Functions I

Functions must be declared, defined and called.

Function Return DataType Function Name(Parameters);

Example

```
void  Populate(Parameter 1, Parameter 2);
```

Parameter Definition Consists of :

DataType Parameter Name

Example

```
int    number;
float  realvalue;
```

Flow of Control

While Loop

Loop the code block if condition is true.

Condition is checked prior to Loop – will execute if true

Loop exits when condition is false

```
initialize some condition; (sometime prior to loop)

while (some condition)

{
        CODE BLOCK
    update condition value;
}
```

Do While Loop

Loop the code block at least once.

Condition checked at end of Loop – will re-execute if true

Loop exits when condition is false

```
initialize some condition; (sometime prior to loop)
do
{
    CODE BLOCK
    update condition value;
} while(some condition);
```

For Loop

Set an initial condition value, check if value condition is true, and update the condition. Loop the code block if condition is true.

```
for(initialize condition; check condition; update condition value)
{
        CODE BLOCK
}

for (int i = 1; i <= 10; i=i+1)
        {
        cout << i << "  ";
        }
cout endl;
```

Break Statement

Break statement is used to exit a loop prior to completion.

Continue Statement (only in for loops).

Continue statement is used to skip the current iteration and jump to the next iteration in the loop.

Nested Loops

```
/*      Before First Row      */
for (int row = 1; row<= 10; row=row+1)
        {
            /*     Before First Column   Processing             */
        for (int col = 1; col <= 10; col = col + 1)
                {
                      /*     Column Processing      */
                 }
            /*     After Last  Column   Processing  endl maybe               */
        }
/*      After Last t Row      */
```

Branching

Logical Operators

&&	AND operator

```
bool b = (w == x) && (y == z)
returns a Boolean value
```

||	OR operator

```
bool b = (w == x) || (y == z);
returns a Boolean value
```

!	NOT operator

```
bool b = !(w == x)
returns a Boolean value
```

if Statements

Single if Statement

if an expression evaluates as true, do something

```
single line:
    if (Boolean condition)
        do something;
multi line:
    if (Boolean condition)
        {
        code block;
        }
```

if-else

if an expression evaluates as true, do something,
otherwise do something else

```
    if (Boolean condition)
        {
        True condition code block;
        }
    else
        {
        False condition code block
        }
```

multi-way if-else

```
if an expression evaluates as true, do something,
otherwise
    if a different expression evaluates as true,
        do something else

    if (Boolean condition)
        {
        True code block
        }
    else
    if (another Boolean condition)
        {
        True code block
        }
    else
        {
        False code block
        }
```

nested if

if-else and else-if statements can be nested

In this example the italicized text highlights the nested portion of the code.

```
if (Boolean condition)
    {
    if (other Boolean condition)
        {
        True code block;
        }
    else
        {
        False code block
        }
    some other statement;
```

```
                 }
```

Truth Tables

AND Logic &&	--	True if both Conditions are True
OR Logic ||	--	True if either Condition is True
Not Logic !	--	If Condition is true then False If Condition is false then Truth

And Logic Truth Table

Condition 1	Condition 2	Result
T	T	T
T	F	F
F	T	F
F	F	F

OR Logic Truth Table

Condition 1	Condition 2	Result
T	T	T
T	F	T
F	T	T
F	F	F

NOT Logic Truth Table

Condition 1	Result
T	F
F	T

Utility Function

- Requires #include <cstdlib>
- Exit Program – exit(int i) – void function

Math Functions

- Requires #include <cmath> or #include <cstdlib>
    - o Square Root- sqrt(double x) - returns a double
    - o Exponent – pow(double x, double e) - returns a double
    - o Absolute Value – abs(int x) - returns an int
    - o Ceiling – ceil(double x)- returns a double; jumps up
    - o Floor - floor(double x)- returns a double; drops down
    - o Round - round(double x)-round up or down to the nearest integer

Pseudorandom numbers

- o rand() – returns an int
- o srand() – sets the starting seed
- o srand(time(NULL))
    - set the starting seed using the current time
    - requires #include <time.h>

```
int num1 = rand() % <Num Values Needed> + <Starting Value>
Num Values Needed = Ending Value – Starting Value + 1
```

Order of Precedence

Priority 1

.	dot operator
[]	array index
( )	function call
n++	postfix increment operator
n--	postfix decrement operator
static_cast	cast operation

Priority 2 (Right-to-Left)

++n	prefix increment operator
--n	prefix decrement operator
!	not
-	unary minus
+	unary plus

Priority 3

*	multiply
/	divide
%	modulo

Priority 4

+	addition
-	subtraction

Priority 5

<<	insertion operator
>>	extraction operator

Priority 6

<	less than
>	greater than

Priority 7

==	equivalence
!=	equivalence negation

Priority 8

&&	and

Priority 9

||	or

Priority 10 (Right-to-Left)

=	assignment
+=	add and assign
-=	subtract and assign
*=	multiply and assign
/=	divide and assign
%/	modulo and assign

Priority 11

,	comma

Identifiers

• Primitive Data Types (Variables)
Case sensitive
Must start with a letter or underscore
Can use letters, digits or underscore are acceptable

Types
integer
short: -32,768 - 32,767
int: -2,147,483,648 - 2,147,483,648
long: -2,147,483,648 - 2,147,483,648

floating point
float: 10-38 to 1038
double: 10-308 to 10308

char
any single ASCII character
see ASCII table

bool
1 or 0 represents true or false
Any number aside from 0 will evaluate as true.

Class Data Type (Variables)
string
Requires #include <string>
Includes a variety of string manipulation options

Constants
constant variables are named in upper case
Syntax: const var_type var_name = literal;
Example: const string PAYDAY = “Friday”;

Casting

```
      converting data types
            Syntax: type variable Name = static_cast<type>(data);
                  int x = 5;
                  double quotient = static_cast<double>(x);

      Used in converting cases  Upper to Lower  & Lower to Upper
```

String Variables

o String variables are used to store string literals.
Recall that a string literal is composed of one or more characters.
Characters:
‘A’, ‘b’, ‘&’, ‘_’, ‘\n’, ‘ ‘
o A string literal can be **assigned** or stored into a string variable using the following syntax:
string name = “John”;

String Functions

o A **function** is a group of related commands which can be executed on demand.
Main is the primary function of a C++ program. Executing a function is referred to as a **function call.** Each function has a **name** and can accept input through **parameters.** Finally, functions may **return** data (a result of the commands) to the calling function.

o There are a variety of functions available for string objects, some of which are described here. To insure compatibility across multiple compilers the **string library** should be included at the top of the program to support these functions (in addition to **iostream**).

```
#include <string>
#include <iostream>
using namespace std;
int main {…}
```

at() function.
This function will issue a runtime error if an attempt is made to access an illegal index.

```
string name = “John Smith”;
cout << name(100) << endl; // illegal access, no error(not safe)
string name = “John Smith”;
cout << name.at(100) << endl; // illegal access, with error
(safe)
```

Strings can be “added” or concatenated using the plus operator +:

```
string a = “Hello”;      string b = “ Goodbye”;
     string c = a + b;   // c is now “Hello Goodbye”
cout << c << endl;
```

A subset of a string can be extracted using the substr() function. This function has two input parameters, the starting index and the number of characters to include (from the start index on). It returns the new substring based upon the input specifications. Recall that the first character has an index of 0.

```
string s = “This is a string.”;
cout << s.substr(5, 4) << endl;
```

Text can be inserted into a string using the insert() function. This function accepts two input parameters, the starting index and the string to be inserted at that location.

```
string s = “John Smith”;
cout << s.insert(5,”Jay “) << endl;
```

Text can be inserted at the end of the string using the append() function.

```
string s = “John Smith”;
cout << s.append(” Jr.“) << endl;
```

Text can be replaced using the replace() function. This function accepts three parameters, the start index, the number of characters to replace (from start index on) and the new string that will replace the specified text.

```
string s = “John Jay Smith”;
cout << s.replace(5,3, “Joseph”) << endl;
```

Text can be erased using the erase() function. This function accepts two parameters, the start index and the number of characters to erase from the string.

```
string s = “John Joseph Smith”;

cout << s.erase(5,7) << endl;
```

ASCII Manipulation

Upper Case A = Decimal 65	**Lower Case a = Decimal 97**
Upper Case Z = Decimal 90	**Lower Case z = Decimal 122**

Convert the value of a char c into its ASCII integer equivalent, then store in an integer i

```
    i = static_cast<int>(c);
```

Convert the value of an integer i into a char, then store in c

```
    c = static_cast<char>(i);
```

Console Input

- Requirements
  - **#include <iostream>**
- cin
  - **cin >> variable**;
- simple console input into a string variable
  - **string day**; // *declare a variable called day*
  - **cin >> day**; // *request console input, store into variable*
  - **cout << day**; // *printout the value stored in the variable*

Column Output

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

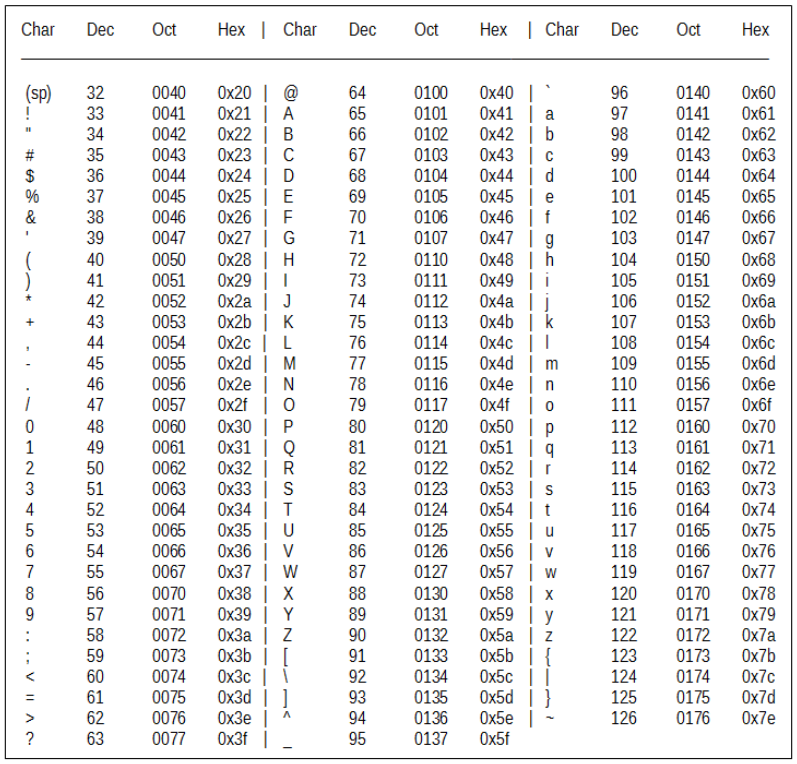
Number Format

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(x);
       x specifies the number of decimal places
setw(y)
       y specifies the number of characters in the column

float f1 = 4.5678;
float f2 = 6.2;
cout << setw(10) << setprecision(3) << f1 << setw(8) << setprecision(1) <<
f2;
```

Basic Console Output

- Requirements
  - **#include <iostream>**
  - This library provides console input and output support.
- **cout**
  - cout << "Some text";
- **endl**
  - cout << "Some text" << endl;
- **\n**
  - cout << "Some text\n";
  - cout << some variable << endl;
- **\t**
  - cout << "The score is:\t 50\n";



Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
?	63	0077	0x3f	_	95	0137	0x5f				