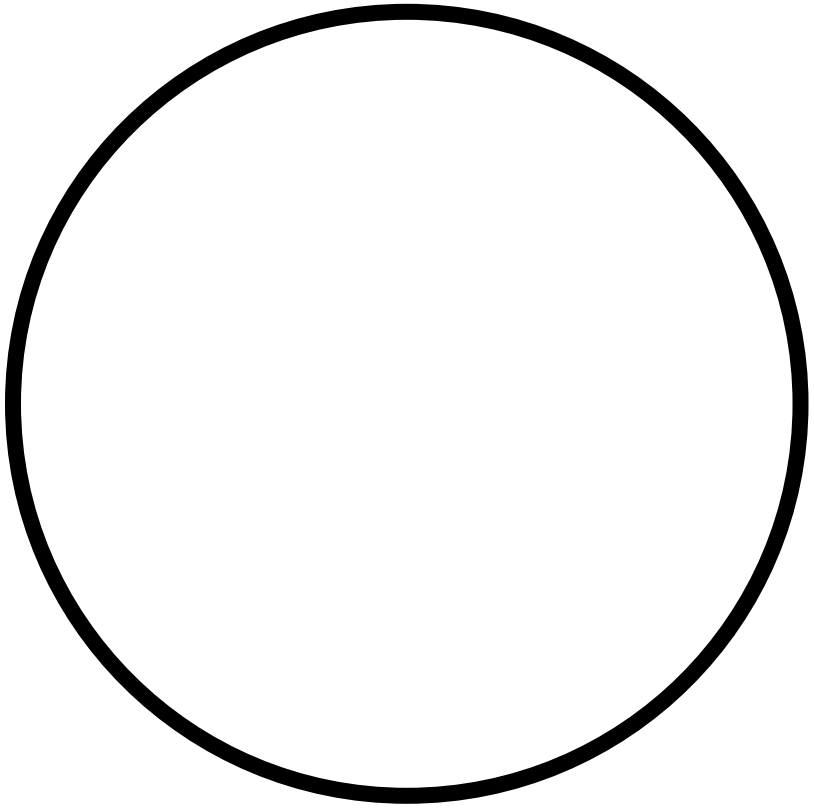
- Sample θ randomly inside $[0, 2\pi)$

v = generateRandomNumber(0, 1);

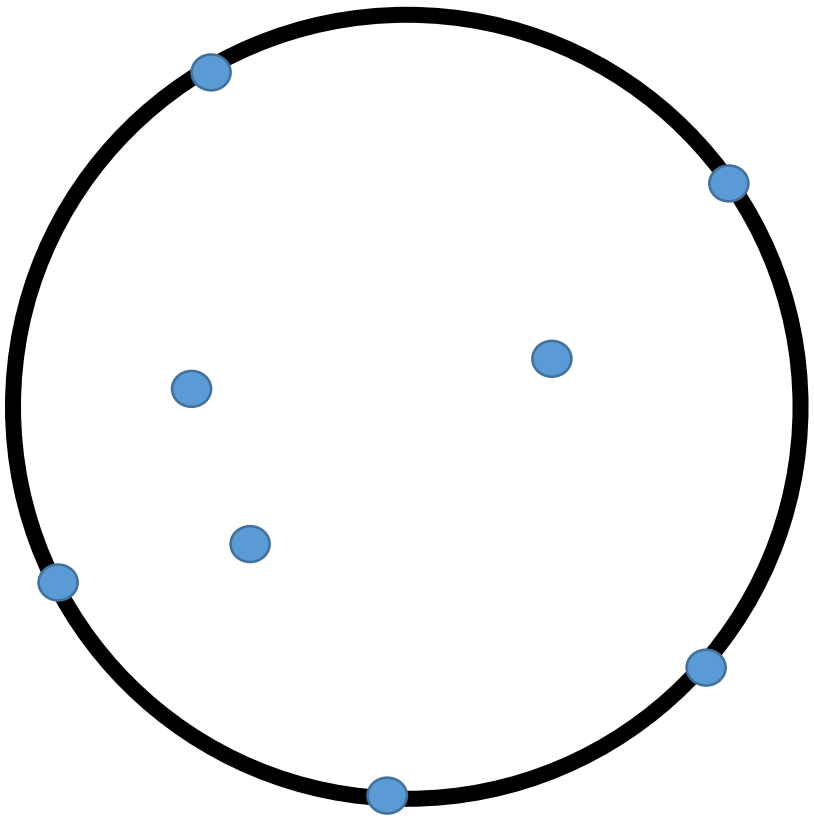
$\theta = 2\pi * v$



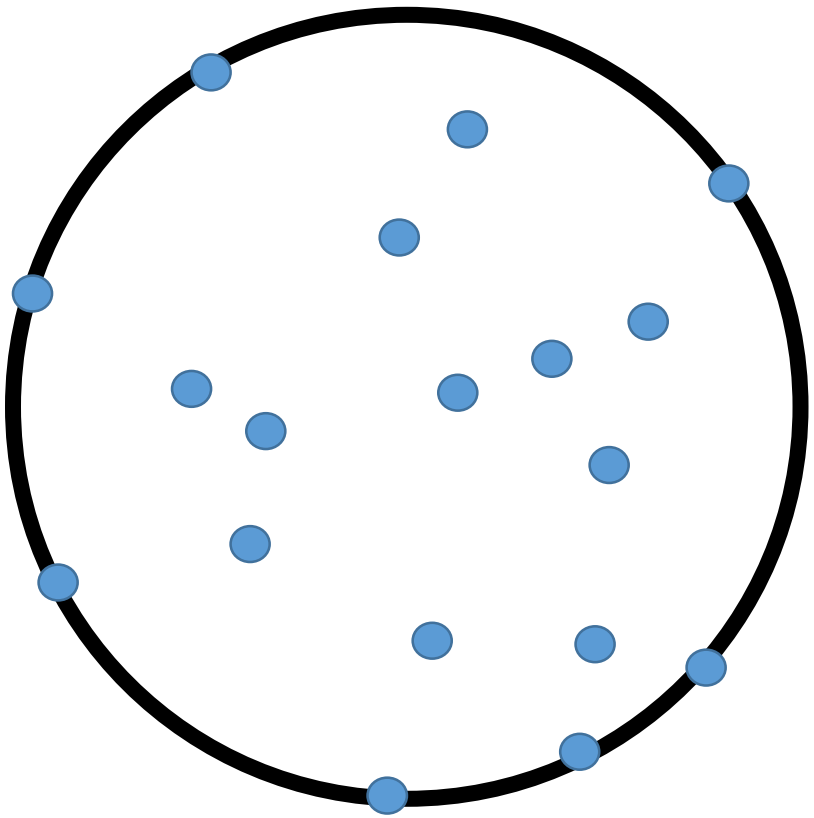
Generate a set of random points uniformly inside a disk (inclusive) with radius r in the two dimensional space. Store the set of points in an array(s).



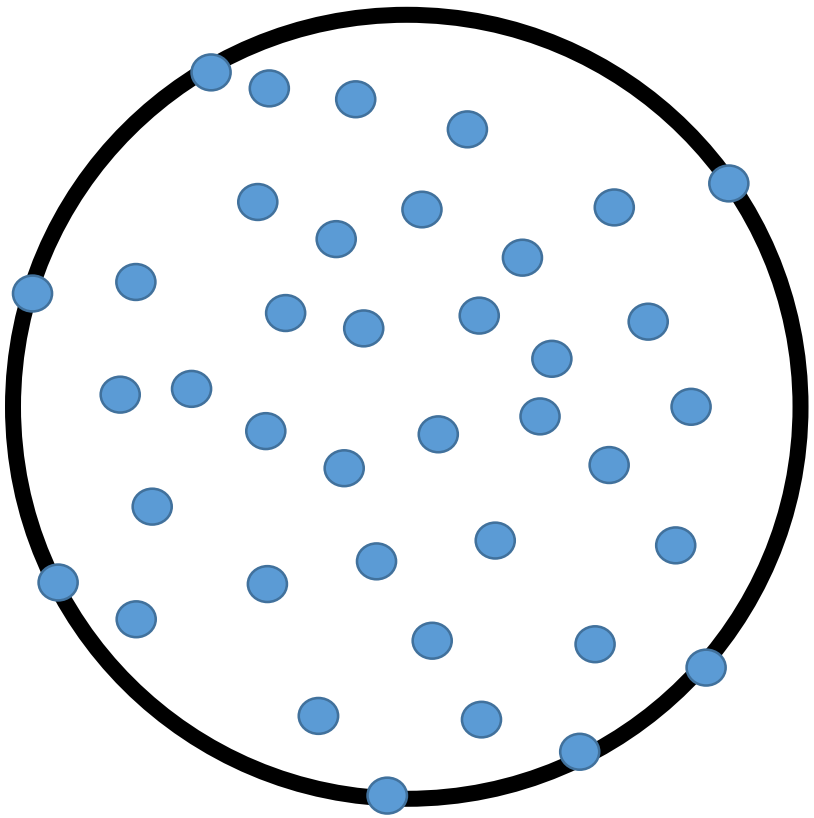
Generate a set of random points uniformly inside a disk (inclusive) with radius r in the two dimensional space. Store the set of points in an array(s).



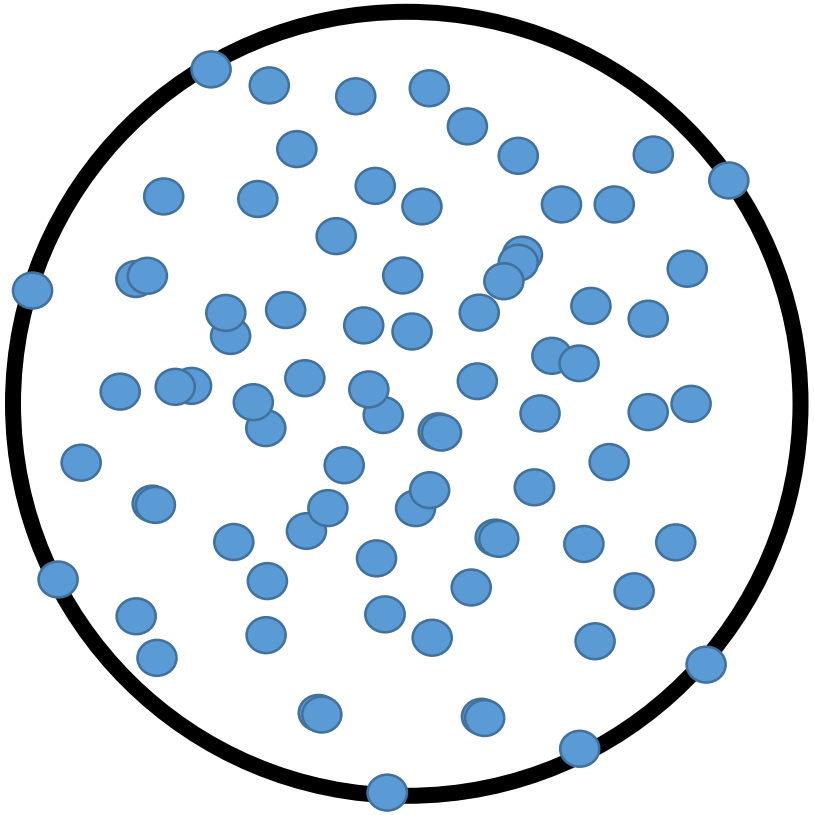
Generate a set of random points uniformly inside a disk (inclusive) with radius r in the two dimensional space. Store the set of points in an array(s).



Generate a set of random points uniformly inside a disk (inclusive) with radius r in the two dimensional space. Store the set of points in an array(s).



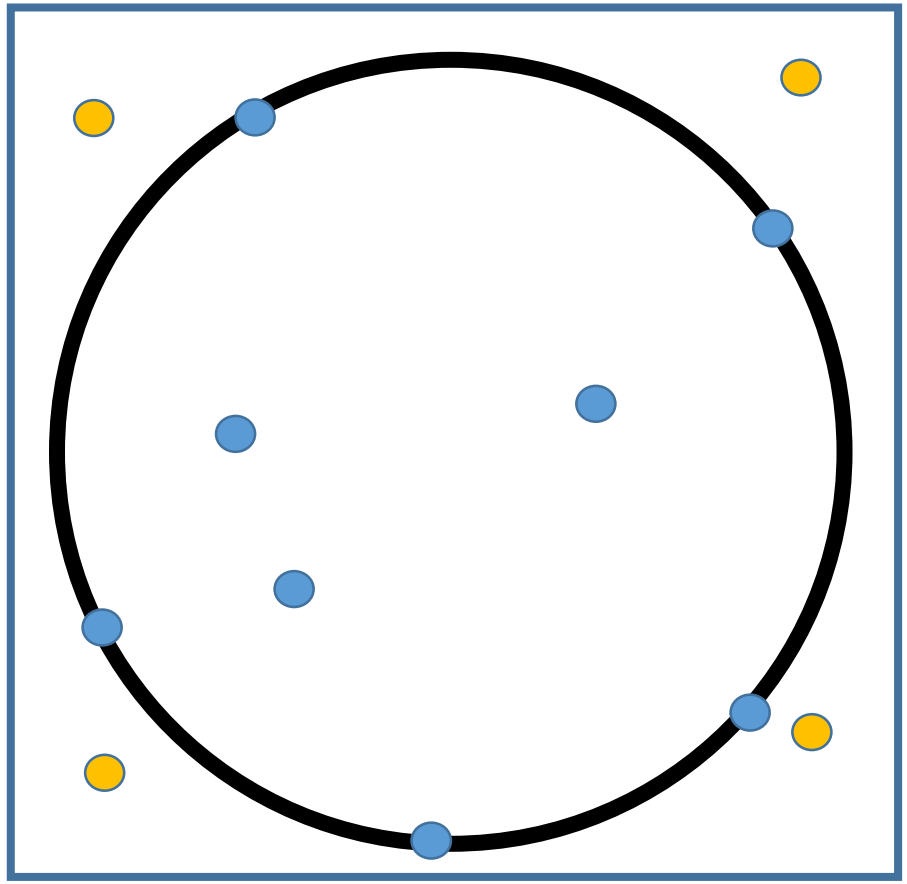
Generate a set of random points uniformly inside a disk (inclusive) with radius r in the two dimensional space. Store the set of points in an array(s).



Generate a set of random points uniformly inside a disk (inclusive) with radius r in the two dimensional space. Store the set of points in an array(s).

Algorithm:
Create a rectangle enclosing the circle.

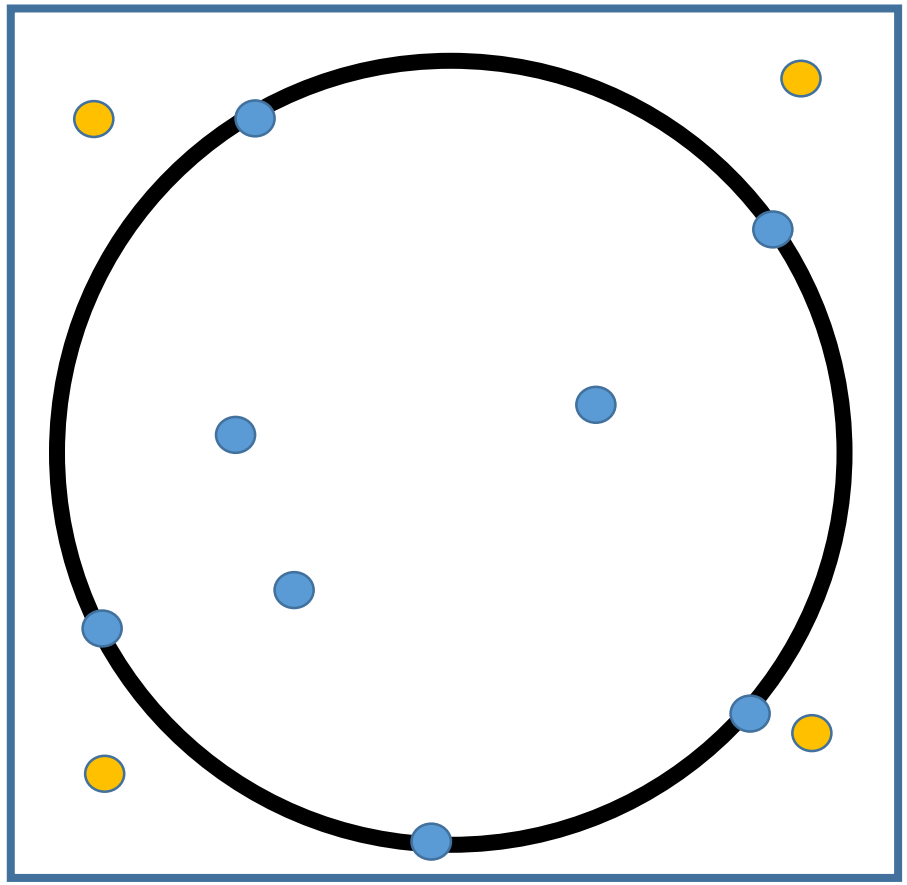
```
while true
   Generate a sample point inside the rectangle.
    if the sample point is inside the circle
            store the sample point
    update state
    if termination condition is satisfied, quit
```



Generate a set of random points uniformly inside a disk (inclusive) with radius r in the two dimensional space. Store the set of points in an array(s).

Algorithm:
Create a rectangle enclosing the circle.

```
while true
   Generate a sample point inside the rectangle.
    if the sample point is inside the circle
            store the sample point
    update state
    if termination condition is satisfied, quit
```



Do you know another way to randomly generate points inside a circle in a uniform manner?

Intended Learning Outcomes

- List some properties of functions
- Describe how to use vectors, characters, and strings
- List some math functions
- Describe the mechanism to write to a file or read from a file in the text mode.
- Implement functions to generate random numbers

Supplemental Material

Mathematical Functions

cmath header: common mathematical functions.

```
#include <cmath>
```

Character Functions

`isdigit(ch)`	Returns true if the specified character is a digit.
`isalpha(ch)`	Returns true if the specified character is a letter.
`isalnum(ch)`	Returns true if the specified character is a letter or digit.
`islower(ch)`	Returns true if the specified character is a lowercase letter.
`isupper(ch)`	Returns true if the specified character is an uppercase letter.
`isspace(ch)`	Returns true if the specified character is a whitespace character.
`tolower(ch)`	Returns the lowercase of the specified character.
`toupper(ch)`	Returns the uppercase of the specified character.

Trigonometric Functions

Trigonometric Functions

- sin(radians) Returns the trigonometric sine of an angle in radians.
- cos(radians) Returns the trigonometric cosine of an angle in radians.
- tan(radians) Returns the trigonometric tangent of an angle in radians.
- asin(a) Returns the angle in radians for the inverse of sine.
- acos(a) Returns the angle in radians for the inverse of cosine.
- atan(a) Returns the angle in radians for the inverse of tangent.

Exponent Functions

- exp(x) Returns e raised to power of x, i.e., e^x
- log(x) Returns the natural logarithm of x, i.e., $\ln(x) = \log_e(x)$
- log10(x) Returns the base 10 logarithm of x, i.e., $\log_{10}(x)$
- pow(a, b) Returns a raised to the power of b, i.e., a^b
- sqrt(x) Returns the square root of x

Read Characters

To read a character from the keyboard

```
cout << "Enter a character: ";
char ch;
cin >> ch;
```

Develop a system which supports five options. Write a program to ask to input x and y. Then ask to input an option. After that, perform the option and show the result.

1. The cost of a book is x dollars. You have y dollars.
How many books that you can buy?

2. There are x people. A car can carry y people.
How many cars do we need to carry all the people?

3. There are two values x and y. Compute the maximum value.

4. There are two values x and y. Compute the minimum value.

5. Compute the absolute difference between x and y.

Develop a system which supports five options. Write a program to ask to input x and y. Then ask to input an option. After that, perform the option and show the result.

1. The cost of a book is x dollars. You have y dollars.
How many books that you can buy? floor(y/x), e.g., floor(5/2) = 2

2. There are x people. A car can carry y people.
How many cars do we need to carry all the people? ceil(y/x), e.g., ceil(5/2) = 3

3. There are two values x and y. Compute the maximum value. max(x,y)

4. There are two values x and y. Compute the minimum value. min(x,y)

5. Compute the **absolute** difference between x and y. abs(x-y)

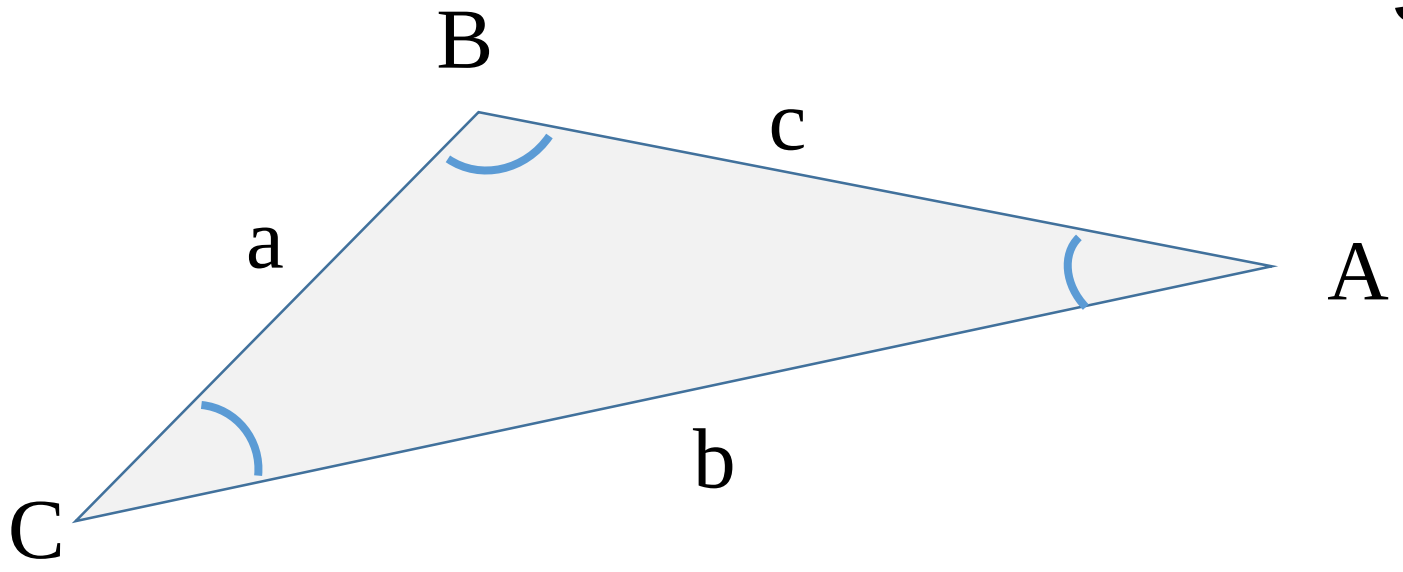
Convertion between integers and chars

```
int i = 'a';
// Same as int i = static_cast<int>('a');


char c = 97;
// Same as char c = static_cast<char>(97);
```

Read the ASCII code table

About triangles



Heron's formula

$$T = \sqrt{s(s-a)(s-b)(s-c)}$$

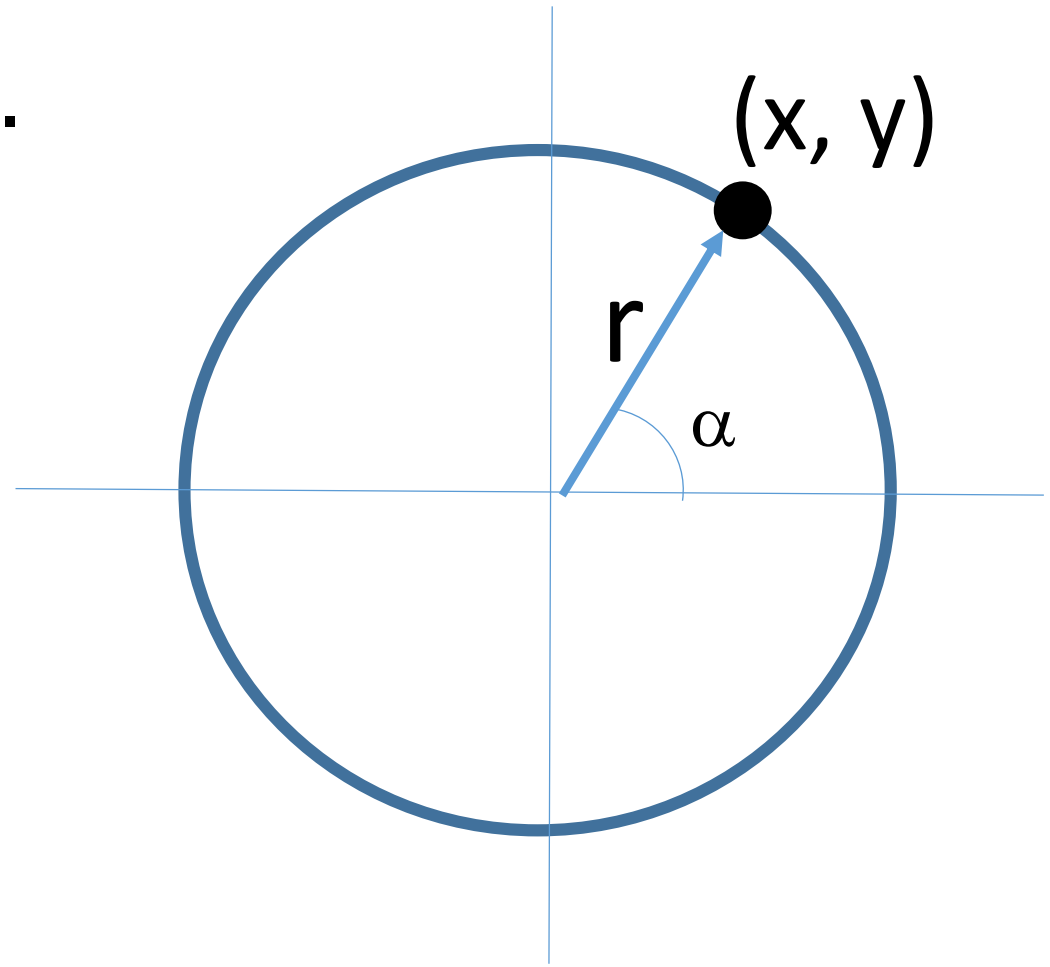
$$s = \frac{a+b+c}{2}$$

Write a program to compute the area of a triangle.

- Prompt the user to enter the x- and y-coordinates of the three corner points
- Display the side lengths of the triangle
- Display the triangle angles
- Display the triangle area

Generate a set of random points **uniformly** inside a disk (inclusive) with radius r in the two dimensional space.

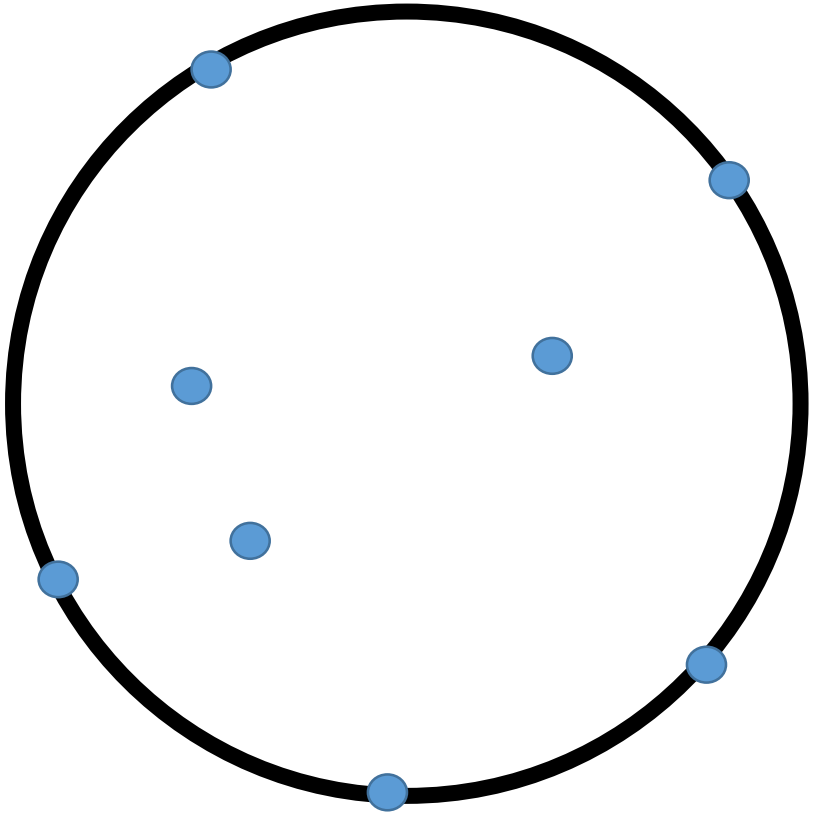
Store the set of points in array(s).



Generate a set of random points uniformly inside a disk (inclusive) with radius r in the two dimensional space. Store the set of points in array(s).

- $x = a \cos(\theta)$
- $y = a \sin(\theta)$

- Sample a uniformly inside [0, r]
- Sample θ uniformly inside $[0, 2\pi)$



Does it work? Please try it.

If it does not work, how to generate the set of the random points uniformly?

Exercise
Converting a Hexadecimal Digit to a Decimal Value

Write a program that converts a hexadecimal digit into a decimal value.

e.g.

‘9’ -> 9

‘A’ - > 10

‘B’ -> 11

‘F’ -> 15

Formatting Console Output

Manipulator	Description
setprecision	set the precision of a floating-point number
fixed	Display floating-point numbers in fixed-point notation
showpoint	…
setw(width)	
left	
right	

Please find more detail ……

Escape Sequences for Special Characters

Character escape sequence	Name	ASCII Code
\b	Backspace	8
\t	Tab	9
\n	Linefeed	10
\f	Formfeed	12
\r	Carriage Return	13
\\	Backslash	92
\'	Single Quote	39
\"	Double Quote	34

ASCII Character Set

Exercise: Study the ASCII Code

Examples: 0x31 = ‘1’, digit 1
0x61 = ‘a’, letter a

What is the ASCII code of ‘A’?

Exercise: Converting a Lowercase to Uppercase

Write a program

- prompt the user to enter a lowercase letter
- find its corresponding uppercase letter
- And vice versa

Numeric Operators on Characters

```
cout << j << " is the ASCII code for character " << static_cast<char>(j) << endl;
```

poor style

```
cout   << j
       << " is the ASCII code for character "
       << static_cast<char>(j)
       << endl;
```

Good style

showpoint Manipulator

```
cout << setprecision(6);
cout << 1.23 << endl;
cout << showpoint << 1.23 << endl;
cout << showpoint << 123.0 << endl;
```

displays

1.23

1.23000

123.000

Simple File Input

Read data from a file

```
 // declare a variable of the ifstream type
ifstream input;
// specify a file. Invoke the open function
input.open("numbers.txt");
```

An alternative to open a file (or create the file at the same time):

```
ifstream input("numbers.txt");
```

To read data, use the stream extraction operator (**>>**).

For example,

```
input << score1 << score2 << score3;
```