DATA TYPES

C Programming deals with several basic types of data: integers, characters, and floating point types are a few examples. Declaring a variable to be a certain type makes it possible for the computer to store, fetch, and interpret the data correctly.

Below are some examples of some declarations of such data types. Shown first is the variable declarations without initialization followed by the variable declaration with initialization in the declaration.

```
int number;
OR
int number = 0;

char answer;
OR
char answer = 'y';

float amount;
OR
float amount = 3.75;
```

The table below shows more of the basic data types along with their sizes in bytes.

C Programming Data Type	Size in Bytes
char	1
unsigned char	1
short int	2
unsigned short int	2
int	4
unsigned int	4
long int	4
unsigned long int	4
long long int	8
unsigned long long int	8
float	4
double	8
long double	12

Below are some example programs that use some of the C Programming data types.

INSTRUCTOR: Joe Russell
EMAIL: joe_programming@yahoo.com

/* Program to demonstrate declaration of some of the C Data Types and display their sizes in bytes */

```
#include <stdio.h>

int main(void)
{
  char a;
  short int b;
  int c;
  long int d;
  long long int e;
  float f;
  double g;
  long double h;

  printf("Size of char is: %d bytes\n", sizeof(char));
  printf("Size of short int is: %d bytes\n", sizeof(short int));
  printf("Size of int is: %d bytes\n", sizeof(int));
  printf("Size of long int is: %d bytes\n", sizeof(long int));
  printf("Size of long long int is: %d bytes\n", sizeof(long long int));
  printf("Size of float is: %d bytes\n", sizeof(float));
  printf("Size of double is: %d bytes\n", sizeof(double));
  printf("Size of long double is: %d bytes\n", sizeof(long double));

  return 0;
}
```

Sample Run:

```
Size of char is: 1 bytes
Size of short int is: 2 bytes
Size of int is: 4 bytes
Size of long int is: 4 bytes
Size of long long int is: 8 bytes
Size of float is: 4 bytes
Size of double is: 8 bytes
Size of long double is: 12 bytes
```

```
/* Program that displays temperature at which water freezes */
#include <stdio.h>
int main(void)
{
      int freezePoint;

      freezePoint = 32;
      printf("Water freezes at %d degrees F.\n", freezePoint);

      return 0;
}
```

```
Sample Run:
Y:\apc-cc-conduit\images>a
Water freezes at 32 degrees F.
```

INSTRUCTOR: Joe Russell
EMAIL: joe_programming@yahoo.com

~~~~~~

SYMBOLIC CONSTANT (MACRO)
It is not a good idea to program hard-coded constant values (as done above) into a program. It makes the program difficult to maintain and is often error-prone. A #define line allows for a symbolic name that is a particular string of characters. The preprocessor performs a string substitution into your code prior to compilation.

```
/* Program that displays temperature at which water freezes */
#include <stdio.h>

#define FREEZE_POINT 32
int main(void)
{
      printf("\nWater freezes at %d degrees F.\n", FREEZE_POINT);

      return 0;
}
```

~~~~~~

DECLARED CONSTANT
The preferred method is to use a declared constant. A declared constant uses the const type qualifier. This establishes an actual variable whose value cannot be modified by assignment or by incrementing or decrementing. It is also useful to have a declared constant when using debuggers because the variable is built into the symbolic table that the debugger uses.

```
/* Program that displays temperature at which water freezes */
#include <stdio.h>

const int FREEZE_POINT = 32;
int main(void)
{
      printf("\nWater freezes at %d degrees F.\n", FREEZE_POINT);

      return 0;
}
```

~~~~~~

VARIABLES:
C Programming variable names are case sensitive and must begin with a letter or an underscore. The entire variable name may contain letters, underscores, and numbers thereafter. ANSI C calls for 31 characters maximum length. If you use more than 31 characters, the compiler ignores those characters beyond the 31st character.

A variable belongs to one of five storage classes as shown below.

Storage Class	**Keyword**	**Duration**	**Scope**
Automatic	auto	Temporary	Local
Register	register	Temporary	Local
Static	static	Persistent	Local
External	extern	Persistent	Global
External Static	extern static	Persistent	Global

Note: The keyword extern is used only to redeclare variables that have been defined externally elsewhere; the act of defining the variable outside of a function makes it external. The general form for declaring a scalar variable is:

storage-class type-specifier variable-name;

As a reminder, you should use meaningful and descriptive names for your variables as shown below.

GOOD VARIABLE NAME CHOICES	**POOR VARIABLE NAME CHOICES**
averageGrade	a
mpg	xx
daysPerWeek	d7

```
/* Program to calculate the area of a circle.  User enters radius */
#include <stdio.h>

const float PI = 3.14159;
int main(void)
{
      float radius, area;

      printf("\nEnter the circle radius: ");
      scanf("%f", &radius);

      area = PI * radius * radius;
      printf("The area = %f\n", area);

      return 0;
}
```

```
Sample Run:
Y:\apc-cc-conduit\images>a

Enter the circle radius: 1
The area = 3.141590
```

INSTRUCTOR: Joe Russell
EMAIL: joe_programming@yahoo.com
~~~~~~

```
Y:\apc-cc-conduit\images>a

Enter the circle radius: 2.7
The area = 22.902193

Y:\apc-cc-conduit\images>
```

~~~~~~

ARITHMETIC OPERATORS:
Below is a table of the C Programming arithmetic operators in the correct order of precedence from highest to lowest. We will use these operators in arithmetic expressions.

Operator	Associativity
( )	Left to Right
++ (increment), --(decrement), - (unary), + (unary)	Right to Left
* / %	Left to Right
+ -	Left to Right

```
/* Program to convert Fahrenheit to Celsius */
#include <stdio.h>

int main(void)
{
      float fahrenheit, celsius;

      printf("\nEnter a temperature in degrees Fahrenheit: ");
      scanf("%f", &fahrenheit);

      celsius = (5.0 / 9.0) * (fahrenheit – 32);
      printf("The temperature in degrees Celsius = %.0f\n", celsius);

      return 0;
}
```

```
Sample Run (Note, the result is rounded up from 23.88888... to 24):
Y:\apc-cc-conduit\images>a

Enter a temperature in degrees Fahrenheit: 75
The temperature in degrees Celsius = 24
```

```
/* Program to convert Celsius to Fahrenheit */
#include <stdio.h>

int main(void)
{
      float fahrenheit, celsius;

      printf("\nEnter a temperature in degrees Celsius: ");
```
~~~~~~

```
      scanf("%f", &celsius);

      fahrenheit = (9.0 / 5.0) * celsius + 32;
      printf("The temperature in degrees Fahrenheit = %.0f\n",
            fahrenheit);

      return 0;
}
```

```
Sample Run:
Y:\apc-cc-conduit\images>a

Enter a temperature in degrees Celsius: 100
The temperature in degrees Fahrenheit = 212
```

The operators above are examples that use the +, -, /, and * C operators. The modulus operator, %, is an operator that returns a remainder. In the example below, random numbers are generated using the pseudo-random-number generator defined in the ANSI C Standard. Notice how the modulo operator is used to provide ranges for the last three random numbers. You should include stdlib.h to use the rand() function.

```
/* Program to generate and print five random numbers */
#include <stdio.h>
#include <stdlib.h>
int main()
{
      unsigned int seed, num1, num2, num3, num4, num5;

      /* Seed the random number generator */
      printf("Enter a random number generator seed (integer): ");
      scanf("%d", &seed);
      srand(seed);

      /* Get 5 random numbers */
      num1 = rand();          /* Number between 0 and RAND_MAX */
      num2 = rand();          /* Number between 0 and RAND_MAX */
      num3 = rand() % 100;    /* Number between 0 and 99 */
      num4 = rand() % 5;      /* Number between 0 and 4 */
      num5 = rand() % 2;      /* Number between 0 and 1 */

      printf("\nHere are 5 random numbers: %d, %d, %d, %d, %d\n",
            num1, num2, num3, num4, num5);

      return 0;
}
```

```
Sample Run:
Y:\apc-cc-conduit\images>a
Enter a random number generator seed (integer): 28

Here are 5 random numbers: 833668133, 497351162, 27, 2, 1

Y:\apc-cc-conduit\images>a
Enter a random number generator seed (integer): 45698

Here are 5 random numbers: 1228656019, 969638352, 57, 1, 1

Y:\apc-cc-conduit\images>a
Enter a random number generator seed (integer): 99507

Here are 5 random numbers: 210123376, 1570058729, 2, 2, 0

Y:\apc-cc-conduit\images>a
Enter a random number generator seed (integer): 45698

Here are 5 random numbers: 1228656019, 969638352, 57, 1, 1

Y:\apc-cc-conduit\images>a
Enter a random number generator seed (integer): 0

Here are 5 random numbers: 12345, 1406932606, 75, 4, 1

Y:\apc-cc-conduit\images>a
Enter a random number generator seed (integer): -5

Here are 5 random numbers: 924887064, 112136817, 30, 1, 0
```

This technique is useful in program simulations where tolerances are considered as a part of an analysis. For example, a 1 kOhm resistor with 10% tolerance varies from 900 kOhm to 1100 kOhm. An example below simulates a measurement of 10 different resistors taken from a larger sample batch.

```
/*
 * Program to simulate 10 measurements of 10 different
 * 1 kOhm resistors taken from a large batch.  Each resistor
 * is assumed to have a 10% tolerance.  Thus, the maximum delta
 * in the resistance, due to the tolerance, is 100 Ohms in this
 * example.
 */
#include <stdio.h>
#include <stdlib.h>

#define SIGN                  (rand() % 2 ? 1 : -1)
#define NOMINAL_RESISTANCE    1000
#define TOLERANCE_MAX         100
#define NUMBER_OF_RESISTORS   10
#define TOLERANCE_RESISTANCE  (SIGN * (rand() % (TOLERANCE_MAX + 1)))
```

INSTRUCTOR: Joe Russell
EMAIL: joe_programming@yahoo.com

```
int main()
{
      unsigned int seed, resistance, count;

      /* Seed the random number generator */
      printf("Enter a random number generator seed (integer): ");
      scanf("%d", &seed);
      srand(seed);

      /* Generate and print ten diffenent resistor values */
      for(count = 0; count < NUMBER_OF_RESISTORS; count++)
      {
            resistance = NOMINAL_RESISTANCE + TOLERANCE_RESISTANCE;
            printf("Resistor [%d] = %d Ohms\n", count, resistance);
      }

      return 0;
}
```

```
Sample Run #1:
Y:\apc-cc-conduit\images>a
Enter a random number generator seed (integer): 88
Resistor [0] = 961 Ohms
Resistor [1] = 935 Ohms
Resistor [2] = 927 Ohms
Resistor [3] = 994 Ohms
Resistor [4] = 961 Ohms
Resistor [5] = 915 Ohms
Resistor [6] = 967 Ohms
Resistor [7] = 995 Ohms
Resistor [8] = 997 Ohms
Resistor [9] = 974 Ohms
Sample Run #2:
Y:\apc-cc-conduit\images>a
Enter a random number generator seed (integer): 89
Resistor [0] = 1091 Ohms
Resistor [1] = 1087 Ohms
Resistor [2] = 1040 Ohms
Resistor [3] = 1000 Ohms
Resistor [4] = 1010 Ohms
Resistor [5] = 1094 Ohms
Resistor [6] = 1076 Ohms
Resistor [7] = 1057 Ohms
Resistor [8] = 1047 Ohms
Resistor [9] = 1015 Ohms
```

The increment and decrement operators are defined here for completeness. They will be most useful to us for flow control, arrays, and pointers (all to be discussed at a later time). There are two types of increment and decrement operators: prefix and postfix.

The prefix increment operator will increment the variable by one in the expression before it is actually used in the expression. The prefix decrement operator will likewise decrement the variable by one in the

expression before it is actually used in the expression. Usually, "one" means the integer "1". However, if it is a pointer that we are incrementing/decrementing, the value of the address that the pointer points to is incremented/decremented by the size of the data type to which the pointer points. We will defer pointer arithmetic until later in the course and illustrate the use of pre/post increment/decrement operators in the example below.

```
/* Program to Illustrate Pre/Post Increment/Decrement Operators */
#include <stdio.h>
int main(void)
{
      int a=5, b=-3, c, d, e;

      /* Pre-Increment Operator */
      c = a + ++b;
      printf("a = %d, b = %d, c = %d\n", a, b, c);

      /* Pre-Decrement Operator */
      d = --b + --a * c;
      printf("a = %d, b = %d, c = %d, d = %d\n", a, b, c, d);

      /* Post-Increment Operator */
      e = d++ + a;
      printf("a = %d, d = %d, e = %d\n", a, d, e);

      /* Post-Decrement Operator */
      e = d++ + a--;
      printf("a = %d, d = %d, e = %d\n", a, d, e);

      return 0;
}
```

```
Sample Run:
a = 5, b = -2, c = 3
a = 4, b = -3, c = 3, d = 9
a = 4, d = 10, e = 13
a = 3, d = 11, e = 14
```

~~~~~~

INPUT AND OUTPUT (I/O):
printf() and scanf(), although not the only I/O functions you can use with C, are the most versatile. C originally left the implementation of I/O up to the compiler writers, however, in the interests of compatibility, various implementations all came up with versions of printf() and scanf(). However, occasional discrepancies would sometimes exist between implementations. The ANSI C standard now describes standard versions of these functions.

For now, we will examine a programming example that used the printf() and scanf() function for various data types. We will revisit I/O in greater detail when we discuss Formatted I/O.

```
/* Program to calculate the average MPH to travel to school */
#include <stdio.h>

const int MINS_PER_HR = 60; /* number of minutes in one hour */

int main(void)
{
      float distanceToSchool, avgMph, drivingTime;
      char firstName[30], middleInitial, lastName[30];

      printf(“Enter your first name: ”);
      scanf(“%s”, firstName);

      printf(“Hello %s!  \nEnter your middle initial: ”, firstName);
      scanf(“ %c”, &middleInitial); /* use “space” to avoid <CR> */

      printf(“And your last name? ”);
      scanf(“%s”, lastName);

      printf(“How many miles do you drive to get to school? ”);
      scanf(“%f”, &distanceToSchool);

      printf(“How many minutes does it take you to drive? ”);
      scanf(“%f”, &drivingTime);

      /* Calculate average miles-per-hour (must convert mins to hrs) */
      avgMph = distanceToSchool / (drivingTime / MINS_PER_HR);

      printf(“\n%s %c %s drives an average of %.1f MPH to school.\n”,
            firstName, middleInitial, lastName, avgMph);

      return 0;
}
```

```
Sample Run:
Y:\apc-cc-conduit\images>a
Enter your first name: Joe
Hello Joe!
Enter your middle initial: K
And your last name? Russell
How many miles do you drive to get to school? 7
How many minutes does it take you to drive? 32

Joe K Russell drives an average of 13.1 MPH to school.
```

INSTRUCTOR: Joe Russell
EMAIL: joe_programming@yahoo.com
~~~~~~