A Quick Introduction to C Programming

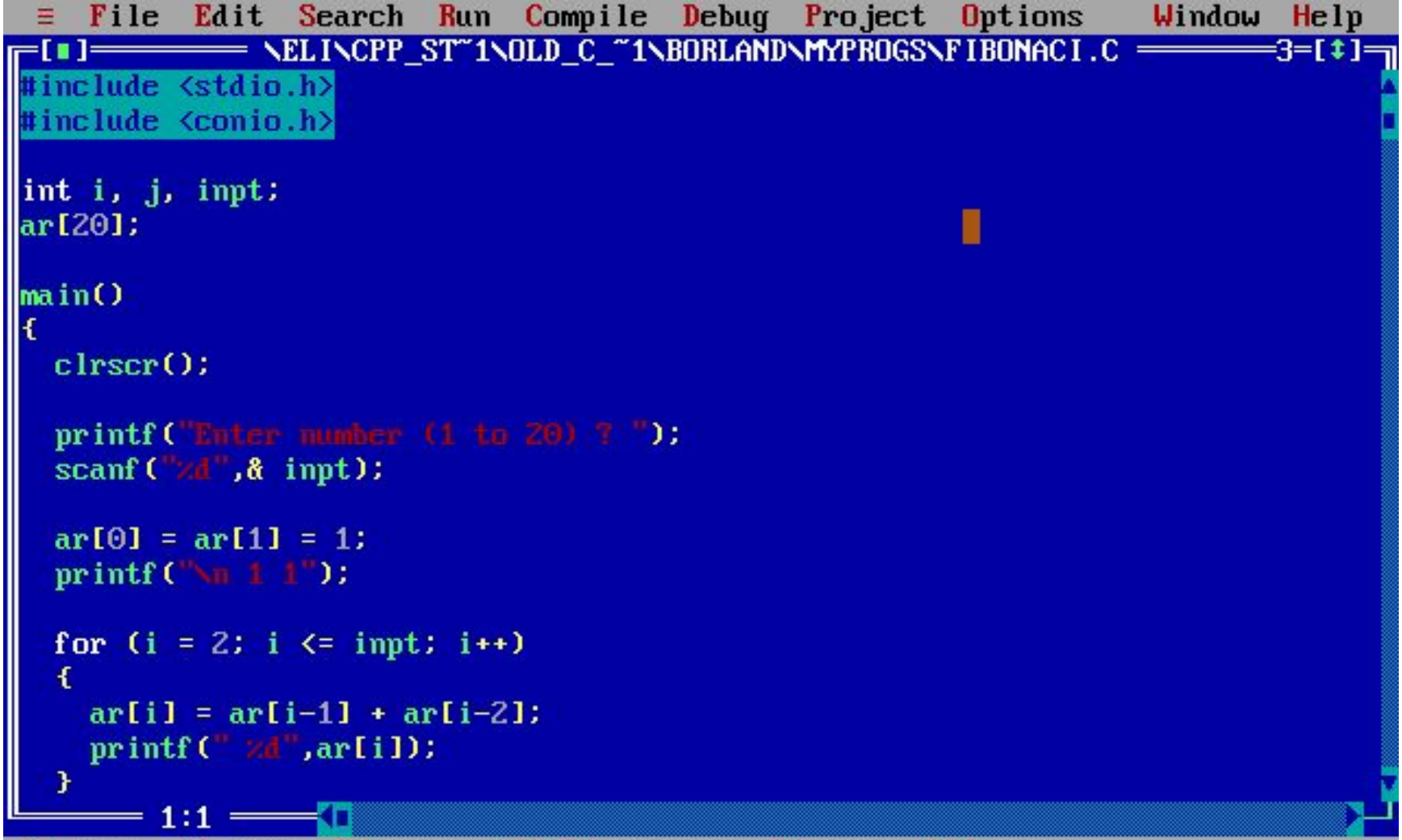
Kailash Prasad
PhD Electrical Engineering

Today's Agenda

- Introduction to C
- Keywords and Identifiers
- Data Types
- Storage Class
- Operators
- Decision Making
- Loops
- Functions
- Arrays
- Strings
- Pointers
- Preprocessors
- I/O
- Recursion
- Assignments

Resources

- https://www.tutorialspoint.com/cprogramming/index.htm
- thenewboston https://www.youtube.com/watch?v=2NWeucMKrLI&list=PL6gx4Cwl9DGAKIXv8Yr6nhGJ9Vlcjyymq


File Edit Search Run Compile Debug Project Options Window Help
\ELI\CPP_ST~1\OLD_C_~1\BORLAND\MYPROGS\FIBONACI.C
3
#include <stdio.h>
#include <conio.h>
int i, j, inpt;
ar[20];
main()
{
clrscr();
printf("Enter number (1 to 20) ? ");
scanf("%d",& inpt);
ar[0] = ar[1] = 1;
printf("\n 1 1");
for (i = 2; i <= inpt; i++)
{
ar[i] = ar[i-1] + ar[i-2];
printf(" %d",ar[i]);
}
1:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

What is C?

- C is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system.

Why C programming in Embedded Systems

- C is one of the oldest programming languages till date
- C is neither low-level nor high-level language. You can closely work with the registers of the CPU. You can embed assembly code in your C program.
- Almost all Operating Systems are written in C. Microprocessor(x86,AMD), Microcontroller(8051, MIPS, ARM), DSPs, DIPs and even FPGA’s can be programmed with C
- C is very fast. Most RTOS (Real Time Operating Systems) that are prevalent are coded with C.

C vs Embedded C

C programming	Embedded C programming
Possesses native development in nature.	Possesses cross development in nature.
Independent of hardware architecture.	Dependent on hardware architecture (microcontroller or other devices).
Used for Desktop applications, OS and PC memories.	Used for limited resources like RAM, ROM and I/O peripherals on embedded controller.

Identifier and Keywords

- Identifiers
  A C identifier is a name used to identify a variable, function, or any other user-defined item.
  An identifier starts with a letter A to Z, a to z, or an underscore '_'.
  mohd zara abc move_name a_123
  myname50 _temp j a23b9 retVal

Identifier and Keywords

- Keywords
  These are reserved words that may not be used as constants or variables or any other identifier names.

auto	else	long	switch
break	enum	register	typedef

DATA TYPE	MEMORY (BYTES)	RANGE	FORMAT SPECIFIER
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned char	1	0 to 255	%c
float	4	1.2E-38 to 3.4E+38	%f
double	8	2.3E-308 to 1.7E+308	%lf
long double	12	3.4E-4932 to 1.1E+4932	%Lf

Storage Classes

- Auto
  The auto storage class is the default storage class for all local variables.
- Register
  The register storage class is used to define local variables that should be stored in a register instead of RAM.
- Static
  Local Variables that retains the value throughout tha program
- Extern
  The extern storage class is used to give a reference of a global variable that is visible to ALL the program files.

Operators

- Arithmetic Operators
  `+  -  *  /  %`
- Relational Operators
  `==  !=  <  >  <=  >=`
- Logical Operators
  `&&  ||  !`
- Bitwise Operators
  `&  |  ^  ~  <<  >>`
- Assignment Operators
  `=  +=  -=  *=  /=  %=  <<=  >>=  |=  &=  ^=`

Operators

- Misc
  * & ?: sizeof()

Decision Making

- If

```
if(boolean_expression) {
   /* statement(s) will execute if the boolean expression is true */
}
```

- If -else

```
if(boolean_expression) {
   /* statement(s) will execute if the boolean expression is true */
} else {
   /* statement(s) will execute if the boolean expression is false */
}
```

Decision Making

- If ...else if... else

```
if(boolean_expression 1) {
   /* Executes when the boolean expression 1 is true */
} else if( boolean_expression 2) {
  /* Executes when the boolean expression 2 is true */
} else if( boolean_expression 3) {
  /* Executes when the boolean expression 3 is true */
} else {
   /* executes when the none of the above condition is true */
}
```

Decision Making

- Switch

```
switch(expression) {
  case constant-expression  :
        statement(s);
        break; /* optional */
case constant-expression  :
        statement(s);
        break; /* optional */
default : /* Optional */
        statement(s)
}
```

Loops

- While loop

```
#include <stdio.h>
int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {
         printf("value of a: %d\n", a);
         a++;
   }

   return 0;
}
```

- For loop

```
#include <stdio.h>
int main () {

   int a;

   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 ){
         printf("value of a: %d\n",
a);
   }
   return 0;
}
```

- Do while loop

```
#include <stdio.h>
int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {
         printf("value of a: %d\n",
a);
         a = a + 1;
   }while( a < 20 );
   return 0;
}
```

Loop control statements

- Break

```
  /* local variable definition */
  int a = 10;

  /* while loop execution */
  while( a < 20 ) {

     printf("value of a: %d\n", a);
          a++;

          if( a > 15) {
          /* terminate the loop
using break statement */
          break;
          }
  }
```

- Continue

```
  /* do loop execution */
  do {

     if( a == 15) {
          /* skip the iteration */
          a = a + 1;
          continue;
          }

          printf("value of a: %d\n",
a);
          a++;

  } while( a < 20 );
```

- Goto

```
  /* local variable definition */
  int a = 10;

  /* do loop execution */
  LOOP:do {

     if( a == 15) {
          /* skip the iteration */
          a = a + 1;
          goto LOOP;
          }

          printf("value of a: %d\n",
a);
          a++;

  }while( a < 20 );
```

Functions

- Function Declaration and Function call

```
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;
   /* calling a function to get max value */
   ret = max(a, b);
   printf( "Max value is : %d\n", ret );
   return 0;
}
```

- Function Definition

```
/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;
   if (num1 > num2)
         result = num1;
   else
         result = num2;
   return result;
}
```

Arrays

- Declaring Arrays

```
type arrayName [ arraySize ];

double balance[10];
```

- Initializing Arrays

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

balance[4] = 50.0;
```

- Example in C

```
#include <stdio.h>
 int main () {

   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;
   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ ) {
          n[ i ] = i + 100; /* set element at location i to i + 100
*/
   }

   /* output each array element's value */
   for (j = 0; j < 10; j++ ) {
          printf("Element[%d] = %d\n", j, n[j] );
   }
   return 0;
}
```

Strings

- Initializing String

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

char greeting[] = "Hello";
```

- Example in C

```
#include <stdio.h>

int main () {

  char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
  printf("Greeting message: %s\n", greeting );
  return 0;
}
```

Pointers

- Find the Address of the variable

```
#include <stdio.h>

int main () {

   int  var1;
   char var2[10];

   printf("Address of var1 variable: %x\n", &var1  );
   printf("Address of var2 variable: %x\n", &var2  );

   return 0;
}
```

- A pointer is a variable whose value is the address of another variable

```
type *var-name;

int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

Pointers

- How to Use pointer

```
#include <stdio.h>

int main () {

   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in pointer variable*/

   printf("Address of var variable: %x\n", &var  );

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

- Pointer to Array

```
#include <stdio.h>

const int MAX = 3;

int main () {

   int  var[] = {10, 100, 200};
   int i, *ptr[MAX];

   for ( i = 0; i < MAX; i++) {
      ptr[i] = &var[i]; /* assign the address of integer. */
   }

   for ( i = 0; i < MAX; i++) {
      printf("Value of var[%d] = %d\n", i, *ptr[i] );
   }

   return 0;
}
```

Pointers

- Pointer to String

```
#include <stdio.h>
const int MAX = 4;
int main () {

  char *names[] = {
     "Zara Ali",
     "Hina Ali",
     "Nuha Ali",
     "Sara Ali"
  };

  int i = 0;

  for ( i = 0; i < MAX; i++) {
     printf("Value of names[%d] = %s\n", i, names[i] );
  }

  return 0;
}
```

Preprocessors

- **#define**
  #define MAX 20
- **#include**
  #include <math.h>
  #include “myheader.h”
- **#undef**
  #undef FILE_SIZE
  #define FILE_SIZE 42
- **#ifndef … #endif**
  #ifndef MESSAGE
   #define MESSAGE "You wish!"
  #endif
- **#ifdef**
  #ifdef DEBUG
    /* Your debugging statements here */
  #endif

Parameterized Macros

```
int square(int x) {
  return x * x;
}

#define square(x) ((x) * (x))
```

```
#include <stdio.h>

#define MAX(x,y) ((x) > (y) ? (x) : (y))

int main(void) {
  printf("Max between 20 and 10 is %d\n", MAX(10, 20));
  return 0;
}
```

I/O

scanf and printf

```
#include <stdio.h>
int main( ) {

   char str[100];
   int i;

   printf( "Enter a value :");
   scanf("%s %d", str, &i);

   printf( "\nYou entered: %s %d ", str, i);

   return 0;
}
```

gets and puts

```
#include <stdio.h>
int main( ) {

   char str[100];

   printf( "Enter a value :");
   gets( str );

   printf( "\nYou entered: ");
   puts( str );

   return 0;
}
```

Recursions

- Syntax

```
void recursion() {
  recursion(); /* function calls itself */
}

int main() {
  recursion();
}
```

- Example in C

```
#include <stdio.h>

unsigned long long int factorial(unsigned int i) {

  if(i <= 1) {
    return 1;
  }
  return i * factorial(i - 1);
}

int  main() {
  int i = 12;
  printf("Factorial of %d is %d\n", i, factorial(i));
  return 0;
}
```

Any Questions?

Assignment

- Write a program in C to flip the nth bit of integer using bitwise operators.
- Write a program in C to convert decimal number to BCD using the function.
  Input 23 Expected Output 0010 0011
- Write a program in C to check whether a number is armstrong or not.
- Write a program in C to sort n numbers.
- Write a program to check whether a character is present in a string or not.