# Introduction to C

# Introduction to C

- A 'C' Program
  - Variable Declarations
  - printf ( )
- Compiling and Running a C Program
- Sizeof Program
  - #include
- What is True in C?
  - if example
- Another C Program
  - #define
  - scanf ( )

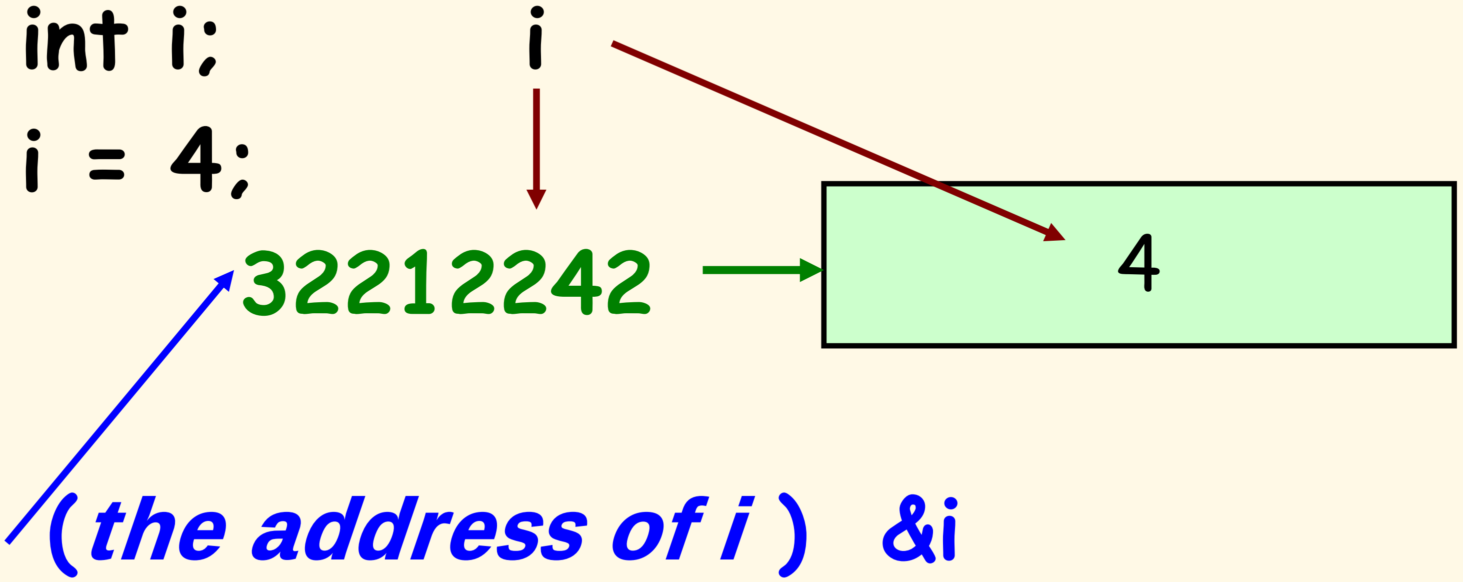
# Introduction to C

- Another C Program (continued)
  - for loop
  - Promotion
- Other C topics
  - Increment and Decrement Operators
  - Casting
  - Operator Precedence
  - Value of Assignment Operator

# Variables

- Variable names correspond to memory locations in memory. Every variable has a **type**, a **name** and a **value.**

```
int i;
i = 4;
```



# printf

```
int main()
{
    …
    printf("%d  %c\n", i , ch);
}
```

- Two components:
  - Formatting template {within quotes}
  - Argument list – variables separated by commas.

# printf

```
int main()
{
    …
    printf("%d  %f %c\n", i , fvar, ch);

}
```

**Formatting template:**

- **Argument list matches up with '%'**
- **Some of the argument types:**
  - %d integers
  - %f floating-point numbers
  - %c characters

# printf

```
int main()
{
    …
    printf("%4d %5f %6.2f\n", i, fvar, f2var);

}
```

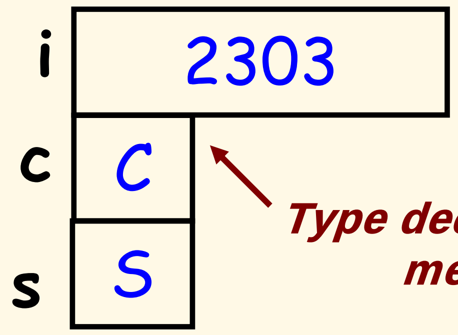
**Width of variable printing:**

- **%4d** – decimal integers at least 4 digits wide
- **%5f** – floating point at least 5 digits wide
- **%6.2f** – floating point at least 6 digits wide with at least 2 after the decimal point

# A Simple C Program

```
/* Example of a simple C Program */
int main()
{
  int i;
  float var;
  char c, s;
  i = 2303;
  c = 'C';
  s = 'S';
  printf("\nHello");
  printf(" %c%c  %d Students!!\n", c, s, i);
  return 0;
}
```

*Comments in C*



*Type declarations specify memory sizes*

## Compiling and Running simple

```
%ls
simple.c
%gcc simple.c
%ls
a.out simple.c
%./a.out

Hello CS  2303 Students!!
%
```

**Alternate Version**

```
%ls
simple.c
%gcc -o simple simple.c
%ls
simple simple.c
%./simple
```

# sizeof operator

```c
/* Fig. 7.17: fig07_17.c
   Demonstrating the sizeof operator */
#include <stdio.h>

int main( void )
{
   char c;
   short s;
   int i;
   long l;
   float f;
   double d;
   long double ld;
   int array[ 20 ];  /* create array of 20 int elements */
   int *ptr = array; /* create pointer to array */

```

preprocessor directive

Figure 7.17 (part 1)

# sizeof operator

```
   printf( "     sizeof c = %d\tsizeof(char)  = %d"
           "\n     sizeof s = %d\tsizeof(short) = %d"
           "\n     sizeof i = %d\tsizeof(int) = %d"
           "\n     sizeof l = %d\tsizeof(long) = %d"
           "\n     sizeof f = %d\tsizeof(float) = %d"
           "\n     sizeof d = %d\tsizeof(double) = %d"
           "\n    sizeof ld = %d\tsizeof(long double) = %d"
           "\n sizeof array = %d"
           "\n   sizeof ptr = %d\n",
          sizeof c, sizeof( char ), sizeof s, sizeof( short ), sizeof i,
          sizeof( int ), sizeof l, sizeof( long ), sizeof f,
          sizeof( float ), sizeof d, sizeof( double ), sizeof ld,
          sizeof( long double ), sizeof array, sizeof ptr );

   return 0; /* indicates successful termination */

} /* end main */
```

```
     sizeof c = 1       sizeof(char)  = 1
     sizeof s = 2       sizeof(short) = 2
     sizeof i = 4       sizeof(int) = 4
     sizeof l = 4       sizeof(long) = 4
     sizeof f = 4       sizeof(float) = 4
     sizeof d = 8       sizeof(double) = 8
    sizeof ld = 8       sizeof(long double) = 8
 sizeof array = 80
   sizeof ptr = 4
```

Figure 7.17 (part 2)

from typelen.c

|  |  |
|---|---|
| char | 1 |
| short | 2 |
| int | 4 |
| long | 4 |
| long long | 8 |
| float | 4 |
| double | 8 |
| long double | 12 |

# Conditional Testing for 'True'

```
/* check to see what conditional does with negative integers */

int main ()
{
   int i = 0;      /* zero is the only value for false in C */

   if (i) printf("%d = true\n", i);
     else
         printf("%d = false\n", i);
   i = 4;
   if (i) printf("Positive integer %d = true\n", i);
     else
        printf("Positive integer %d = false\n",
   i = -4;
   if (i) printf("Negative integer %d = true\n
     else
       printf("Negative integer %d = false\n"
  return 0;
}
```

```
$./a.out
0 = false
Positive integer 4 = true
Negative integer -4 = true
```

# Another C Program

```
#define SIZE 5
int main ()
{
   int i, start, finish;
   float celsius;

   scanf("%d", &start);
   finish = start + SIZE;
   for (i=start; i<finish; i++)
   {
      celsius = (5.0/9.0)* (i - 32.0);
      printf("%3d %6.1f\n", i, celsius);
   }
   return 0;
}
```

preprocessor directive

scanf needs the address

use of define

# Another C Program

```
#define SIZE 5
int main ()
{
   int i, start, finish;
   float celsius;

   scanf("%d", &start);
   finish = start + SIZE;
   for (i=start; i<finish; i++)
   {
      celsius = (5.0/9.0)* (i - 32.0);
      printf("%3d %6.1f\n", i, celsius);
   }
   return 0;
}
```

initial value

continue to loop if True

after each interation

# Another C Program

```
#define SIZE 5
int main ()
{
   int i, start, finish;
   float celsius;

   scanf("%d", &start);
   finish = start + SIZE;
   for (i=start; i<finish; i++)
   {
      celsius = (5.0/9.0)* (i - 32.0);
      printf("%3d %6.1f\n", i, celsius);
   }
   return 0;
}
```

example of 'promotion'

```
$./a.out
30
 30   -1.1
 31   -0.6
 32    0.0
 33    0.6
 34    1.1
```

# Other C Topics

- Increment and decrement operators
- Casting operator ( *type* )
- Operator precedence
- Danger :: the value of the assignment operator
- Variable scope
- Switch
- Conditional operator ?:

## Increment and decrement operators

| Operator | Sample expression | Explanation |
|---|---|---|
| `++` | `++a` | Increment **a** by 1, then use the new value of **a** in the expression in which **a** resides. |
| `++` | `a++` | Use the current value of **a** in the expression in which **a** resides, then increment **a** by 1. |
| `--` | `--b` | Decrement **b** by 1, then use the new value of **b** in the expression in which **b** resides. |
| `--` | `b--` | Use the current value of **b** in the expression in which **b** resides, then decrement **b** by 1. |

**Fig. 3.12**

**Increment and decrement operators**

# casting

- Cast is a unary operator.
- Cast is often useful when an iteration index is used in mixed type arithmetic.
- Later, it will be important to make sure arguments passed are properly matched between called and calling routines.

Example:

```
int total, count;
float average;
…
average = (float) total / counter;
```

***When in doubt, be conservative and use cast to be sure!***

# Fig 4.16 Operator Precedence

| Operators | Associativity | Type |
|---|---|---|
| `++` *(postfix)* `--` *(postfix)* | right to left | postfix |
| `+ - ! ++` *(prefix)* `--` *(prefix)* `(`*type*`)` | right to left | unary |
| `* / %` | left to right | multiplicative |
| `+ -` | | additive |
| `< <= > >=` | left to right | relational |
| `== !=` | | equality |
| `&&` | left to right | logical AND |
| `\|\|` | | logical OR |
| `?:` | right to left | conditional |
| `= += -= *= /= %=` | right to left | assignment |
| `,` | left to right | comma |

cast

arithmetic

boolean

logical

# Value of Assignment

- The value of assignment is the same as the contents deposited into the variable type on the left.
- Note: There are several potential dangers here – especially when the programmer creates new types!!

Examples (for now):

```
    if ( i = 0 )              if ( i == 0)
    if ( i = 4 )              if ( i == 4)
```

What is the problem ??

# Review/Summary

This presentation covers many important C topics quickly including:

- Declaration of variable types
  - memory allocation by type
  - The address of a variable **&**
- **printf ( )** , **scanf ( )**
- C arithmetic (operators, precedence, casting, promotion, assignment value)
- C booleans (true and false)
- **if**
- Preprocessor directives
  - **#define, #include**
- **for**

You are now ready to due lab 1 and once we cover functions everyone should be able to due Program 1.