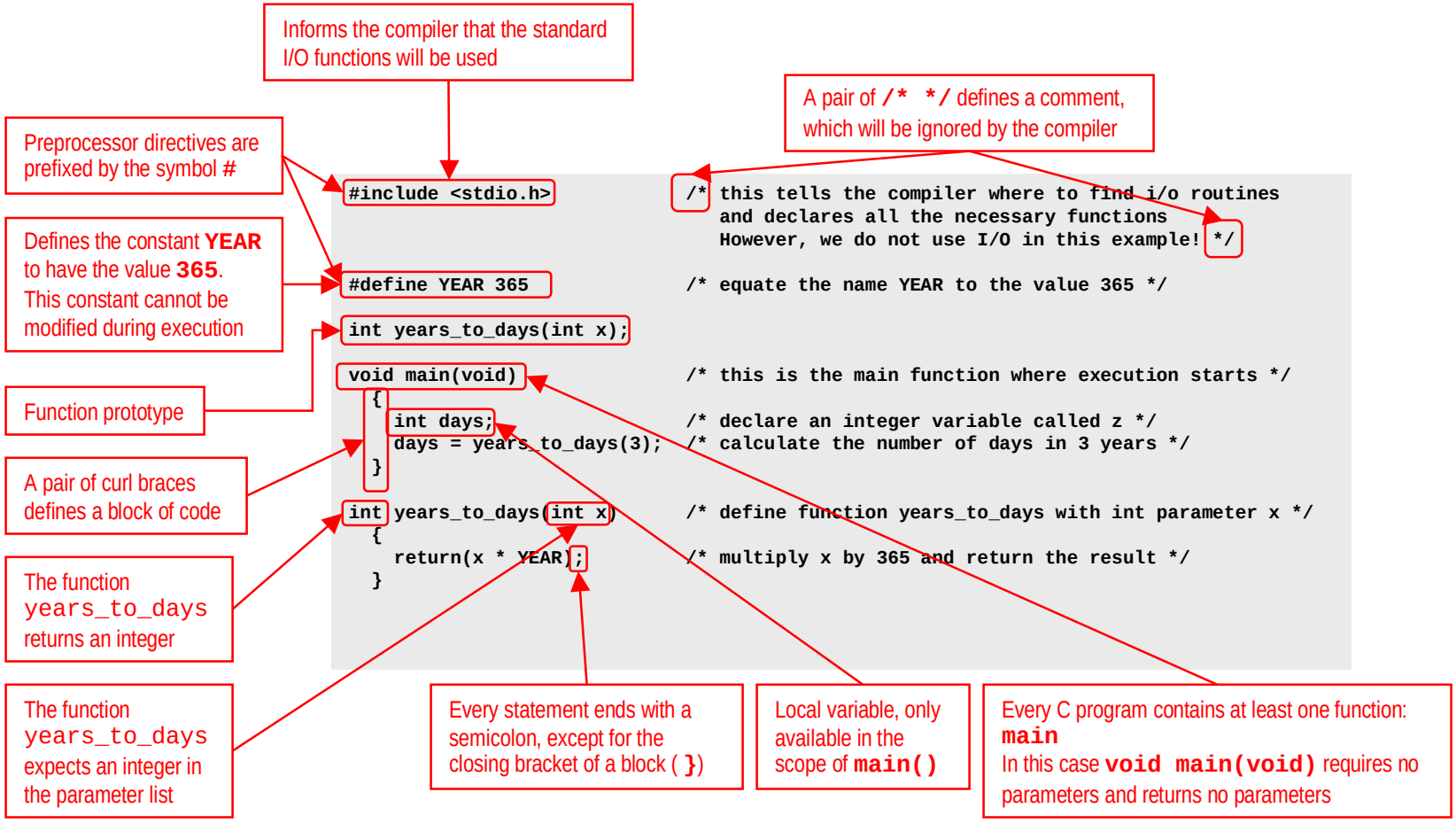
Lecture 8: C language

- **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



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

Type	Width (bits)	Minimum range
`char`	8	-127 to 127
`unsigned char`	8	0 to 255
`signed char`	8	-127 to 127
`int`	16	-32,767 to 32,767
`unsigned int`	16	0 to 65,535
`signed int`	16	Same as int
`short int`	16	Same as int
`unsigned short int`	8	0 to 65,535
`signed short int`	8	Same as short int
`long int`	32	-2,147,483,647 to 2,147,483,647
`signed long int`	32	--2,147,483,647 to 2,147,483,647
`unsigned long int`	32	0 to 4,294,967,295
`float`	32	Six-digit precision
`double`	64	Ten-digit precision
`long double`	128	Ten-digit precision

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 of afterwards**

```
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)
  {
     int j;             /* Integer j is local to this block within function main */

  }                     /* This is the point at which integer j ceases to exist */
```

C operators

Type	Operator	Action
Arithmetic	-	Subtraction
	+	Adittion
	*	Multiplication
	/	Division
	%	Modulus
	--	Decrement (by 1)
	++	Increment (by 1)
	+=	Increment (a+=b means a=a+b)
	-=	Decrement (a-=b means a=a-b)
Relational	>	Greater than
	>=	Greater than or equal to
	<	Less than
	<=	Less than or equal to
	==	Equal to
	!=	Different from
Logic	&&	AND
	||	OR
	!	NOT
Bit-wise	&	AND
	|	OR
	^	XOR
	~	NOT
	>>	Right shift
	<<	Left shitf
Miscellaneus	?	Ternary (y=x>9?100:200)
	& and *	Pointer operators
	sizeof	Width of a datatype (in bytes)
	. and ->	Acceess to structures
	[]	Access to arrays

Precedence	Operator
Most ↑	( ) [ ] -> .
	! ~ ++ -- - (cast) * & sizeof
	/ %
	<< >>
	< <= > >=
	== !=
	&
	|
	&&
	||
	?
	= += -= *= /=
Least	,

Loops and iterations

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

if-else

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

while, do-while

```
while (expression) {
  block;
}

do {
block;
} while (expression);
```

goto

```
goto label;
block1;
label:
block2;
```

for

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

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

switch-case

```
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**

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

- **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

```
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  */
}
```

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

```
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?

```
#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);
}
```

- **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

```
#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;t>=0;t--) {
    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**

```
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**

```
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**

```
struct employee staff[200];

strcpy(staff[0].family_name, “Doe”);
strcpy(staff[0].first_name, “John”);
staff[0].stipend = 10000;
```

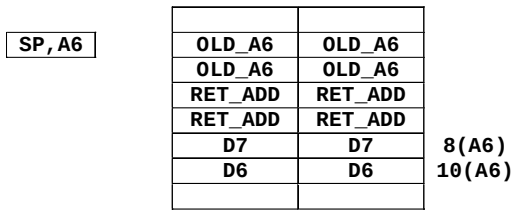
Cross-compilation

- **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
    - **`C68332 filename.c -no -i`**
    - where:
      - **`filename.c`** is the name of the source file
      - **`-no`** is an option that suppresses compiler optimization
      - **`-i`** is an option that control the format of the output
    - Read the "**`CD_drive:\C\readme.txt`**" file for more information

C and assembly language: example 1

```
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);
}
```




```
* Feb  7 1999  18:46:00
* bc sid : @(#)bc68000.PL 5.133.1.2
* options : -no -p -t 68332 -nv
* cpf sid : @(#)cpf.PL 6.66.1.4
*1      int adder(int x, int y) {

* Parameter x is at 8(A6)
* Parameter y is at 10(A6)

                                                    SECTION         S_adder,,"code"

                                                    XDEF            _adder
                                            _adder
                000000          4e560000            LINK            A6,#0
                                            __P1    EQU             $000004
*2        return x + y;
                000000 + __P1   322e0008            MOVE            8(A6),D1
                000004 + __P1   d26e000a            ADD             10(A6),D1
                000008 + __P1   3001                MOVE            D1,D0
*3      }
                00000e          4e5e                UNLK            A6
                000010          4e75                RTS
* Function size = 18
*4
*5      void main (void) {

                                                    XREF            __main

* Variable a is in the D7 Register
* Variable b is in the D6 Register
* Variable c is in the D5 Register

                                                    XDEF            _main
                                            _main
                                            __P2    EQU             $000012

*6        register int a, b, c;
*7        a = 1; b = 2;
                000000 + __P2   7e01                MOVEQ.L         #1,D7
                000002 + __P2   7c02                MOVEQ.L         #2,D6
*8        c = adder(a, b);
                000004 + __P2   3f06                MOVE            D6,-(A7)
                000006 + __P2   3f07                MOVE            D7,-(A7)
                000008 + __P2   4eb9________        JSR             _adder
                00000e + __P2   3a00                MOVE            D0,D5
                000022          588f                ADDQ.L          #4,A7
*9      }
                000024          4e75                RTS
* Function size = 20
* bytes of code = 38
* bytes of idata = 0
* bytes of udata = 0
* bytes of sdata = 0

                                                    _dgroup         data
                                                    END
```

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);
   }
```

SP	TMP	00	02	-2(A6)
A6		OLD_A6	OLD_A6	
		OLD_A6	OLD_A6	2(A6)
		RET_ADD	RET_ADD	4(A6)
		RET_ADD	RET_ADD	6(A6)
	A	00	02 (03)	8(A6)
	B	00	03 (02)	10(A6)
	Y	00	03	
	X	00	02	
		OLD_A6	OLD_A6	
		OLD_A6	OLD_A6	

This code does not work because the subroutine does not update the proper locations in the stack (the ones labeled with x and y)

```
*1      void swap (int a, int b)
                                                        SECTION     S_swap,,"code"

* Parameter a is at 8(A6)
* Parameter b is at 10(A6)
* Variable temp is at -2(A6)
                                                        XDEF        _swap
                                              _swap
                000000          4e56fffe              LINK        A6,#-2
                                              __P1      EQU         $000004
*2          {
*3             int temp;
*4             temp = a;
                000000 + __P1   3d6e0008fffe          MOVE        8(A6),-2(A6)
*5             a = b;
                000006 + __P1   3d6e000a0008          MOVE        10(A6),8(A6)
*6             b = temp;
                00000c + __P1   3d6efffe000a          MOVE        -2(A6),10(A6)
*7          }
                000016          4e5e                  UNLK        A6
                000018          4e75                  RTS
* Function size = 26
*8      void main (void)
                                                        XREF        __main

* Variable x is at -2(A6)
* Variable y is at -4(A6)
                                                        XDEF        _main
                                              _main
                00001a          4e56fffc              LINK        A6,#-4
                                              __P2      EQU         $00001e
*9          {
*10            int x = 2, y = 3;
                000000 + __P2   3d7c0002fffe          MOVE        #2,-2(A6)
                000006 + __P2   3d7c0003fffc          MOVE        #3,-4(A6)
*11            swap (x, y);
                00000c + __P2   3f3c0003              MOVE        #3,-(A7)
                000010 + __P2   3f3c0002              MOVE        #2,-(A7)
                000014 + __P2   4eb9________          JSR         _swap
*12         }
                000038          4e5e                  UNLK        A6
                00003a          4e75                  RTS
* Function size = 34
* bytes of code = 60
* bytes of idata = 0
* bytes of udata = 0
* bytes of sdata = 0


                                                        _dgroup     data
                                                        END

*13
```

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);
   }
```

SP	TEMP	00	02	-2(A6)
A6		OLD_A6	OLD_A6	
		OLD_A6	OLD_A6	2(A6)
		RET_ADD	RET_ADD	4(A6)
		RET_ADD	RET_ADD	6(A6)
		&x	&x	8(A6)
		&x	&x	10(A6)
		&y	&y	12(A6)
		&y	&y	14(A6)
	Y	00	03 (02)	16(A6)
	X	00	02 (03)	18(A6)
		OLD_A6	OLD_A6	
		OLD_A6	OLD_A6	

```
*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
                                            _swap
            000000          4e56fffe                LINK       A6,#-2
                                            __P1    EQU        $000004
*2          {
*3             int temp;
*4             temp = *a;
            000000 + __P1   286e0008                MOVEA.L    8(A6),A4
            000004 + __P1   3d54fffe                MOVE       (A4),-2(A6)
*5             *a = *b;
            000008 + __P1   206e000c                MOVEA.L    12(A6),A0
            00000c + __P1   3890                    MOVE       (A0),(A4)
*6             *b = temp;
            00000e + __P1   286e000c                MOVEA.L    12(A6),A4
            000012 + __P1   38aefffe                MOVE       -2(A6),(A4)
*7          }
            00001a          4e5e                    UNLK       A6
            00001c          4e75                    RTS
* Function size = 30
*8      void main (void)
                                                    XREF       __main
* Variable x is at -2(A6)
* Variable y is at -4(A6)
                                                    XDEF       _main
                                            _main
            00001e          4e56fffc                LINK       A6,#-4
                                            __P2    EQU        $000022
*9          {
*10            int x = 2, y = 3;
            000000 + __P2   3d7c0002fffe            MOVE       #2,-2(A6)
            000006 + __P2   3d7c0003fffc            MOVE       #3,-4(A6)
*11            swap(&x, &y);
            00000c + __P2   486efffc                PEA.L      -4(A6)
            000010 + __P2   486efffe                PEA.L      -2(A6)
            000014 + __P2   4eb9________            JSR        _swap
*12         }
            00003c          4e5e                    UNLK       A6
            00003e          4e75                    RTS
* Function size = 34
* bytes of code = 64
                                                    _dgroup    data
                                                    END
*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);
  }
```

SP,A6

OLD_A6	OLD_A6
OLD_A6	OLD_A6
RET	RET
RET	RET
00	01
00	00
00	02
OLD_A6	OLD_A6
OLD_A6	OLD_A6
RET_ADD	RET_ADD
RET_ADD	RET_ADD
00	02
00	02
OLD_A6	OLD_A6
OLD_A6	OLD_A6
---	---

```
*1      int factorial(int n)
                                                        SECTION     S_factorial,,"code"
* Parameter n is at 8(A6)
                                                        XDEF        _factorial
                                          _factorial
                 000000         4e560000                LINK        A6,#0
                                          __P1          EQU         $000004
*3          if (n==1)
                 000000 + __P1  0c6e00010008            CMPI        #1,8(A6)
                 000006 + __P1  6600____                BNE         L1
*4            return (1);
                 00000a + __P1  7001                    MOVEQ.L     #1,D0
                 00000c + __P1  6000____                BRA         L2
                                          L1
*5          else return( factorial(n-1) * n );
                 000010 + __P1  322e0008                MOVE        8(A6),D1
                 000014 + __P1  2f01                    MOVE.L      D1,-(A7)
                 000016 + __P1  5341                    SUBQ        #1,D1
                 000018 + __P1  3f01                    MOVE        D1,-(A7)
                 00001a + __P1  4eb9________            JSR         _factorial
                 000020 + __P1  548f                    ADDQ.L      #2,A7
                 000022 + __P1  221f                    MOVE.L      (A7)+,D1
                 000024 + __P1  c1c1                    MULS        D1,D0
                                          L2
*6        }
                 00002a         4e5e                    UNLK        A6
                 00002c         4e75                    RTS
* Function size = 46
*8      void main()
                                                        XREF        __main
* Variable y is at -2(A6)
* Variable count is at -4(A6)
                                                        XDEF        _main
                                          _main
                 00002e         4e56fffc                LINK        A6,#-4
                                          __P2          EQU         $000032
*9        {
*10         int y, count = 6;
                 000000 + __P2  3d7c0006fffc            MOVE        #2,-4(A6)
*11         y = factorial(count);
                 000006 + __P2  3f3c0006                MOVE        #2,-(A7)
                 00000a + __P2  4eb9________            JSR         _factorial
                 000010 + __P2  3d40fffe                MOVE        D0,-2(A6)
*12       }
                 000046         4e5e                    UNLK        A6
                 000048         4e75                    RTS
* bytes of sdata = 0
                                                        _dgroup     data
                                                        END
*13
```