Lecture 8: C language

- History of C
- Structure of a C program
- C data types
- Variable declaration and scope
- C operators
- Loops and iterations
- Pointers
- Structures in C
- C and assembly language
History of C

- Developed in 1972 by Dennis Ritchie on a DEC PDP-11 at Bell Systems Lab as a *system development language*
  - Derived from the language B of Ken Thompson, which itself was based on BCPL, developed by Martin Richards
- For many years the *de-facto* C standard was the version provided with Unix System V
  - The C Programming Language, Brian Kernigham and Dennis Ritchie, Prentice-Hall 1978
- In 1983 ANSI creates a group to begin the standardization of C
  - ANSI C is finalized in 1989, and ISO adopts it in 1990
Structure of a C program

```
#include <stdio.h>         /* this tells the compiler where to find I/O routines and declares all the necessary functions. However, we do not use I/O in this example! */
#define YEAR 365              /* equate the name YEAR to the value 365 */
int years_to_days(int x);

void main(void)               /* this is the main function where execution starts */
{
    int days;                 /* declare an integer variable called z */
    days = years_to_days(3);  /* calculate the number of days in 3 years */
}

int years_to_days(int x)      /* define function years_to_days with int parameter x */
{
    return(x * YEAR);         /* multiply x by 365 and return the result */
}
```

- Every C program contains at least one function: `main`
- In this case `void main(void)` requires no parameters and returns no parameters
- Every statement ends with a semicolon, except for the closing bracket of a block `{ }`
- Local variable, only available in the scope of `main()`
C data types

- Four basic data types
  - char: character
  - int: integer
  - float: real or floating point
  - double: double precision float

- Four modifiers
  - signed
  - unsigned
  - long
  - short

- Four storage classes
  - auto: variable is not required outside its block (the default)
  - register: the variable will be allocated on a CPU register
  - static: allows a local variable to retain its previous value upon reentry
  - extern: global variable declared in another file

- Additionally, C supports
  - the null data type: void
  - Any user-defined types

<table>
<thead>
<tr>
<th>Type</th>
<th>Width (bits)</th>
<th>Minimum range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>8</td>
<td>-127 to 127</td>
</tr>
<tr>
<td>unsigned char</td>
<td>8</td>
<td>0 to 255</td>
</tr>
<tr>
<td>signed char</td>
<td>8</td>
<td>-127 to 127</td>
</tr>
<tr>
<td>int</td>
<td>16</td>
<td>-32,767 to 32,767</td>
</tr>
<tr>
<td>unsigned int</td>
<td>16</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>signed int</td>
<td>16</td>
<td>Same as int</td>
</tr>
<tr>
<td>short int</td>
<td>16</td>
<td>Same as int</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>8</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>signed short int</td>
<td>8</td>
<td>Same as short int</td>
</tr>
<tr>
<td>long int</td>
<td>32</td>
<td>-2,147,483,647 to 2,147,483,647</td>
</tr>
<tr>
<td>signed long int</td>
<td>32</td>
<td>-2,147,483,647 to 2,147,483,647</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>32</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>float</td>
<td>32</td>
<td>Six-digit precision</td>
</tr>
<tr>
<td>double</td>
<td>64</td>
<td>Ten-digit precision</td>
</tr>
<tr>
<td>long double</td>
<td>128</td>
<td>Ten-digit precision</td>
</tr>
</tbody>
</table>
Variable declaration and scope

- Variables **MUST** be declared before they are used
  - Any declaration MUST precede the first statement in a block
- Variables declared inside a block are local to that block
  - They cannot be accessed from outside the block
- Variables can be initialized when they are declared or afterwards

```c
int i; /* Integer i is global to the entire program and is visible to everything from this point */
void function_1(void) /* A function with no parameters */
{
    int k; /* Integer k is local to function_1 */
    {
        int q; /* Integer q exists only in this block */
        int j; /* Integer j is local and not the same as j in main */
    }
}
void main(void) /* Integer j is local to this block within function main */
{
    int j; /* This is the point at which integer j ceases to exist */
}
```
# C operators

<table>
<thead>
<tr>
<th>Type</th>
<th>Operator</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td></td>
<td>%</td>
<td>Modulus</td>
</tr>
<tr>
<td></td>
<td>--</td>
<td>Decrement (by 1)</td>
</tr>
<tr>
<td></td>
<td>++</td>
<td>Increment (by 1)</td>
</tr>
<tr>
<td></td>
<td>+=</td>
<td>Increment ($a+=b$ means $a=a+b$)</td>
</tr>
<tr>
<td></td>
<td>-=</td>
<td>Decrement ($a-=b$ means $a=a-b$)</td>
</tr>
<tr>
<td>Relational</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td></td>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td></td>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td></td>
<td>!=</td>
<td>Different from</td>
</tr>
<tr>
<td>Logic</td>
<td>&amp;&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>!</td>
<td>NOT</td>
</tr>
<tr>
<td>Bit-wise</td>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>^</td>
<td>XOR</td>
</tr>
<tr>
<td></td>
<td>~</td>
<td>NOT</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;</td>
<td>Right shift</td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;</td>
<td>Left shift</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>?</td>
<td>Ternary ($y=x&gt;9?100:200$)</td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
<td>Pointer operators</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>Width of a datatype (in bytes)</td>
</tr>
<tr>
<td></td>
<td>.</td>
<td>Access to structures</td>
</tr>
<tr>
<td></td>
<td>-&gt;</td>
<td>Access to arrays</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most</td>
<td>( ) [ ] -&gt; .</td>
</tr>
<tr>
<td></td>
<td>! ~ ++ -- - (cast) * &amp;</td>
</tr>
<tr>
<td></td>
<td>sizeof / %</td>
</tr>
<tr>
<td></td>
<td>&lt;&lt; &gt;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt; &lt;= &gt; &gt;=</td>
</tr>
<tr>
<td></td>
<td>== !=</td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>? = += -= *= /= \</td>
</tr>
<tr>
<td>Least</td>
<td></td>
</tr>
</tbody>
</table>
Loops and iterations

- In C any expression different than ZERO is `TRUE`, including negative numbers, strings, ...
- C provides the following constructs

```c
if (expr2) {
    block2;
} else if (expr3) {
    block3;
} else {
    default_block;
}

for (initialization; condition; increment) {
    block;
}

for (;;) {
    block;
    if (expr)
        break;
}

while (expression) {
    block;
}

do {
    block;
} while (expression);

goto label;
block1;
label:
block2;

switch (expression) {
    case constant1:
        block1;
        break;
    case constant2:
        block2;
        break;
    default:
        block_default;
}
```
Pointers (I)

- A pointer is a variable that stores a memory address
- A pointer must be declared and initialized before it can be used

- Pointers and arrays are closely related
  - the name of the array serves as a pointer to its first element
  - the first element has index 0
  - array elements can be addressed using brackets or pointer arithmetic

- Strings of characters and arrays are closely related
  - A string is an array of characters followed by the ‘\0’ null character

```c
void main() {  
  int a=10;  
  int *p;  
  p = &a;  
  *p = 20;  /* 'a' contains the value 20*/  
}
```

```c
void main() {  
  int array[5]={1,2,3,4,5};  
  int value, *p;  
  p = &array[0];  /* these two expressions */  
  p = array;  /* are equivalent */  
  array[2];  /* both expressions point to */  
  *(p+2);  /* the 3rd element in array */  
}
```

```c
void main() {  
  char *string1="hello"  
  char string2[6]={'h','e','l','l','o','\0'};  
  char *p_string;  
  p_string = string1;  
  printf("%s\n",string1);  /* these expressions */  
  printf("%s\n",string2);  /* will produce the */  
  printf("%s\n",p_string);  /* same result */  
}
```
Pointers (II)

- Pointers can point to C functions
  - The pointer will point to the memory address that stores the first instruction of the function
  - Our knowledge of assembly language makes this idea easier to understand, doesn’t it?

- Pointers and dynamic memory allocation
  - Sometimes the length of an array is unknown at compilation time
  - Using pointers and the malloc() family of instructions we can allocate memory at run-time

```c
#include <stdio.h>
#include <string.h>

void my_strcmp(char *a, char *b, int (*ptr)() ) {
    if ( !(ptr)(a,b) ) printf("EQUAL");
    else printf("DIFFERENT");
}

void main() {
    char c1[80], c2[80];
    int (*p)();
    p = strcmp; /* p points to the function strcmp() */
    gets(c1); /* get the strings */
    gets(c2); /* from the keyboard */
    my_strcmp(c1,c2,p);
}
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main(void) {
    char *c;
    int i;
    c = malloc(80); /* allocate 80 characters */
    if (c==NULL) { /* error handling */
        printf("error allocating memory\n");
        exit(1);
    }
    gets(c); /* get a string from keyboard*/
    for (i=strlen(c)-1; i>=0; i--) {
        putchar(c[i]); /* display each element of string */
    }
    free(c); /* deallocate memory !!!!!!! */
}
```
Structures in C

- C allows definition of non-homogeneous data types with multiple fields, called structures

```c
struct struct_def {
    type field1;
    type field2;
    ...
    type fieldn;
};
struct struct_def struct_instance;
```

- Fields in a structures can be accessed using ‘dot’ notation or ‘arrow’ notation

```c
struct employee {
    char      family_name[30];
    char      first_name[30];
    long int  stipend;
};
struct employee gra, *ptr;
gra.stipend   = 1000;    /* these two expressions */
ptr->stipend  = 1000;    /* are equivalent */
```

- Structure fields can be arrays and we can define arrays of structures

```c
struct employee staff[200];
strcpy(staff[0].family_name, "Doe");
strcpy(staff[0].first_name, "John");
staff[0].stipend = 10000;
```
Cross-compilation

- To generate the assembly code from a C program you can use the cross-compiler provided in the CD-ROM
  - Cross-compiler is located in:
    \[ CD\_drive:\C\FILES\I2DEMO\ITOOLS\X\c68332.exe \]
  - To cross-compile
    - \texttt{C68332 filename.c -no -i}
    - where:
      - \texttt{filename.c} is the name of the source file
      - \texttt{-no} is an option that suppresses compiler optimization
      - \texttt{-i} is an option that control the format of the output
    - Read the “\texttt{CD\_drive:\C\readme.txt}” file for more information
int adder(int x, int y) {
    return x + y;
}

void main (void) {
    register int a, b, c;
    a = 1; b = 2;
    c = adder(a, b);
}
C and assembly language: example 2

void swap (int a, int b)  
{   
  int temp;  
  temp = a;  
  a = b;  
  b = temp;  
}  
void main (void)  
{   
  int x = 2, y = 3;  
  swap (x, y);  
}

This code does not work because the subroutine does not update the proper locations in the stack (the ones labeled with x and y)
C and assembly language: example 3

```c
void swap (int *a, int *b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
void main (void) {
    int x = 2, y = 3;
    swap(&x, &y);
}
```

```assembly
*1  void swap (int *a, int *b)  SECTION  S_swap,,"code"
    * Parameter a is at 8(A6)
    * Parameter b is at 12(A6)
    * Variable temp is at -2(A6)
    XDEF  _swap
    XREF  __main
    _swap
    000000  4e56ffee  _swap
    LINK  A6,#+-2
    __P1  EQU $000004
    *2    {
    *3      int temp;
    *4      temp = *a;
        000000 + __P1  286e0008
        000004 + __P1  3d54ffee
    *5      *a = *b;
        000008 + __P1  206e000c
        00000c + __P1  3890
    *6      *b = temp;
        00000e + __P1  286e000c
        000012 + __P1  38aeffee
    *7    }
        00001a  4e5e
        00001c  4e75
        UNLK  A6
        RTS
    * Function size = 30
*8  void main (void)  XREF  __main
    XDEF  _main
    _main
    00001e  4e56fffc  _main
    LINK  A6,#+-4
    __P2  EQU $000022
    *9    {
    *10       int x = 2, y = 3;
        000000 + __P2  3d7c0002ffee
        000006 + __P2  3d7c0003ffee
    *11       swap(&x, &y);
        00000c + __P2  48e0ffee
        000010 + __P2  48e0ffee
        000014 + __P2  4eb9________
    *12    }
        00003a  4e5e
        00003c  4e75
        UNLK  A6
        RTS
        _dgroup  data
        END
    * Function size = 34
    * bytes of code = 64
*13
```
C and assembly language: recursion

int factorial(int n)
{
    if (n==1)
        return (1);
    else
        return(factorial(n-1)*n);
}

void main()
{
    int y, count = 2;
    y = factorial(count);
}

*1 int factorial(int n)
SECTION S_factorial,,"code"
XDEF _factorial
XREF __main
SP,A6
OLD_A6 OLD_A6
OLD_A6 OLD_A6
RET RET
00 01
00 00
00 02
00 02
OLD_A6 OLD_A6
OLD_A6 OLD_A6
RET_ADDR RET_ADDR
RET_ADDR RET_ADDR
00 02
00 02
OLD_A6 OLD_A6
OLD_A6 OLD_A6
--- ---

*1 int factorial(int n)
SECTION S_factorial,,"code"
XDEF _factorial
XREF __main

* Parameter n is at 8(A6)

+00 0000 4e560000 _factorial LINK A6,#0
+00 0006 + _factorial 6600___ EQU $000004
+03 if (n==1)
+00 0000 + ___P1 0c6e00010008 CMPI #1,8(A6)
+06 + ___P1 6600____ BNE L1
+09 return (1);
+00 0000a + ___P1 7001 MOVEQ.L #1,D0
+00 0000c + ___P1 6000____ BRA L2
+0b L1
+0d return( factorial(n-1) * n );
+00 00010 + ___P1 322e0008 MOVE 8(A6),D1
+00 00014 + ___P1 2f01 MOVE.L D1,-(A7)
+00 00018 + ___P1 3f01 MOV E L D1,-(A7)
+00 0001a + ___P1 4eb9________ JSR _factorial
+00 00022 + ___P1 548f ADDQ.L #2,A7
+00 00024 + ___P1 c1f1 MULS D1,D0
+00 0002a 4e5e UNLK A6
+00 0002c 4e75 RTS

* Function size = 46

*8 void main()
SECTION S_main,,"code"
XDEF _main
XREF __main

* Variable y is at -2(A6)
* Variable count is at -4(A6)

+00 0002a 4e56fff8 LINK A6,#-4
+00 0002c 4e75 RTS

* Function size = 46

*8 void main()
SECTION S_main,,"code"
XDEF _main
XREF __main

+00 0002a 4e56fff8 LINK A6,#-4
+00 0002c 4e75 RTS

* Variable y is at -2(A6)
* Variable count is at -4(A6)

* Function size = 46

*8 void main()