C (Basic) Identifiers

Identifiers

- A 'C' program consist of two types of elements , user defined and system defined. Identifiers is nothing but a name given to these elements.
- An identifier is a word used by a programmer to name a variable , function, or label.
- identifiers consist of letters and digits, in any order, except that the first character or label.
- Identifiers consist of letters and digits if any order, except that the first character must be letter.
- Both Upper and lowercase letters can be used

C (Basic) Keywords

- Keywords are nothing but system defined identifiers.
- Keywords are reserved words of the language.
- They have specific meaning in the language and cannot be used by the programmer as variable or constant names
- C is case senitive, it means these must be used as it is
- 32 Keywords in C Programming

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Constants

- A 'C' program consist of two types of elements , user defined and system defined. Idetifiers is nothing but a name given to these eleme
- nts.
- An identifier is a word used by a programmer to name a variable , function, or label.
- identifiers consist of letters and digits, in any order, except that the first charecter or lable.
- Identifiers consist of letters and digits if any order,except that the first charecter must be letter.
- Both Upper and lowercase letters can be used

- A constant is a value or an identifier whose value cannot be altered in a program. For example: 1, 2.5,
- As mentioned, an identifier also can be defined as a constant. eg. const double PI = 3.14
- Here, PI is a constant. Basically what it means is that, PI and 3.14 is same for this program.

Integer constants

- A integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:

- decimal constant(base 10)
- octal constant(base 8)
- hexadecimal constant(base 16)

Escape Sequences

Sometimes, it is necessary to use characters which cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc. In order to use these characters, escape sequence is used.

- For example: \n is used for newline. The backslash (\ ) causes "escape" from the normal way the characters are interpreted by the compiler. Escape

Sequences	Character
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

C (Basic) Data Types

-different data representations need different types in programming-

C BASIC (DATA) TYPES

In C, data type categorized as:

1. Primitive Types in ANSI C (C89)/ISO C (C90) - `char`, `short`, `int`, `float` and `double`.
2. Primitive Types added to ISO C (C99) - `long long`
3. User Defined Types – `struct`, `union`, `enum` and `typedef` (will be discussed in separate session).
4. Derived Types – pointer, array and function pointer (will be discussed in separate session).

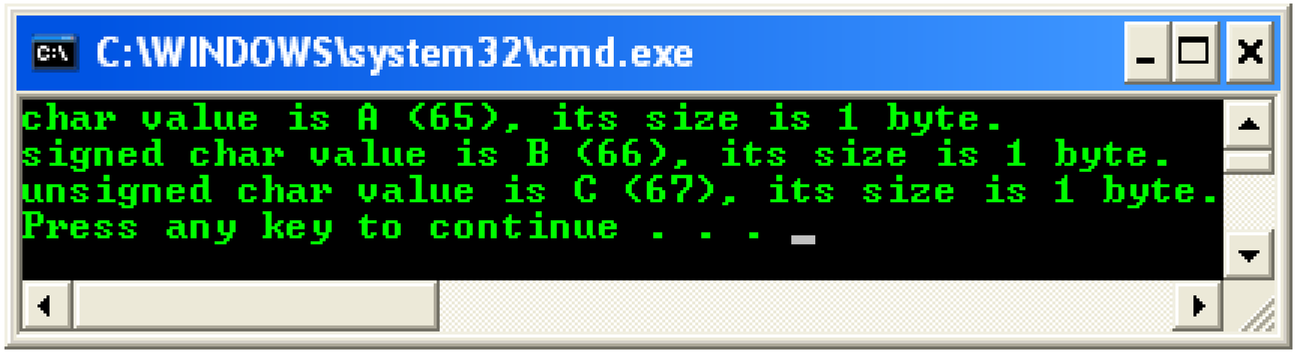
C BASIC (DATA) TYPES

Type	Size in Bits	Comments	Other Names
Primitive Types in ANSI C (C89)/ISO C (C90)			
char	≥ 8	▪ **sizeof()** will give the size in units of **char**s. ▪ need not be 8-bit ▪ The number of bits is given by the CHAR_BIT macro in the limits.h header. ▪ Integer operations can be performed portably only for the range: 0 ~ 127 (2^8 / 2).	—
signed char	Same as **char** but guaranteed to be signed	▪ Can store integers in the range: -127 ~ 127 (2^8) portably.	—
unsigned char	Same as **char** but guaranteed to be unsigned.	▪ Can store integers in the range: 0 ~ 255 (2^8) portably.	—

char type program example

C BASIC (DATA) TYPES

- A sample output.



```
char value is A (65), its size is 1 byte.
signed char value is B (66), its size is 1 byte.
unsigned char value is C (67), its size is 1 byte.
Press any key to continue . . . _
```

C BASIC (DATA) TYPES

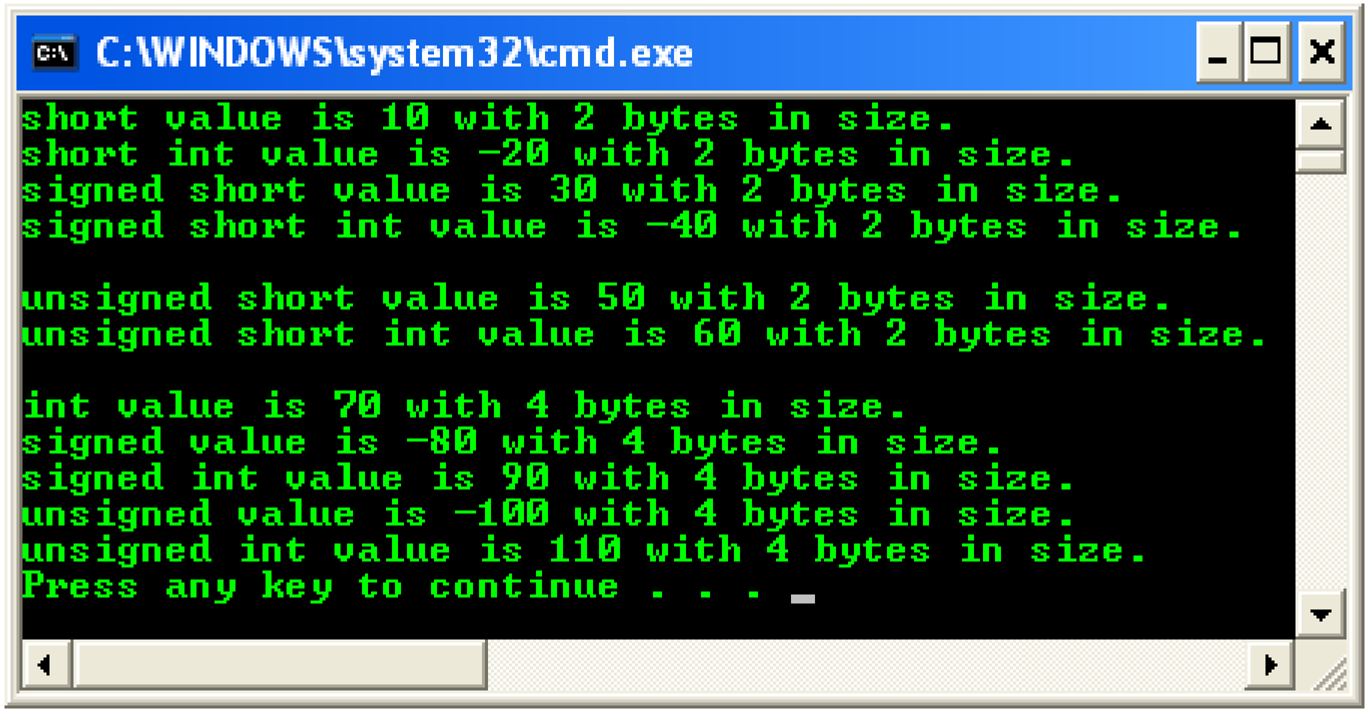
short	≥ 16, ≥ size of **char**	▪ Can store integers in the range: -32767 ~ 32767 (2^{16} / 2) portably. ▪ Reduce memory usage though the resulting executable may be larger and probably slower as compared to using **int**.	**short int**, **signed short**, **signed short int**
unsigned short	Same as **short** but unsigned	▪ Can store integers in the range: 0 ~ 65535 (2^{16}) portably. ▪ Used to reduce memory usage though the resulting executable may be larger and probably slower as compared to using **int**.	**unsigned short int**
int	≥ 16, ≥ size of **short**	▪ Basic signed integer type. ▪ Represent a typical processor's data size which is word-size ▪ An integral data-type. ▪ Can store integers in the range: -32767 ~ 32767 (2^{16} / 2) portably.	**signed**, **signed int**
unsigned int	Same as **int** but unsigned.	▪ Can store integers in the range: 0 ~ 65535 (2^{16}) portably.	**unsigned**

short int type program example

, ©

C BASIC (DATA) TYPES

- A sample output.



```
short value is 10 with 2 bytes in size.
short int value is -20 with 2 bytes in size.
signed short value is 30 with 2 bytes in size.
signed short int value is -40 with 2 bytes in size.

unsigned short value is 50 with 2 bytes in size.
unsigned short int value is 60 with 2 bytes in size.

int value is 70 with 4 bytes in size.
signed value is -80 with 4 bytes in size.
signed int value is 90 with 4 bytes in size.
unsigned value is -100 with 4 bytes in size.
unsigned int value is 110 with 4 bytes in size.
Press any key to continue . . . _
```

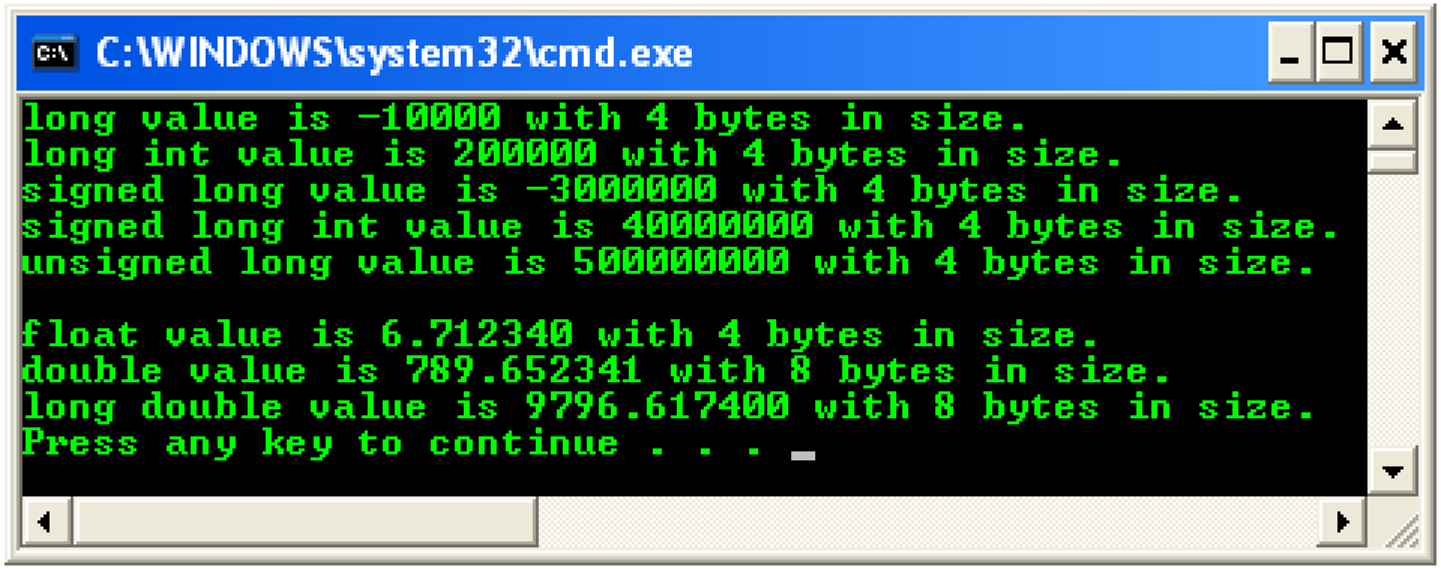
C BASIC (DATA) TYPES

long	≥ 32, ≥ size of **int**	▪ long signed integer type. ▪ Can store integers in the range: -2147483647 ~ 2147483647 (2^{32} / 2) portably.	**long int**, **signed long**, **signed long int**
unsigned long	Same as **long** but unsigned	▪ Can store integers in the range: 0 ~ 4294967295 (2^{32}) portably.	**unsigned long int**
float	≥ size of **char**	▪ Used to reduce memory usage when the values used do not vary widely. ▪ The format used is implementation defined and unnecessarily obeys the IEEE 754 single-precision format. ▪ **unsigned** cannot be specified.	—
double	≥ size of **float**	▪ Typical floating-point data type used by processor. ▪ The format used is implementation defined and unnecessarily obeys the IEEE 754 double-precision format. ▪ **unsigned** cannot be specified.	—
long double	≥ size of **double**	▪ **unsigned** cannot be specified.	—

long int type program example

C BASIC (DATA) TYPES

- A sample output.



```
long value is -10000 with 4 bytes in size.
long int value is 200000 with 4 bytes in size.
signed long value is -3000000 with 4 bytes in size.
signed long int value is 40000000 with 4 bytes in size.
unsigned long value is 500000000 with 4 bytes in size.

float value is 6.712340 with 4 bytes in size.
double value is 789.652341 with 8 bytes in size.
long double value is 9796.617400 with 8 bytes in size.
Press any key to continue . . . _
```

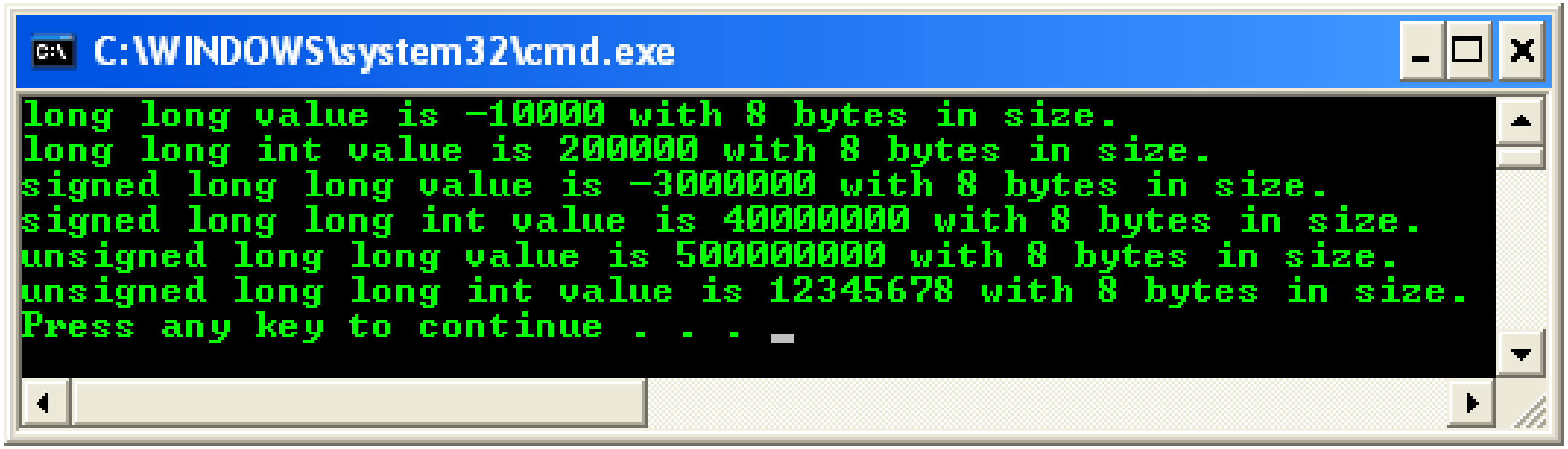
C BASIC (DATA) TYPES

Type	Size in Bits	Comments	Other Names
Primitive Types added to ISO C (C99)			
`long long`	≥ 64, ≥ size of **`long`**	▪ Can store integers in the range: -9223372036854775807 ~ 9223372036854775807 (2^{64} / 2) portably.	**long long int, signed long long, signed long long int**
`unsigned long long`	Same as **`long long`**, but `unsigned`.	▪ Can store integers in the range: 0 ~ 18446744073709551615 (2^{64}) portably.	**unsigned long long int**

long long int type program example

C BASIC (DATA) TYPES

- A sample output.



```
long long value is -10000 with 8 bytes in size.
long long int value is 200000 with 8 bytes in size.
signed long long value is -3000000 with 8 bytes in size.
signed long long int value is 40000000 with 8 bytes in size.
unsigned long long value is 500000000 with 8 bytes in size.
unsigned long long int value is 12345678 with 8 bytes in size.
Press any key to continue . . . _
```

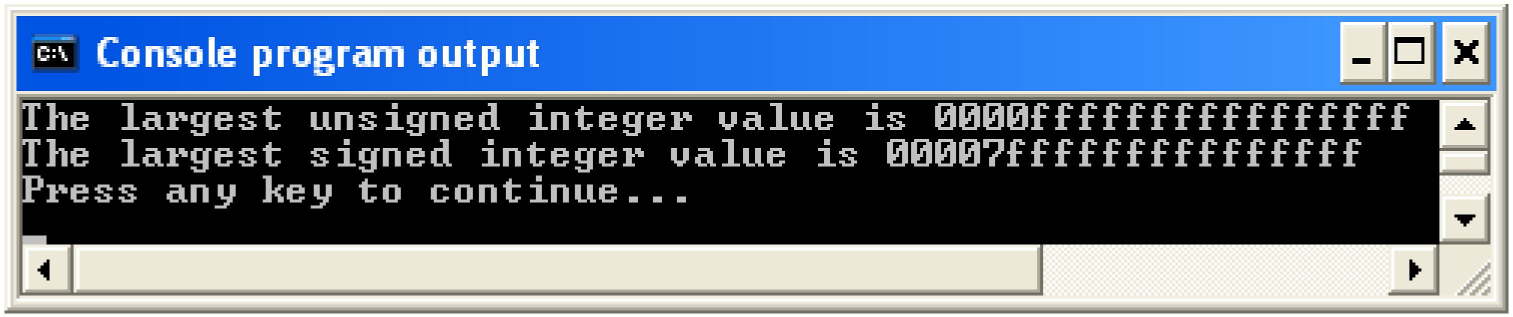
C BASIC (DATA) TYPES

`intmax_t`	Signed integer types capable of representing any value of any signed integer type.	▪ It is a `typedef` represents the signed integer type with largest possible range. ▪ If you want an integer with the widest range possible on the platform on which it is being used.	—
`uintmax_t`	Unsigned integer types capable of representing any value of any unsigned integer type	▪ It is a `typedef` represents the unsigned integer type with largest possible range. ▪ If you want an integer with the widest range possible on the platform on which it is being used.	—

Program example

C BASIC (DATA) TYPES

- A sample output.



```
The largest unsigned integer value is 0000ffffffffffffffff
The largest signed integer value is 00007fffffffffffffff
Press any key to continue...
```

C BASIC (DATA) TYPES

Boolean type

- The boolean (true/false) type is `_Bool` defined in stdbool.h
- The stdbool.h type also defines a few useful identifiers as macros: `bool` is defined as `_Bool`, true as `1`, false as `0`.
- Additionally, `__bool_true_false_are_defined` is defined as 1.
- The `_Bool` type and stdbool.h header did not exist in pre-1999 versions of the standard.

bool in VC++ example

bool in Pelles C example

C BASIC (DATA) TYPES

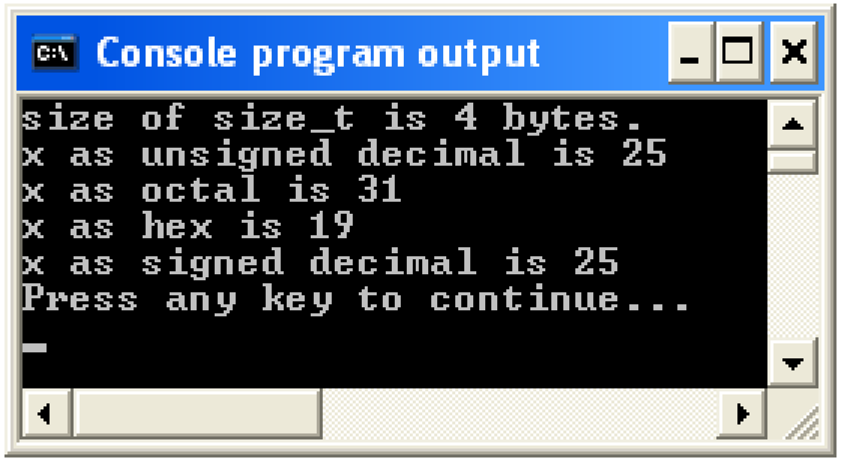
Size and pointer difference types

- Separate `size_t` and `ptrdiff_t` types to represent memory-related quantities.
- Existing types were inadequate, because their size is defined according to the target processor's arithmetic capabilities, not the memory capabilities, such as the address space availability.
- Both of these types are defined in the stddef.h header file (cstddef in C++).
- `size_t` is used to represent the maximum size of any object (including arrays) in the particular implementation.
- An unsigned integer type used to represent the sizes of objects

size_t program example

C BASIC (DATA) TYPES

- A sample output.



```
size of size_t is 4 bytes.
x as unsigned decimal is 25
x as octal is 31
x as hex is 19
x as signed decimal is 25
Press any key to continue...
_
```

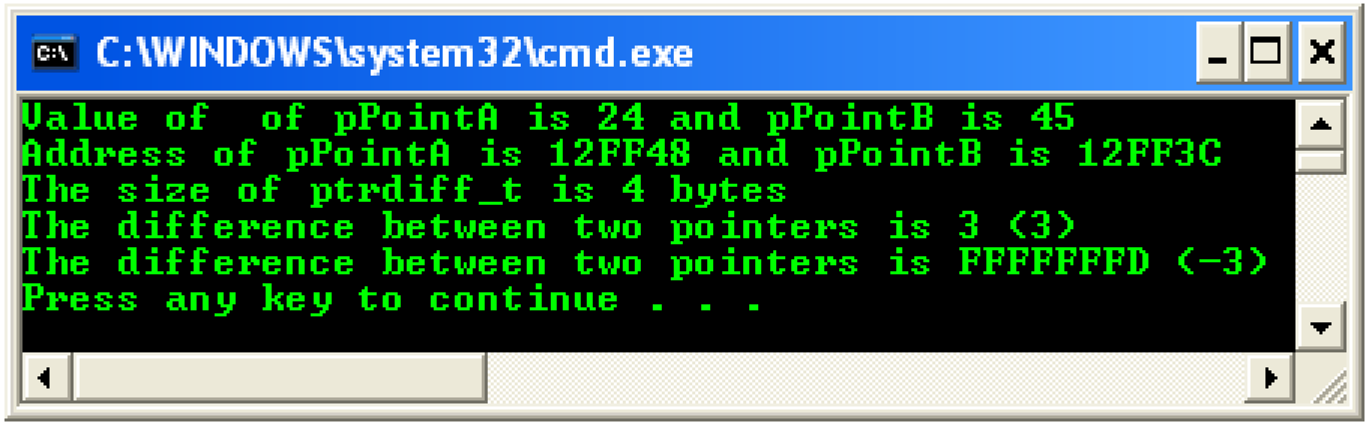
C BASIC (DATA) TYPES

- Used as the return type of the `sizeof()` operator.
- The maximum size of `size_t` is provided via `SIZE_MAX`, a macro constant which is defined in the stdint.h header file (`cstdint` in C++).
- It is guaranteed to be at least `65535`.
- `ptrdiff_t` is used to represent the difference between pointers.
- Is the signed integer type of the result of subtracting two pointers.
- The type's size is chosen so that it could store the maximum size of a theoretically possible array of any type.
- On a 32-bit system `ptrdiff_t` will take 32 bits and on a 64-bit one - 64 bits and it is portable.

size_t and ptrdiff_t: a story ptrdiff_t program example

C BASIC (DATA) TYPES

- A sample output.



```
Value of  of pPointA is 24 and pPointB is 45
Address of pPointA is 12FF48 and pPointB is 12FF3C
The size of ptrdiff_t is 4 bytes
The difference between two pointers is 3 (3)
The difference between two pointers is FFFFFFFD (-3)
Press any key to continue . . .
```

C BASIC (DATA) TYPES

Interface to the properties of the basic types

- Information about the actual properties, such as size, of the basic arithmetic types, is provided via macro constants in two header files,

  1) limits.h header (`climits` in C++) defines macros for integer types.
  2) float.h header (`cfloat` in C++) defines macros for floating-point types.

- The actual values depend on the implementation.

C BASIC (DATA) TYPES

Fixed width integer types

- C99 standard includes definitions of several new integer types to enhance programs' portability.
- Existing basic integer types were considered inadequate; because their actual sizes are implementation defined and may vary across different systems.
- The new types are especially useful in embedded environments where hardware supports limited to several types and varies from system to system.
- All new types are defined in inttypes.h (`cinttypes` in C++) and stdint.h (`cstdint` in C++) header files.
- The types can be grouped into the following categories:

C BASIC (DATA) TYPES

- **Exact width** integer types - are guaranteed to have the same number **N** of bits across all implementations. Included only if it is available in the implementation.
- **Least width** integer types - are guaranteed to be the smallest type available in the implementation, that has at least specified number **N** of bits. Guaranteed to be specified for at least N=8, 16, 32, 64.
- **Fastest** integer types - are guaranteed to be the fastest integer type available in the implementation, that has at least specified number **N** of bits. Guaranteed to be specified for at least N=8, 16, 32, 64.
- **Pointer** integer types - are guaranteed to be able to hold a pointer.
- **Maximum width** integer types - are guaranteed to be the largest integer type in the implementation.

C BASIC (DATA) TYPES

- The following table summarizes the types and the interface to acquire the implementation details (**N** refers to the number of bits).

Type category	Signed types			Unsigned types		
	Type	Min value	Max value	Type	Min value	Max value
Exact width	`int`**`N`**`_t`	`INT`**`N`**`_MIN`	`INT`**`N`**`_MAX`	`uint`**`N`**`_t`	0	`UINT`**`N`**`_MAX`
Least width	`int_least`**`N`**`_t`	`INT_LEAST`**`N`**`_MIN`	`INT_LEAST`**`N`**`_MAX`	`uint_least`**`N`**`_t`	0	`UINT_LEAST`**`N`**`_MAX`
Fastest	`int_fast`**`N`**`_t`	`INT_FAST`**`N`**`_MIN`	`INT_FAST`**`N`**`_MAX`	`uint_fast`**`N`**`_t`	0	`UINT_FAST`**`N`**`_MAX`
Pointer	`intptr_t`	`INTPTR_MIN`	`INTPTR_MAX`	`uintptr_t`	0	`UINTPTR_MAX`
Maximum width	`intmax_t`	`INTMAX_MIN`	`INTMAX_MAX`	`uintmax_t`	0	`UINTMAX_MAX`

USER DEFINED (DATA) TYPES

Keyword	Size	Note
struct	≥ sum of size of each member	An aggregate type which can contain more than one different types.

tag or label is optional

```
struct theEmployee {
            int age;
            double salary;
            char department;
            char name[15];
            char address[5][25];
         };
struct theEmployee workerRec;
```

```
typedef struct
{
        int x;
        int SomeArray[100];
} MyFoo;

int main()
{
        MyFoo strctVar;

        return 0;
}
```

```
struct newPoint {
    short xPoint;
    short yPoint;
} justPoint;

justPoint thePoint;
```

USER DEFINED (DATA) TYPES

`union`	≥ size of the largest member	An aggregate type which can contain more than one other types. `union` uses <u>shared memory space</u> compared to `struct`, so only one member can be accessed at one time.

```
union someData
{
    int  pNum;
    float qNum;
    double rNum;
};
union someData simpleData;
```

```
union OtherData{
   char aNum;
   int xNum;
   float fNum;
} simpleData;
simpleData saveData;
```

USER DEFINED (DATA) TYPES

`enum`	≥ size of `char`	Enumerations are a separate type from **`int`**s, though they are mutually convertible. Used to declare identifiers as constants in an ordered manner.

```
enum ndays {Mon, Tue, Wed, Thu, Fri, Sat, Sun};

/ * Creates enum days type, which the identifiers are set
    automatically to the integers 0 to 6. */
enum ndays ndayCount;
```

```
enum trafficDirection{
   north,
   south,
   east,
   west
  };

enum trafficDirection newDirection;
```

```
enum cColor = {red = 2,
green, blue, black};

Enum cColor ccolorCode;
```

USER DEFINED (DATA) TYPES

typedef	same as the type; being given a new name	**typedef** used to give new identifier names or alias (to simplify the long identifier names), normally used for aggregate defined types.

```
typedef unsigned char BYTE; /* Declares BYTE to be a synonym for unsigned char */
typedef float FLOAT; /* Declares FLOAT (uppercase letter) to be a synonym for unsigned float
(lowercase) */
```

tag or label is optional

```
typedef struct simpleData
 {     int nData;
    char cData;
} newNameType;
```

Or

```
typedef struct {     int nData;
char cData;} newNameType;
newNameType strctType;
```

```
typedef struct TOKEN_SOURCE {
  CHAR     SourceName[8];
  LUID     SourceIdentifier;
} TOKEN_SOURCE, *PTOKEN_SOURCE;

TOKEN_SOURCE newToken;
```

```
typedef union unData{
     double lngSalary;
     int nDay;
}newUntype;

newUnType lntotalSalary;
```

```
typedef enum DayNames { Monday,
                   Tuesday,
                   Wednesday,
                   Thursday,
                   Friday, Saturday, Sunday
         } Weekdays;
Weekdays dayOfWeek;
```

DERIVED (DATA) TYPES

Type	Size	Note
`type*` (a pointer)	≥ size of **`char`**	▪ Hold the memory address which point to the actual data/value. ▪ `0` address always represents the null pointer (an address where no data can be placed), irrespective of what bit sequence represents the value of a null pointer. ▪ Pointers to different types will have different sizes. So they are not convertible to one another. ▪ Even in an implementation which guarantees all data pointers to be of the same size, function pointers and data pointers are in general incompatible with each other. ▪ For functions taking a variable number of arguments, the arguments passed must be of appropriate type.

`char *ptoChar;`
`char csimpleChr = 'T';`
`char *chptr;`
`// assignment`
`chptr = &csimpleChr;`
		`int iNumber = 20;`
		`int *imyPtr = &iNumber;`

Note

sizeof (short) ≤ sizeof (int) ≤ sizeof (long) ≤ sizeof (long long)

DERIVED (DATA) TYPES

`type [integer]` (an array)	≥ integer × size of type	▪ Use to declare a variable with collection of identical properties or types. ▪ Simplify variable declaration. ▪ In a declaration which also initializes the array (including a function parameter declaration), the size of the array (the integer) can be omitted, which is called unsized. ▪ type [ ] is not the same as type*. Only under some circumstances one can be converted to the other.

`int fstudentNumber[3] = {4,7,1};`
`int nrowandColumn[1][2] = {34, 21};`
`int nlongHeightWidth[3][4][5] = 0;`
		`char cName1[ ] =`
		`{'a','r','r','a','y'};`
		`char cName2[ ] = {"array"};`
		`char cName3[6] = "array";`
		`int nrowCol[2][3] = {4,2,3,7,2,8};`

C (Basic) :Variables in C

Variables in C:

- C programmers generally agree on the following **conventions** for naming variables.
  - Begin variable names with lowercase letters
  - Use meaningful identifiers
  - Separate “words” within identifiers with underscores or mixed upper and lower case.
  - Examples: surfaceArea or surface_Area
  - Be consistent!

- C is **case sensitive**
  - It matters whether an **identifier**, such as a variable name, is uppercase or lowercase.
  - Example:

    ```
    area
    Area
    AREA
    ArEa
    ```

    are all seen as <u>different</u> variables by the compiler.

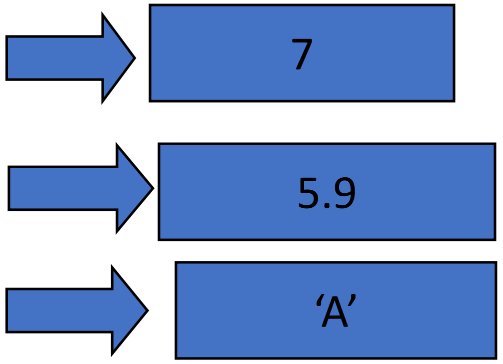
C has three basic predefined data types:

- Integers (whole numbers)
  - **int**, long int, short int, unsigned int
- Floating point (real numbers)
  - **float**, **double**
- Characters
  - **char**
- At this point, you need only be concerned with the data types that are bolded.
  - Variables may be be given initial values, or **initialized**, when declared. Examples:

int length = 7 ;

float diameter = 5.9 ;

char initial = ‘A’ ;



Thank you

Sem 3 ka safar, C language ka pyaar,
Preetam Sir ke saath, coding har baar!
‘#include’ se shuru, ‘return 0’ tak ka raasta,
Logic ka game, ab sabko hai aata!
Loops ho ya pointer, sab kuch clear hai,
Sir ki class mein, har doubt disappear hai!