Portability of compiler

[START 3]

◆ BCPL compiler [historically] composed of 2 parts

Source BCPL

The front end converts source BCPL into OCODE.

OCODE made up machine code for a hypothetical machine (Stack based).

The front end performs the following parts of the compilation process:

◆ Lexical Analysis
  Recognise meaningful tokens in the language
  + - * / $( $) for while

◆ Syntax analysis
  Check that the formation of the BCPL program follows the grammatical rules of the language.
  Grammar defined in a form of BNF

◆ Semantic analysis
  Check that the semantics are correct:
  Has a variable been declared
  Is the variable of the correct type

◆ Code generation
  Generate OCODE rather than the machine code for a specific machine.
To implement a compiler for BCPL on a new machine with a different machine architecture (T)

Write a new back end to convert OCODE to the assembly code of the target machine (T).

When this works

Convert the front end of the compiler [in OCODE] to (T's) assembly language.

Convert the back end of the compiler that converts OCODE to the target machines T's assembly code [in OCODE] to the target machines (T's) assembly language.

Now have a BCPL compiler front and back end that can run on the new target machine.

Of course instead of producing source assembly code, which we would need to run through the target machines assembler, we could have produced executable binary code directly.

Usually producing executable binary code is more difficult than producing source assembly code.

[END 3]
A translator for OCODE (OCODE-> arm m/c code) is written in the language BCPL and the program is compiled and can be executed on a x86 machine to generate arm code.

A translator for OCODE (OCODE-> arm m/c code) is written in the language BCPL and then converted to OCODE. The OCODE is then executed by an OCODE interpreter running on a X86 machine. Writing an interpreter is [usually] simpler than writing a program to convert OCODE instructions into machine code.
◆ A compiler for the language BCPL on a PC.

A cross compiler for the language BCPL to arm code. Again easier if produce arm assembly code.

◆ It is usually easier to produce source x86 assembly code, and then use the 'system' assembler.
Porting the compiler to an arm machine

The OCODE output from the BCPL compiler is moved to an arm machine and converted on the arm machine into an arm program using a program running on the arm processor that converts OCODE to the arm machine code. This program is written in a Language L.

Usually the language L would of course be BCPL. So how did we get a BCPL program running on the arm machine before we ported the compiler to the arm machine?

[Usually would be written on a machine which has BCPL as an available programming language. Once working the first productive use made of the program would be to convert itself in OCODE into arm machine code]

Hence the term bootstrapping, lifting oneself of the ground by pulling on your boot laces.

We now have the BCPL compiler on the arm machine.
Full Bootstrap
Do not have access to a compiler for the language (L) that we wish to implement on the host machine.

Hence need to write the compiler in a language that we have access to. Then if we wish can convert this language to (L).

Half Bootstrap
Do have access to a compiler for the language that we wish to implement on the host computer. Of course the language does not as yet exist on the host computer.
Overview

Types of programming languages

- Imperative
  FORTRAN, Algol 60, COBOL, C, Pascal

- Object-Oriented
  Simula 67, Java, C++, C#

- Functional
  ML, Hascal, Scala (Mixed)
  Composed of functions, with no destructive assignment or conventional loops

- Logical
  Prolog
  Program draws conclusions from a list of premises
Fortran (1957)

- Consists of separately compiled subroutines

- Subprograms can not be nested or called recursively, stack not used in function call

- Variables could be shared across subroutines by using common blocks. (However no checking that common blocks represent the same storage between functions is made)

```
MASTER EXAMPLE
COMMON /SHARED/ A,B,C
A = 2
B = 3
C = 4
RES = SUM
END

REAL SUBROUTINE SUM
COMMON /SHARED/ A,B,C
SUM = A + B + C
RETURN
END
```

- Subprogram visibility local data plus common data.

- Data structures - none
  Arrays, fixed bounds

[END 4]
Format of Language
Uses fixed columns, as when FORTRAN was first created input to a computer was [usually] on punched cards.

Columns 1-5
A numeric label

Column 6
A C in column 6 indicated that the line is a continuation of the previous line.

Columns 7-72
For a FORTRAN Statement

A fragment of FORTRAN code to add together the first 10 numbers.

DO 10 I=1,10
    IRESULT = IRESULT + I
10    CONTINUE

In FORTRAN originally there was no need to declare variables, a variable starting with I,J,K,L,M,N is an integer all other variables are real.

Is this a good idea.

Other statements:

IF (value) 10,20,30
    GOTO label 20 if value -ve, Label 20 if 0, label 30 if +ve.
GOTO 20
    Transfer control to label 20
A = B + C
    Assignment statement
Cobol (1959)

- 4 Main divisions
  - Identification division
  - Environment division
    - Configuration section
      Type of computer to be used
    - Input-Output section
      Which files are to be used
  - Data division
    Data used in program
  - Procedure division
    What is to be done
    
    \texttt{MULTIPLY RATE BY HOURS GIVING PAY}
Format of Language {Historic}
Uses fixed columns, as when COBOL was first created input to a computer was [usually] on punched cards.

Approximately 300 reserved words

Columns 1-6
Card line number, ignored by compiler

Column 7
* A comment follows on this card
- Continuation of previous line

Columns 8-11
DIVISION, SECTION,

Columns 12-72
Code, Data, etc.
Stylised example of early COBOL program

**IDENTIFICATION DIVISION**
- **PROGRAM-ID.** TOTAL.
- **AUTHOR.** MAS.

**ENVIRONMENT DIVISION**
- **INPUT-OUTPUT SECTION**
  - **FILE-CONTROL**
    - SELECT CF ASSIGN TO 'S01' UNIT-RECORD CREADER.

**DATA DIVISION**
- **FILE SECTION**
  - **FD** CF
    - DATA RECORD IS CUSTOMER-RECORD.
      - 01 CUSTOMER-RECORD.
        - 05 NAME PICTURE A(30).
        - 05 DOB
          - 10 DAY PICTURE 99.
          - 10 MONTH PICTURE AAA.
          - 10 YEAR PICTURE 9999.
        - 05 CUSTOMER-ID PICTURE AA999999.
        - 05 BALANCE PICTURE 9999.99.
        - 05 CUSTOMER-USER-NAME PICTURE XXXX.

**WORKING-STORAGE SECTION.**
- 77 TOTAL PICTURE 99999.99 VALUE ZEROS.
- 77 EOF PICTURE A VALUE 'N'

**PROCEDURAL DIVISION**
- DISPLAY "Start of program".
- OPEN INPUT CF.
- PERFORM INPUT-DATA UNTIL EOF = 'T'.
- DISPLAY TOTAL.
- CLOSE CF.
- STOP RUN.

**INPUT-DATA.**
- READ FILE-1 AT END MOVE 'Y' TO EOF.
- ADD CUSTOMER-RECORD.BALANCE TO TOTAL.
**ALTER** statement

A way of creating self modifying code. This is very bad, as it makes following a program very difficult and does not allow sharing of code [segments] for multiple users of the application.

Self modifying code is when a program changes the code that its executing dynamically at run time.

```cobol
PERFORM PARA-1 THRU PARA-99 UNTIL EOF = 'T'.

PARA-1
   GOTO PARA-10.

PARA-10
   * Do some initialisation

PARA-50
   **ALTER** PARA-1 TO PARA-60.
   * Change the single goto in the para PARA-1
   * to GOTO the para PARA-60

PARA-60
   * Now do some work
   * Read record at end set EOF to 'T'

PARA-99
   **EXIT**

**EXIT** is treated as a dummy statement.
Note the danger of
   **PERFORM** PARA-n THRU PARA-m
Not immediately obvious that this is a loop.

History footnote:

**COBOL** 1985 **ALTER** Deemed obsolete
**COBOL** 2002 **ALTER** Deleted
Algol 60 (Algol 58)

- Block structured language (N levels)
- Procedures/ functions can be nested and called recursively
- Defined unambiguously (syntax) using BNF

Below is extracted from the Algol 60 report

Meta symbols

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

digit is 0 or 1 or 2 ...

<identifier> ::= <letter> | <identifier><letter> | <identifier> <digit>

<statement> ::= <unconditional statement> | <conditional statement> | <for statement>

<unconditional statement> ::= <basic statement> | <compound statement> | <block>
Semantics of language defined in English.

Note left recursion used in definition. Easier to process if right recursive, i.e. recursion at the end.

**So rewrite**

\[
\text{identifier} ::= \text{letter} | \text{identifier}\text{letter} | \text{identifier}\text{digit}
\]

\[
\text{as}
\]

\[
\text{identifier} ::= \text{letter} \text{identifierRest}
\]

\[
\text{identifierRest} ::= \text{digit} \text{identifierRest} | \text{letter} \text{identifierRest} | \text{empty}
\]

Try matching using the above [modified] grammar

CostOfItem2
A formal grammar that contains left recursion in general can not be parsed by a LL(k) parser. An LL(k) parser parses from Left to right performing left most derivation first with k symbols of look ahead.

Most languages constructed to be able to be parsed by a LL(1) parser. Which needs 1 symbol look ahead.
A way of converting a left recursive grammar rule to a right recursive grammar rule.

To recognise
\[ B \ BZ \ BZZ \ BZZZ \ldots \quad /\!/ \quad BZ^* \]

Essentially replace the rule
\[
<A> ::= <A> \ Z \mid B \quad /\!/ \quad BZ^*
\]
by
\[
<A> ::= B<AP> \quad /\!/ \\
<AP> ::= Z<AP> \mid e \quad /\!/ \quad BZ^*
\]

**Explanation**
\[
Z^* \quad /\!/ \quad \text{Zero or more } Z's \\
e \quad /\!/ \quad \text{Empty}
\]
Syntax diagram to recognise an identifier
Its like a flow chart, Used by Niklaus Wirth to describe the syntax of Pascal.

Constructed from the modified BNF which is right recursive.

Note, only 1 symbol look ahead is required to decided which path to take.

If no path can be taken then this represents a syntax error in the program that is being checked.

In some cases the BNF rules would be converted in a function/ method that would be recursively called.

Hence [relatively] easy to write a parser for a program.
Syntax Diagram of part of the PL/0 language

PL/0 is a made up language designed to teach about compiler construction

Usually there is a lexical analysis phase to a compiler which splits a program up into individual tokens which are classified as:

- **reserved word**: begin do else end if then while ...
- **Special symbol**: ( ) <= >= := ...
- **identifier**: A token which is not a reserved word or symbol [Identifier].

You can easily convert this into a program to recognise part of the PL/0 language.

[END 5]
Example Algol 60 program

```
begin
  procedure proc1(p1);
    value p1; integer p1;
  begin
    integer procedure proc2(p2);
      integer p2;
    begin
      integer local2;
      local2 := p2 + 7;
      proc2 := p2;
    end;
    integer local1;
    local1 := proc2(p1 * 2);  proc1(p1);
  end;

  procedure proc3;
  begin
  end;

  integer local3;
  proc1(10); proc3;
end;
```

◆ In body of `proc1` can see:
  `proc1, proc2, p1, local1`

◆ In body of `proc2` can see:
  `proc1, proc2, p1, p2, local2`

◆ In body of program can see:
  `proc1, proc3, local3`
Note a parameter can be called by value (copy) or by name. In call by name, the actual parameter is textual substituted into the called procedure.

Consider:

```pascal
procedure swap( a, b )
integer a,b;
begin
  integer tmp;
  tmp := a; a:= b; b := tmp;
end;

swap( x, y );
  Swaps x and y

swap( i, element[i] );

  Does not do what you expect as the code executed is [fragment]
  tmp := i;
  i:= element[i]
  element[i] := tmp;
```

This can be exploited in a technique called 'jensen's device'.

This [parameter passing mechanism] is not usually provided in modern programming languages.

In Algol 60 I/O was not part of the definition of the language. How I/O is performed is left to the discretion of the language implementor.
Consequences of [many] lexical levels is the need to run a 'display' to keep track of what is visible when you call a function/ procedure. There will be potentially multiple lexical levels.

Hence calling a function/ procedure can be expensive.

Remember this has to be able to cope with recursive functions/ procedures calls as well.

So the code overhead for writing short functions/ procedures will be high. Think about some methods in java which only execute a small number of statements.
A local variable in a procedure can be *own* in which case, the contents are not lost on exit and will be preserved for subsequent calls of the procedure.

Why have this?

static variables Java

Data structures - none
Arrays: dynamic bounds possible, if in an inner block
Only numbers integer and real

A comment was a statement

```java
comment Some comment ;
```

Was this a good idea?
BCPL background

BCPL Basic Combined Programming Language

1966 Designed by Martin Richards Cambridge
First compiler 1967

Paradigm Procedural
Typeing typeless (everything is a word/bit pattern)
Influenced by CPL
Influenced B, C

◆ Block structured language (2 levels)
  variables either global or local.

◆ Functions/ procedures cannot be nested but can be called recursively
Example of a BCPL program

To print the 12 times table

GET "LIBHDR" //Include library header file

LET START() = VALOF //Function
$(
   WRITEF("12 Times Table *N")
   PRINTTABLE( 12 ) //; not required
   RESULTIS 0 //at end of line
$)

AND PRINTTABLE( N ) BE //Procedure
$(
   LET MAX = 12 //Declaration
   FOR I = 1 to MAX DO //I Read only
   $(
      WRITEF( "%N ** %N = %N*N", N, I, N*I )
   )
)$(

// In WRITEF
// *N Stands for the newline character
// ** Stands for the * character
// Example of LIBHDR

GLOBAL
$(
  RDCH: 13  // CH := RDCH()
  WRCH: 14  // WRCH(CH)
  WRITEN: 76  // WRITEN(F, A1, A2, ..., A11)
  GETVEC: 87  // V := GETVEC(SIZE)
  FREEVEC: 88  // FREEVEC(V)
)$

MANIFEST
$(
  ENDSIZECH = -1
  BYTESPERWORD = 4
  BITSPERWORD = 32
  MAXINT = #X7FFFFFFF
  MININT = #X80000000
  MAXSTRLENGTH = 255
  FIRSTFREEGLOBAL = 100
  UG = FIRSTFREEGLOBAL
)$
:=       // Assignment

Operators

+ _ * /       // Normal arithmetic operators
< <= = >= >   // Relational operators

<< >>       // Left and right shift operators

@WHERE       // Deliver the address of WHERE
!P           // Deliver the contents of the location
             whose address is held in P
P!2          // Same as ! (P+2)
             // Allows array indexing

&            // Logical and
            // Logical or

Other features

A<B<C       // Equivalent to A<B & B<C

Other constructs

IF e DO
UNLESS e DO       // Inverse
TEST e THEN c1 OR c2
WHILE e DO
UNTIL e DO        // inverse
c REPEAT          // Endless loop
SWITCHON e INTO
    CASE 1 :
    DEFAULT

C Background

Denis Ritchie, Ken Thomson
Early developments 1969-1973 (Most creative 1972)

K&R C (Book The C programming language Brian Kernighan & Denis Ritchie)
added amongst other things
+= changed to += previously: i+=10; // what does it mean

Ansi C standard 1989
   Inline functions, // Coments, long long int, ....
C11 ISO 9899:2011
   Multi-threading, Improved Unicode support, ...

Paradigm                Procedural
Typeing                 static, weak unsafe
Influenced by           B, BCPL
Influenced              C++, Java, C#
C (Early 1973)

- Block structured language (2 levels) variables either global or local.
- Functions cannot be nested but can be called recursively

```c
int global1;
int twice( int p1 )
{
    return p1 + p1;
}

int main( char *argv[] )
{
    int local1;
    local1 = twice( 10 );
}
```

- Separate compilation allowed
  
  Header files used to facilitate sharing of variables between separately compiled units

```c
extern int data; /* In header file */
```

```c
int data; /* C code file */
```

- Data structures - yes
  
  Arrays: no subscript checking, no dynamic bounds
C Major features

- Emphasises code quality, as good as assembler
  ```c
  int a;
  a = ( a >> 2 ) & 0xFF;
  if (Condition) Statement;
  ```
- { } used as begin end.
- Weak typing
  ```c
  int i = 1.2;
  ```
- Can easily access machine level features
  - Explicit access to pointers
  - Operators & address, * indirection
  ```c
  while ( *copy++ = *p++ ); // copy
  ```
- Standard library (Part of language)
  - Small, No GUI
- Separate compilation, through individual files, linking not type safe

C Application area

- Operating systems
  - Applications where code quality matters (size & speed: )
C example: cat

```c
#include <unistd.h>      // read/write
#include <sys/file.h>    // open/close values
#include <string.h>      // strlen

const int BUFSIZE=4096;

void fail( char file[] );
void copy( char file[] );

int main( int argc, char *argv[], char *env[] )
{
    for ( int i=1; i<argc; i++ )
    {
        copy( argv[i] );
    }
}

void copy( char file[] )
{
    int fd = open( file, O_RDONLY, 0 );
    if ( fd >= 0 )
    {
        bool eof = false;
        char buf[BUFSIZE];
        while ( !eof )
        {
            int bytes = read( fd, buf, BUFSIZE );
            if ( bytes > 0 )
                write( 1, buf, bytes );
            else
                eof = true;
        }
        close( fd );
    }
    else
    {
        fail( file );
    }
}

void fail( char file[] )
{
    write( 2, "cat: ", 5 );
    write( 2, file, strlen( file ) );
    write( 2, " no such file or directory", 27 );
}
```
C

lvalue is address of variable
rvalue is contents of variable

& delivers the lvalue
* delivers the contents of a variable given its lvalue

```
int mem;
int *p = &mem;
*p = 99;
p[0] = 99;
p[1] = 99
```

Declare int variable
Declare p to contain a pointer to mem
Put 99 in variable mem
Write somewhere

Explicitly by programmer

```
int main()
{
    int *p_int;  // Pointer to an int or array of ints
    p_int = (int*) malloc( sizeof(int) * 10 ); // Array of 10 ints
    p_int[0] = 00; // Array index *(p_int + sizeof(int)*0)
    p_int[9] = 99; // Array index *(p_int + sizeof(int)*9)
    free( p_int );
}
```

- Memory allocation
  Allocated in bytes, hence sizeof(int) * 10

- Problems
  Possibility for error is high
  No array bound checking
strcpy( destination, source )

String copy src to dst: from arrays to pointers

```c
void strcpy3(char dst[], char src[]) {
    int i = 0;
    while (src[i] != '') {
        dst[i] = src[i];
        i++;
    }
    dst[i] = src[i];
}
```

```c
void strcpy2(char *dst, char *src) {
    while (*src != '') {
        *dst = *src;
        dst++; src++;
    }
    *dst = *src;
}
```

```c
void strcpy1(char *dst, char *src) {
    while (*src != '') {
        *dst++ = *src++;
    }
    *dst = *src;
}
```

Efficient code on register file architecture

```c
void strcpy(char *dst, char *src) {
    while (*dst++ = *src++);
}
```
Ada 95/2005 (Ada 83)

- Block structured language (N levels)
  Procedures functions may be nested and called recursively

- Separate compilation of procedures/ functions

```ada
with package_one; -- Make available
use package one; -- Do not need to prefix
procedure usefull( data: in Integer ) is
  Integer local;
begin
  -- Code body
end usefull;
```

- Package groups types, variables, procedures and functions that may be separately compiled into a unit that can be shared amongst other units.

Concept of specification and implementation

- Data structures - yes
  Arrays: Subscript checking, Dynamic bounds inner block
Ada 95 background

Study 1973-1974
US DOD annual software (>50% embedded) costs $3 billion
1977: 4 contractors selected to produce prototype languages
   (green, red, blue, yellow)
1979 May: Green selected (Named after Ada Lovelace)
Ada83
Ada95 = Ada83 + OO (Over simplification)
Ada2005 = Ada95 + ..

Paradigm Procedural, OO
Typeing static, strong
Influenced by Pascal
Influenced
Ada Major features

- Emphasises safety (Strong typing)

```
declare
    type TMark is range 0 .. 100;
    type TAge  is range 0 .. 125;
    mark : TMark;
    age  : TAge;
begin
    mark := -2;  -- Warning, compile time
    mark := age;-- Error, compile time
    mark := mark + mark/10;  -- RunTime check
end;
```

- Tasking
  Part of language, protected records, task priority, failure

- Can access machine level features

- Object-oriented -- Ada 95

- Ada 95
  Standard library (Part of language)
  Small, No GUI, No collection classes

- Package
  Specification, Implementation

Ada Application area

- Embedded systems, Real-time systems

- General purpose
with Ada.Text_IO;  -- Include this package
use Ada.Text_IO;  -- But do not have to prefix
                  -- components with package name

procedure main is

  type TMark is range 0 .. 100;
  type TAge  is range 0 .. 125;

  mark : TMark := 50;
  age  : TAge  := 20;

begin

  mark := -2;       -- Warning, RunTime failure
  mark := age;      -- Error, compile time
  mark := 90;       -- RunTime check
  mark := mark + 20;-- RunTime check

  declare
    subtype GoodMark is TMark range 40 .. 100;
    myMark : GoodMark;
  begin
    myMark := 40;     -- RunTime check
    myMark := 20;     -- Warning, RunTime failure
  end;

  declare
    subtype SummerTemp is Integer range 8 .. 30;
    subtype WinterTemp is Integer range -5 .. 12;
    summerDayTemp : SummerTemp := 10;
    winterDayTemp : WinterTemp := 12;
  begin
    IF winterDayTemp >= summerDayTemp then
      put("Unexpected!");
    end IF;
  end;

end main;
Ada example

```ada
with Ada.Text_io, Ada.Command_line;
use Ada.Text_io, Ada.Command_line;
procedure cat is
  fd  : Ada.Text_io.File_type;  -- File descriptor
  ch  : Character;            -- Current character
begin
  if argument_count >= 1 then
    for i in 1 .. argument_count loop  -- Repeat for each file
      begin
        open( File=>fd, Mode=>IN_FILE, -- Open file
          Name=>argument(i) );
        while not end_of_file(fd) loop  -- For each Line
          while not end_of_line(fd) loop-- For each character
            get(fd,ch); put(ch);       -- Read / Write char
          end loop;
          skip_line(fd); new_line;    -- Next line / new line
        end loop;
        close(fd);                    -- Close file
      exception
        when Name_error =>
          put("cat: " & argument(i) & " no such file" );
          new_line;
        when Status_error =>
          put("cat: " & argument(i) & " all ready open" );
          new_line;
      end;
    end loop;
  else
    put("Usage: cat file1 ... "); new_line;
  end if;
end cat;
```

[END 7]
Memory management

Life times of storage in block structured languages

◆Globals
Life time of the program

◆Local to a function
Life time of the active procedure/ function

◆Dynamic storage
Storage allocated when required

◆Life time decided by programmer

◆Life time decided by system, in effect as long as the storage is active
Block structured languages

A function /procedure call uses a stack to allocate space for locals

Local Variables
Held on the system stack

Global Variables

Program code
With dynamic memory allocation

A function procedure call uses a stack to allocate space for locals. In addition memory can be allocated for a user defined time on the stack.

- Local Variables
  Held on the system stack

- Dynamically allocated memory
  Held in the Heap

- Data
- Global Variables

- Text
- Program code
Problems with dynamic memory allocation

<table>
<thead>
<tr>
<th>Problem</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory leak</td>
<td>The storage that is allocated is not always returned to the system. For a program which executes for a long time, this can result in eventual out of memory error messages.</td>
</tr>
<tr>
<td>Accidentally using the same storage twice for different data items. Two pointers to same area</td>
<td>This will result in corrupt data in the program and probably a crash which is difficult to understand.</td>
</tr>
<tr>
<td>Corruption of the heap holding the data.</td>
<td>Most likely a program crash will occur some time after the corruption of the heap.</td>
</tr>
<tr>
<td>Time taken to allocate and de-allocate storage is not always constant.</td>
<td>There may be unpredictable delays in a real-time system. However a worst case Figure can usually be calculated.</td>
</tr>
</tbody>
</table>
## Dynamic memory allocation issues

<table>
<thead>
<tr>
<th>Process</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage reclamation implicitly managed by the system. Static de-allocation</td>
<td>No problem about de-allocating active storage.</td>
<td>May result in a program consuming large amounts of storage even though its actual use of storage is small. In extreme cases this may prevent a program from continuing to run.</td>
</tr>
<tr>
<td><strong>Ada</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage reclamation implicitly managed by the system. dynamic de-allocation</td>
<td>No problem about de-allocating active storage.</td>
<td>Cost of running de-allocation system.</td>
</tr>
<tr>
<td><strong>Java</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage de-allocation explicitly initiated by a programmer. <strong>Ada &amp; C++</strong></td>
<td>Prevents inactive storage consuming program address space.</td>
<td>If the programmer makes an error in the de-allocation then this may be very difficult to track down.</td>
</tr>
</tbody>
</table>
Dynamic memory de-allocation

- **Static De-allocation**
  De-allocation of dynamically allocated instances of type automatically done when type goes out of scope.

- **Dynamic de-allocation**
  De-allocation done when storage is no-longer accessible.

- **Programmers responsibility**
  De-allocation explicitly done by programmer

<table>
<thead>
<tr>
<th>Language</th>
<th>Static de-allocation</th>
<th>Dynamic de-allocation</th>
<th>Programmers responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>BCPL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Lisp</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Ada example dynamic storage

```
declare
    Max_Chs : constant := 7;
    type Gender  is ( Female, Male );
    type Height_Cm is range 0 .. 300;
    type Person;  --Incomplete declaration
    type P_Person is access Person;  --Access type

type Person is record
    Name   : String( 1 .. Max_Chs );  --Name as a String
    Height : Height_Cm := 0;         --Height in cm.
    Sex    : Gender;                --Gender of person
    Next   : P_Person;
end record;

People : P_Person;

begin
    People := new Person'("Mike   ", 183, Male, null);
    People.Next := new Person'("Corinna", 171, Female, null);

    --or

    People := new Person'("Mike   ", 183, Male,
                           new Person'("Corinna", 171, Female, null) );

end;
```

When the declaration for P_Person goes out of scope, dynamic storage associated with it (People) will be deallocated.
Packages in Ada

Specification
What the package does

```ada
package Class_account is
  type Account is tagged private;
  subtype Money is Float;
  subtype PMoney is Float range 0.0 .. Float'Last;

  procedure statement( the: in Account );
  procedure deposit( the: in out Account; amount: in PMoney );
  procedure withdraw( the: in out Account;
                     amount: in PMoney; get: out PMoney );
  function balance( the: in Account ) return Money;
  procedure set_overdraft( the: in out Account;
                           overdraft: in PMoney );

private
  type Account is tagged record
    balance_of : Money := 0.00;  -- Amount on deposit
    overdraft_allowed : Money := 0.00;  -- Overdraft limit
  end record;
end Class_account;
```
Implementation
How the package does it (Code)

```
with Ada.Float_Text_io, Ada.Text_io;
use Ada.Float_Text_io, Ada.Text_io;
package body Class_account is

   procedure statement( the:in Account ) is
   begin
      put("Mini statement: The amount on deposit is "$);
      put( the.balance_of, aft=>2, exp=>0 );
      new_line(2);
   end statement;

   procedure deposit( the:in out Account; amount:in PMoney ) is
   begin
      the.balance_of := the.balance_of + amount;
   end deposit;

   procedure withdraw( the:in out Account;
                        amount:in PMoney; get:out PMoney ) is
   begin
      if the.balance_of + the.overdraft_allowed >= amount then
         the.balance_of := the.balance_of - amount;
         get := amount;
      else
         get := 0.00;
      end if;
   end withdraw;

   function balance( the:in Account ) return Money is
   begin
      return the.balance_of;
   end balance;

   procedure set_overdraft( the:in out Account;
                           overdraft:in PMoney ) is
   begin
      the.overdraft_allowed := overdraft;
   end set_overdraft;

end Class_account;
```
procedure ex1 is
   mike :Account; -- Normal Account
   corinna :Account; -- Normal Account
   obtained:Money;
begin
   set_overdraft( mike, 100.00 ); -- $100 Overdraft
   statement( mike );
   put("Deposit $50.00 into Mike's account"); new_line;
   deposit( mike, 50.00 );
   statement( mike );
   put("Withdraw $80.00 from Mike's account"); new_line;
   withdraw( mike, 80.00, obtained );
   statement( mike );
   put("Deposit $500.00 into Corinna's account"); new_line;
   deposit( corinna, 500.00 );
   statement( corinna );
end main;
Ada compiler checks at compile time for some errors connected with dynamic storage allocation.

```ada
procedure Main is
    Max_Chs : constant := 7;
    type Height_Cm is range 0 .. 300;

    type Person is record
        Name   : String( 1 .. Max_Chs );  --Name as a String
        Height : Height_Cm := 0;          --Height in cm.
    end record;

    Mike   : aliased Person := Person'("Mike",156);
begin

    declare
        type P_Person is access all Person; --Access type
        P_Human: P_Person;
    begin
        P_Human:= Mike'access;          --OK

        declare
            Clive : aliased Person := Person'("Clive", 171);
        begin
            P_Human := clive'Access;        -- Compile time error
            P_Human := Clive'Unchecked_Access;
        end;

        Put( P_Human.Name ); New_Line;   --Clive no longer exists
        P_Human := Mike'access;          --Change to Mike
    end;

end Main;
```
package Class_account is

    type Account is private;
    subtype Money is Float;
    subtype PMoney is Float range 0.0 .. Float'Last;

    procedure deposit ( the: in out Account; amount: in PMoney );
    procedure withdraw( the: in out Account; amount: in PMoney;
                        get: out PMoney );
    function balance( the: in Account ) return Money;

private

    type Account is record
        balance_of : Money := 0.00; -- Amount in account
    end record;
end Class_account;

package body Class_account is

    procedure deposit ( the: in out Account; amount: in PMoney ) is
    begin
        the.balance_of := the.balance_of + amount;
    end deposit;

    procedure withdraw( the: in out Account; amount: in PMoney;
                        get: out PMoney ) is
    begin
        if the.balance_of >= amount then
            the.balance_of := the.balance_of - amount;
            get := amount;
        else
            get := 0.00;
        end if;
    end withdraw;

    function balance( the: in Account ) return Money is
    begin
        return the.balance_of;
    end balance;
end Class_account;
with Ada.Text_IO, Ada.Float_Text_IO, Class_Account;
use Ada.Text_IO, Ada.Float_Text_IO, Class_Account;
procedure Main is
  Mike    : Account;       -- Account
  Miranda : Account;       -- Account
  Obtained: Float;
begin

  Put("Deposit £50.00 into Mike's Account"); New_Line; --
  Deposit( Mike, 50.00 );

  Put("GetBalance -> ");
  Put( GetBalance( Mike ), Aft=>2, Exp=>0 ); New_Line; --

  Put("Withdraw £80.00 from Mike's Account"); New_Line; --
  Withdraw( Mike, 80.00, Obtained );
  Put("Obtained  -> ");
  Put( Obtained, Aft=>2, Exp=>0 ); New_Line; --

  Put("Deposit £500.00 into Miranda's Account"); New_Line;
  Deposit( Miranda, 500.00 );

end Main;
generic
    type Stack_Element is private;        --

package Class_Stack is
    type Stack is limited private;        --NO copying
    Stack_Error : exception;

    procedure Push( The:in out Stack; Item:in Stack_Element );
    procedure Pop(The:in out Stack; Item :out Stack_Element );
    procedure Reset( The:in out Stack );

private

    type Node;                        --Mutually recursive def
    type P_Node is access Node;      --Pointer to a Node
    pragma Controlled( P_Node );     --We do deallocation

    type Node is record
        Item   : Stack_Element;        --Node holds the data
        P_Next : P_Node;               --The stored item
    end record;

    type Stack is record
        P_Head : P_Node := null;       --First node in list
    end record;
end Class_Stack;
with Unchecked_Deallocation;
pragma Elaborate_All( Unchecked_Deallocation );
package body Class_Stack is

procedure Dispose is
  new Unchecked_Deallocation( Node, P_Node );

procedure Push( The: in out Stack; Item: in Stack_Element ) is
  Tmp : P_Node;                       -- Allocated node
begin
  Tmp := new Node'(Item=>Item, P_Next=>The.P_Head);
  The.P_Head := Tmp;
end Push;

procedure Pop( The: in out Stack; Item : out Stack_Element ) is
  Tmp : P_Node;                        -- Free node
begin
  if The.P_Head /= null then        -- if item then
    Tmp := The.P_Head;               -- isolate top node
    Item := The.P_Head.Item;         -- extract item stored
    The.P_Head := The.P_Head.P_Next; -- Relink
    Dispose( Tmp );                 -- return storage
  else
    raise Stack_Error;               -- Failure
  end if;
end Pop;

procedure Reset( The: in out Stack ) is
  Tmp : Stack_Element;
begin
  while The.P_Head /= null loop     -- Re-initialize stack
    Pop( The, Tmp );
  end loop;
end Reset;

end Class_Stack;

with Class_Stack;
package Class_Stack_Int is new Class_Stack(Integer);
Removing the need for run-time checks

```ada
with Ada.Text_io;
use Ada.Text_io;
procedure main is

  type Rooms_index is range 410 .. 414;
  subtype Rooms_range is Rooms_index;
  type Rooms_array is array (Rooms_range) of Natural;

  computers_in_room : Rooms_array;
  sum : Integer := 0;

begin
  computers_in_room := ( 10, 20, 30, 2, 3 );

  put("Computers in rooms 410 .. 414");
  for i in Rooms_range loop
    sum := sum + computers_in_room(i);
  end loop;
  put( Integer'Image(sum) ); new_line;

  sum := 0;
  put("Computers in rooms 410 .. 412");
  for i in 410 .. Rooms_index(412) loop
    sum := sum + computers_in_room(i);
  end loop;
  put( Integer'Image(sum) ); new_line;

end main;
```

Computers in rooms 410 .. 414 65
Computers in rooms 410 .. 412 60
Tasking in Ada

Specification (What)

```adaprogram
package Pack_Factorial is
  task type Task_Factorial is
    entry Start( F: in Positive );
    entry Finish( Result: out Positive );
  end Task_Factorial;
end Pack_Factorial;
```

Implementation (How)

```adaprogram
package body Pack_Factorial is
  task body Task_Factorial is
    Factorial : Positive;
    Answer    : Positive := 1;
    begin
      accept Start( F: in Positive ) do
        Factorial := F;
      end Start;
      for I in 2 .. Factorial loop
        Answer := Answer * I;
      end loop;
      accept Finish( Result: out Positive ) do
        Result := Answer;
      end Finish;
    end Task_Factorial;
end Pack_Factorial;
```
Creation

Thread : Task_Factorial;

Rendezvous

Thread.Start(45);  --Start factorial calculation

Thread.Finish(res);  --Rendezvous and return result
## Protected type

<table>
<thead>
<tr>
<th>Unit</th>
<th>Commentary</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure</td>
<td>A procedure will only execute when no other units are being executed. If necessary the procedure will wait until the currently executing unit(s) have finished.</td>
<td>Read and write.</td>
</tr>
<tr>
<td>function</td>
<td>A function may execute simultaneously with other executing functions. However, a function cannot execute if a procedure is currently executing.</td>
<td>Read only.</td>
</tr>
<tr>
<td>entry</td>
<td>Like a procedure but may also have a barrier condition associated with the entry. If the barrier condition is false the entry is queued until the barrier becomes true.</td>
<td>Read and write</td>
</tr>
</tbody>
</table>

### Barrier condition entry

The protected type uses one additional facility, that of a barrier entry. If the buffer becomes full, a mechanism is needed to prevent further data being added. The barrier:

```vhdl
erentry Put( Ch:in Character; No_More:in Boolean )
  when No_In_Queue < Queue_Size  is
```
Readers and writers
protected type PT_Buffer is  --Task type specification
  entry Put( Ch:in Character; No_More:in Boolean );
  entry Get( Ch:in out Character; No_More:out Boolean);
private
  Elements    : Queue_Array;              --Array of elements
  Head        : Queue_Index := 0;         --Index
  Tail        : Queue_Index := 0;         --Index
  No_In_Queue : Queue_No := 0;           --Number in queue
  Fin          : Boolean := False;        --Finish;
end PT_Buffer ;

protected body PT_Buffer is
  entry Put( Ch:in Character; No_More:in Boolean )
      when No_In_Queue < Queue_Size is
    begin  
      --Add item to queue
    end;

  entry Get(Ch:in out Character; No_More:out Boolean)
      when No_In_Queue > 0 or else Fin is
    begin
      --Remove item from Queue
    end;
end PT_Buffer ;
delay 2.5;

delay until Time_Of(2015,1,1,0.0); -- Until 1 Jan 2015

select
  accept option1 do
    ...
  end;
or
  accept option2 do
    ...
  end;
end select;

select
  accept Source1 do  --Successful rendezvous
    ...
  end;
or
  accept Source2 do  --Successful rendezvous
    ...
  end;
or
  delay 0.2;         --Time out
    Put("WARNING Watchdog failure");
    New_Line;
  exit;
end select;

select
  delay 5.0;        --What happens if it takes > 5.0 seconds
    Put("Did not complete");
  then abort
    Process_Data;  --Can be aborted
  end select
Machine specific issues in Ada

- Handled by `pragma` directive to compiler
  
  ```ada
  pragma Storage_Size( 2000 )
  pragma Priority( 16 )
  ```
Java background

1995: Released to the world
2011: 1.7, Java 7
2014: 1.8, Java 8 // Lambda functions

Paradigm OO
Typeing static, strong
Influenced by Objective-C, C++
Influenced C#
Java Major features

◆ Object-oriented

◆ Emphasises safety (Strong typing)
  ```java
  int age;
  age = 2.3; // Error
  ```

◆ try { } catch ()

◆ Threads
  Part of language, synchronized, task priority,

◆ Standard library (Part of language)
  Large GUI, collection classes, Database access,

Java Application area

◆ Applications, Applets, Client Server computing
◆ General purpose
Java example

```java
import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

class Cat {
    public static void main( String args[] )
    {
        for ( int i=0; i<args.length; i++ )
        {
            try
            {
                FileInputStream in = new FileInputStream( args[i] );
                try
                {
                    int c;
                    while ( ( c = in.read() ) != -1 )
                    {
                        System.out.print( (char) c );
                    }
                    in.close();
                }
                catch ( IOException name )
                {
                    System.err.println( "Error reading from : " + args[i] );
                }
                catch ( FileNotFoundException e )
                {
                    System.err.println( "Can not open : " + args[i] );
                }
            }
            catch ( IOException name )
            {
                System.err.println( "Error reading from : " + args[i] );
            }
        }
    }
}
```
C++ Background

Bjarne Stroustrup
1979 C with Classes
1983 C++ First used
1998 ANSI-ISO standard

Paradigm        Multi-paradigm, generic programming,
                 OO, procedural
Typeing         static, weak unsafe
Influenced by   C, Simula, Ada83, CLU
Influenced      Ada 95, C#, Java
C++ Major features

- Not a superset of C
  There are some differences e.g. `new` is not a keyword in C

- `{ }` used as begin end.

- Emphasises 'A better C' code quality as good as assembler at times, however lines of source code not always an indicator of code size.

```cpp
int a;
Account mike; mike.deposit(100.00);
if (Condition) Statement;
```

- Strong typing
  ```cpp
  int i = 1.2;  // Warning conversion
  float f = 1;  // 1 promoted to 1.0
  // Is this good or bad
  ```

- Can easily access machine level features
  Explicit access to pointers
  Operators & address, * indirection
  ```cpp
  while ( *p ) *copy++ = *p++;
  while ( *copy++ = *p++ );
  ```

- Standard library (Part of language)
  Small, No GUI

- Separate compilation, through individual files, linking type safe

C++ Application area

- Operating systems,
  Applications where code quality matters (size & speed: )
C++ example: cat

```cpp
#include <iostream>             // Normal I/O
#include <iomanip>              // Allows iomanipulators
#include <fstream>              // Fstream

int main( const int argc, const char * const argv[] )
{
    for ( int i=1; i<argc; i++ )            // For each argument
    {
        std::fstream s( argv[i], std::ios::in ); // Open as file
        if ( !s.fail() )                      // Ok
        {
            char ch;
            s >> std::resetiosflags( std::ios::skipws );

            while( s >> ch, !s.eof() )         // Read ch from file
            {
                std::cout << ch;
            }
            s.close();                        // Close stream
        }
        else
        {
            std::cerr << "Can not open " << argv[i] << "\n";
        }
    }
    return 0;                            // Command Ok
}
```
C++

Explicitly by programmer

```c++
int main()
{
  int *p_int;       // p_int holds a pointer to an int
  p_int = new int[10]; // return pointer to 10 ints on heap
  p_int[0] = 00;    // Use as an array
  p_int[9] = 99;    // Ooops no bound checking
  delete [] p_ch;  // Return the storage
}
```

◆ Problems
   Possibility for error is high
   No array bound checking
Using a standard library class

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

void print( string mes, vector <int> &aList );

int main()
{
    vector <int> list;
    list.push_back(10 );  // Add at end
    list.push_back(20 );  // Add at end
    list.push_back(30 );  // Add at end
    list.push_back(40 );  // Add at end

    vector <int> copy = list;

    list.erase( list.begin() + 1 );  // 2nd element

    print( "original", list );
    print( "copy",      copy );
}

void print( string mes, vector <int> &aList )
{
    cout << mes << ": ";
    for ( vector<int>::iterator it = aList.begin(); it != aList.end(); it++ )
    {
        cout << *it << " ";  // Element
    }
    cout << endl;  // Newline
}

original: 10 30 40
copy: 10 20 30 40
```

An instance of the class `std::vector <int>` will have a destructor which will clean up (in this case release any dynamically allocated storage) just before the instance of the conventionally allocated storage is released.
Access to real memory

In Ada

Print the time from DOS

```ada

procedure Main is
   Time_High_Address : constant Address := To_Address( 16#046C# );
   Time_Low_Address  : constant Address := To_Address( 16#046E# );
   type Seconds_T    is range 0 .. 1_000_000_000;  -- up to 65k * 5
   type Time         is range 0 .. 65365;          -- Unsigned
   for Time'Size use 16;                          -- in 2 bytes
   Time_Low    : Time;
   for Time_High'Address use Time_High_Address;
   Time_High  : Time;
   for Time_Low'Address use Time_Low_Address;
   Seconds    : Seconds_T;
begin
   Put("Time is ");
   Put( Time'Image(Time_High) ); Put(" : "); -- Hour
   Seconds := (Seconds_T(Time_Low) * 5) / 91;
   Put(Seconds_T'Image(Seconds/60)) Put(" : "); -- Mins
   Put(Seconds_T'Image(Seconds rem 60));       -- Seconds
   New_Line;
end Main;
```
In C++

Access a device register for a simple output port

```cpp
struct Acia {
    char    status;        // Status information
    char    dummy1;        // 1 byte pad
    char    data;          // Data register
};

volatile Acia * base_vdu = reinterpret_cast<Acia*>(0x0C0080);
const int ACIA_RM  =  0x01;
const int ACIA_TM  =  0x02;
//
//   Wait for device to become free
//   Read a character from the terminal
//
char getchar()
{
    while( (base_vdu->status & ACIA_RM) == 0 );
    return base_vdu->data;
}
```
### Class definition

<table>
<thead>
<tr>
<th>Specification part</th>
<th>Implementation part</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>class Account {</code></td>
<td><code>Account::Account()</code></td>
</tr>
<tr>
<td><code>public:</code></td>
<td><code>{</code></td>
</tr>
<tr>
<td><code>Account();</code></td>
<td><code>the_balance = the_min_balance = 0.00;</code></td>
</tr>
<tr>
<td><code>float get_balance();</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><code>float withdraw(float );</code></td>
<td><code>float Account::get_balance()</code></td>
</tr>
<tr>
<td><code>void deposit(float );</code></td>
<td><code>{</code></td>
</tr>
<tr>
<td><code>void set_min_balance(float );</code></td>
<td><code>return the_balance;</code></td>
</tr>
<tr>
<td><code>private:</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><code>float the_balance;</code></td>
<td><code>float Account::withdraw(float money)</code></td>
</tr>
<tr>
<td><code>float the_min_balance;</code></td>
<td><code>{</code></td>
</tr>
<tr>
<td><code>};</code></td>
<td><code>if (the_balance-money &gt;= the_min_balance)</code></td>
</tr>
<tr>
<td></td>
<td><code>{</code></td>
</tr>
<tr>
<td></td>
<td><code>the_balance = the_balance - money;</code></td>
</tr>
<tr>
<td></td>
<td><code>return money;</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
<tr>
<td></td>
<td><code>else {</code></td>
</tr>
<tr>
<td></td>
<td><code>return 0.00;</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
<tr>
<td></td>
<td><code>void Account::deposit(float money)</code></td>
</tr>
<tr>
<td></td>
<td><code>{</code></td>
</tr>
<tr>
<td></td>
<td><code>the_balance = the_balance + money;</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
<tr>
<td></td>
<td><code>void Account::set_min_balance(float money)</code></td>
</tr>
<tr>
<td></td>
<td><code>{</code></td>
</tr>
<tr>
<td></td>
<td><code>the_min_balance = money;</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
<tr>
<td><code>int main()</code></td>
<td><code>int main()</code></td>
</tr>
<tr>
<td><code>{</code></td>
<td><code>{</code></td>
</tr>
<tr>
<td></td>
<td><code>Account mike, cori;</code></td>
</tr>
<tr>
<td></td>
<td><code>float obtained;</code></td>
</tr>
<tr>
<td></td>
<td><code>mike.deposit(100.00);</code></td>
</tr>
<tr>
<td></td>
<td><code>cori.deposit(120.00);</code></td>
</tr>
<tr>
<td></td>
<td><code>std::cout&lt;&lt;&quot;Mike's account = &quot; &lt;&lt; mike.get_balance()&lt;&lt;&quot;\n&quot;;</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>
Java

class Main()
{
    public static void main( String args[] )
    {
        Account mike = new Account();
    }
}

The Java system will de-allocate the storage for the instance of the class Account when it can be no longer reached.
Deep copy

\[ a \rightarrow \text{after: } b = a \]

- \( a \) & \( b \) do not share storage
- Time to copy may be large

Shallow copy

\[ a \rightarrow \text{after: } b = a \]

- \( a \) & \( b \) share storage
- Very fast to implement
Representation of Objects

**In Java**

Mike

Does this matter / What are the consequences

Java

Easy to share objects between different classes

**In C++**

Mike

100.00

You make a new copy when you share
If you do want this to happen have to use a pointer to the object.
Memory leak now a possability (when copy pointer)
Who has responsibility for releasing the storage

Java

If you want a new copy of an object have to use clone

```java
Board b = new Board();
Board backup = b;    // ooops
Board backup = b.clone();    // OK
```

If the class implements the interface `cloneable`, you get a bitwise copy of the object. This bitwise copy (shallow copy) is implemented in the class `Object`. If you want more than this you have to provide a method `clone` to do the action.
Using an object with pointers C++

```cpp
class SimpleIntVector {
public:
    SimpleIntVector();              //Allocate storage
    ~SimpleIntVector();             //De-allocate storage
    void add( int );                //Add new number at end
    int& operator[]( const int );   //indexing
private:
    int* storage;                   // Storage for array
};

{
    SimpleIntVector numbers;
    numbers.add(1);
    numbers.add(2);

    std::cout << numbers[0] << " " << numbers[1];
}
```
```java
{ 
    SimpleIntVector numbers;
    SimpleIntVector otherNo;
    numbers.add(1);
    numbers.add(2);
    otherNo = numbers;
}
```

Diagram:
- `numbers` contains elements 1 and 2.
- `otherNo` is pointing to the same memory as `numbers`.

**Memory leak**
Solution

Overload the assignment operator with a new meaning

class SimpleIntVector {
public:
    SimpleIntVector(); // Allocate storage
    ~SimpleIntVector(); // De-allocate storage
    void add( int ); // Add new number at end
    int& operator[] ( const int ); // Indexing
    SimpleIntVector& operator = ( SimpleIntVector ); // Assignment
private:
    int* storage; // Storage for array
};

{
    SimpleIntVector numbers;
    SimpleIntVector otherNo;
    numbers.add(1);
    numbers.add(2);
}

numbers

otherNo
otherNo = numbers;
}

Now assignment operator
Does the assignment, increments the ref count for numbers
Decrement the ref count for otherNo, as it is now 0
releases the storage
Hidden assignments

◆ When you pass an instance of a SimpleIntVector to a function/method

So need to overload the copy constructor

```cpp
class SimpleIntVector {
public:
    SimpleIntVector();
    ~SimpleIntVector();
    void add(int);
    int& operator[](const int);
    SimpleIntVector& operator=(SimpleIntVector);
    SimpleIntVector(SimpleIntVector&); // copy construc.
private:
    int* storage; // Storage for array
};
```

Copy constructor (Which you provide)
Does the copy
Increments the ref count of the target only
Concurency / Tasking/ Threads

The ability to run a separate task/ thread in a program.

Why used

- Most modern GUI based programs have several threads. Background calculation/ display whilst data is being entered.
- A server will run many threads, each thread will be responsible for processing an individual user request.

Issues in concurency

- Is part of language
- How access to shared resources is handled
  Serialization of access
- Machine specific features
  Thread priority, resources used

<table>
<thead>
<tr>
<th>Language</th>
<th>Tasks part of language</th>
<th>Use Tasking from O/S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>BCPL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>C++</td>
<td>✓ (C++ 11)</td>
<td>✓</td>
</tr>
<tr>
<td>Java</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

© M A Smith 2012 May 13, 2015 Page 82
Tasking in Ada 95

Specification (What)

```ada
package Pack_Factorial is
  task type Task_Factorial is
    entry Start( F: in Positive );
    entry Finish( Result: out Positive );
  end Task_Factorial;
end Pack_Factorial;
```

Implementation (How)

```ada
package body Pack_Factorial is
  task body Task_Factorial is
    Factorial : Positive;
    Answer    : Positive := 1;
  begin
    accept Start( F: in Positive ) do
      Factorial := F;
    end Start;
    for I in 2 .. Factorial loop
      Answer := Answer * I;
    end loop;
    accept Finish( Result: out Positive ) do
      Result := Answer;
    end Finish;
  end Task_Factorial;
end Pack_Factorial;
```
Creation

Thread : Task_Factorial;

Rendezvous

Thread.Start(45); --Start factorial calculation

Thread.Finish(res); --Rendezvous and return result
Protected type

<table>
<thead>
<tr>
<th>Unit</th>
<th>Commentary</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure</td>
<td>A procedure will only execute when no other units are being executed. If necessary the procedure will wait until the currently executing unit(s) have finished.</td>
<td>Read and write.</td>
</tr>
<tr>
<td>function</td>
<td>A function may execute simultaneously with other executing functions. However, a function cannot execute if a procedure is currently executing.</td>
<td>Read only.</td>
</tr>
<tr>
<td>entry</td>
<td>Like a procedure but may also have a barrier condition associated with the entry. If the barrier condition is false the entry is queued until the barrier becomes true.</td>
<td>Read and write</td>
</tr>
</tbody>
</table>

Barrier condition entry

The protected type uses one additional facility, that of a barrier entry. If the buffer becomes full, a mechanism is needed to prevent further data being added. The barrier:

```ada
entry Put( Ch: in Character; No_More: in Boolean )
when No_In_Queue < Queue_Size    is
```
Readers and writers

Diagram shows:
- Disk to Reader task: put
- Reader task to Buffer: put
- Buffer to Writer task: get
- Writer task to Disk: put
- Disk to Reader task: put
- Reader task to Buffer: put
- Buffer to Writer task: get
- Writer task to Disk: put
protected type PT_Buffer is  --Task type specification
    entry Put( Ch:in Character; No_More:in Boolean );
    entry Get( Ch:in out Character; No_More:out Boolean );
private
    Elements : Queue_Array;  --Array of elements
    Head      : Queue_Index := 0;  --Index
    Tail      : Queue_Index := 0;  --Index
    No_In_Queue : Queue_No := 0;  --Number in queue
    Fin       : Boolean := False;  --Finish;
end PT_Buffer ;

protected body PT_Buffer is
    entry Put( Ch:in Character; No_More:in Boolean )
        when No_In_Queue < Queue_Size is
    begin
        --Add item to queue
    end;

    entry Get( Ch:in out Character; No_More:out Boolean)
        when No_In_Queue > 0 or else Fin is
    begin
        --Remove item from Queue
    end;
end PT_Buffer ;
```plaintext
delay 2.5;

delay until Time_Of(2015,1,1,0.0);   -- Until 1 Jan 2015

select                      -- Choice of accepts
  accept option1 do
    ...
  end;
or
  accept option2 do
    ...
  end;
end select;

select
  accept Source1 do    --Successful rendezvous
    ...
  end;
or
  accept Source2 do    --Successful rendezvous
    ...
  end;
or
  delay 0.2;           --Time out
    Put("WARNING Watchdog failure");
    New_Line;
    exit;
end select;

select
  delay 5.0;           --What happens if it takes > 5.0 seconds
    Put("Did not complete");
then abort
    Process_Data;    --Can be aborted
end select
```
Machine specific issues in Ada

- Handled by **pragma** directive to compiler

  `pragma Storage_Size( 2000 )`
  `pragma Priority( 16 )`
Concurency in Java

```java
class Factorial extends Thread {
    private int theNumberOfTerms; //Input data
    private long theAnswer;       //Output results

    public Factorial( final int terms ) //Construct
    {
        theNumberOfTerms = terms;
    }

    public void run()                      //Execution
    {
    }

    public long result()                  //Deliver results
    {
        return theAnswer;
    }
}

public static void main()
{
    try
    {
        Factorial thread = new Factorial(45);  //Construct thread
        //Start calls run in newly created thread
        thread.start();

        //Other work
        thread.join();                   //Wait till finishes

        long answer = thread.result();   //Retrieve answer
    }
    catch ( InterruptedException exc )
    {
    }
}
```
In Java

![Diagram showing disk, Reader thread, put, get, Writer thread, and disk with shared data]

Methods of Thread

<table>
<thead>
<tr>
<th>Method</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread()</td>
<td>Creates a new thread with name name.</td>
</tr>
<tr>
<td>Thread( name )</td>
<td>Creates a new thread that will run the function object fo that implements the interface Runnable.</td>
</tr>
<tr>
<td>Thread(fo)</td>
<td></td>
</tr>
<tr>
<td>Thread(fo,name)</td>
<td></td>
</tr>
<tr>
<td>activeCount()</td>
<td>Returns the number of active threads in this thread group.</td>
</tr>
<tr>
<td>destroy()</td>
<td>Destroys the thread. There is no cleanup.</td>
</tr>
<tr>
<td>getName()</td>
<td>Returns the name of the thread.</td>
</tr>
<tr>
<td>isAlive()</td>
<td>Returns true if the thread is alive.</td>
</tr>
<tr>
<td>join()</td>
<td>Waits for the thread to die.</td>
</tr>
<tr>
<td>join( delay )</td>
<td>Waits at most delay milliseconds for the thread to die.</td>
</tr>
<tr>
<td>run()</td>
<td>Runs the thread’s function object that implements the interface Runnable.</td>
</tr>
<tr>
<td>sleep( delay )</td>
<td>Causes the thread to sleep for delay milliseconds.</td>
</tr>
<tr>
<td>start()</td>
<td>Causes the thread to start, the run method in the thread is called.</td>
</tr>
<tr>
<td>yield()</td>
<td>After pausing the thread allow other threads to continue.</td>
</tr>
<tr>
<td>notifyAll()</td>
<td>Wakes up all threads that are waiting on this object. A thread enters the wait state when it calls one of the wait methods.</td>
</tr>
<tr>
<td>notify()</td>
<td>Wakes up a single thread that is waiting on this object.</td>
</tr>
<tr>
<td>wait( delay )</td>
<td>Waits until either of the following two conditions occur:</td>
</tr>
<tr>
<td></td>
<td>● One of the methods notify or notifyall is called from another thread on this object.</td>
</tr>
<tr>
<td></td>
<td>● The time delay in milliseconds has passed.</td>
</tr>
<tr>
<td>wait()</td>
<td>Waits until one of the methods notify or notifyall is called from another thread on this object.</td>
</tr>
</tbody>
</table>
Methods of Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>notify()</td>
<td>Wakes up a single thread that is waiting on this object.</td>
</tr>
<tr>
<td>notifyAll()</td>
<td>Wakes up all threads that are waiting on this object.</td>
</tr>
<tr>
<td></td>
<td>A thread enters the wait state when it calls one of the wait methods.</td>
</tr>
<tr>
<td>wait(delay)</td>
<td>Waits until either of the following two conditions occur:</td>
</tr>
<tr>
<td></td>
<td>● One of the methods notify or notifyall is called from another thread on this object.</td>
</tr>
<tr>
<td></td>
<td>● The time delay in milliseconds has passed.</td>
</tr>
<tr>
<td>wait()</td>
<td>Waits until one of the methods notify or notifyall is called from another thread on this object.</td>
</tr>
</tbody>
</table>
class Buffer
{
    private static final int QUEUESIZE = 5;       //Size
    private static final int LAST = QUEUESIZE-1;  //Last index

    private Object theQueue[] = new Object[ QUEUESIZE ];
    private int theNoInQueue = 0;                  //# items
    private int theHead = 0;                       //Head of
    private int theTail = 0;                       //Tail of

    public synchronized Object get()
    {
        while ( theNoInQueue <= 0 )
        {
            try {wait();} catch (InterruptedException e) {} //Suspend
        }
        Object res = theQueue[theHead];                   //Remove
        theHead = theHead == LAST ? 0 : theHead+1;         //Cycle
        theNoInQueue--;                                   //1 less
        notify();                                         //Wake up
        return res;
    }

    public synchronized void put( final Object value )
    {
        while ( theNoInQueue >= QUEUESIZE )
        {
            try {wait();} catch (InterruptedException e) {} //Suspend
        }
        theQueue[theTail] = value;                        //Add
        theTail = theTail == LAST ? 0 : theTail+1;        //Cycle
        theNoInQueue++;                                   //1 more
        notify();                                         //Wake up
    }
}

◆ Key points

◆ **synchronized**
Serialised access to **synchronized** methods

◆ A class **extends** (inherits) from the class object if it does not inherit from another class.
Which is better for tasking/ threads

- **Ada 95**
  - Part of language
  - Idea of barrier conditions in protected type
  - Selective rendezvous
  - Time-out on rendezvous

- **Java**
  - Part of language
  - Built on OO ideas
Reducing complexity

- Hiding of non-relevant information
- Grouping of related data together
Visibility (Reducing complexity)

- What is the visibility of names in a programming language.

- By controlling visibility we simply the complexity involved in program construction.

- **Functional decomposition**
  Split program into functions/ procedures, which implement a part of the program. The implementation of the function/ procedure is hidden from the caller. Functions/ procedures may be nested to hide inner functions.

  **Concepts**
  Local variables in functions, static (C, C++, Java) / own (Algol 60) variables to retain state between function calls.

- **OO programming**
  Split program into objects, messages sent to objects to implement the functionality of program. Implementation of the object is hidden from the caller.

  **Concepts**
  Local variables, in methods, objects retain state between method calls, using instance variables.
C++ (Nov 1997 Standard, 2011 C++11)

- Block structured language (2 levels)
  Functions may not be nested.

- Separate compilation of Functions

- Class
  (Can be) Split into specification and implementation

  public: Visible outside the class
  protected: Visible to inheriting class
  private: No visibility outside

Classes may be nested

```
class Account {
  public:
      Account();
      float account_balance();
      float withdraw( float );
      void deposit( float );
  private:
      float the_balance;
};

Account::Account()
{
    the_balance = 0.0;
}

float Account::account_balance()
{
    return the_balance;
}

float Account::withdraw(float money)
{
    if ( money <= the_balance )
    {
        the_balance = the_balance-money;
        return money;
    } else {
        return 0;
    }
}

void Account::deposit( float money )
{
    the_balance = the_balance + money;
}
```
Namespace
Allows the partitioning of the names into several spaces thus preventing pollution of the global name space.

```cpp
namespace Red //The red world
{
    std::string colour() { return "red"; }
    std::string fruit() { return "strawberries"; }
}

namespace Blue //The blue world
{
    std::string colour() { return "blue"; }
    std::string fruit() { return "blueberries"; }
}

int main()
{
    using namespace Red;
    std::cout << "My favourite colour is " << colour()
              << " and fruit is " << fruit() << "\n";
    return 0;
}
```

Data structures - yes
Arrays: no subscript checking, container classes
Java (1995)

- Block structured language (2 levels)
  Functions/methods may not be nested but can be called recursively.

- Separate compilation of Functions, classes

- Classes can be nested (Inner class)

```java
class Account {
    private double theBalance = 0.00; //Balance of account

    public Account( double balance ) {
        theBalance = balance; //Opening balance
    }

    public double getBalance() {
        return theBalance;
    }

    public double withdraw( final double money ) {
        if ( theBalance - money >= 0.00 ) {
            theBalance = theBalance - money;
            return money;
        } else {
            return 0.00;
        }
    }

    public void deposit( final double money ) {
        theBalance = theBalance + money;
    }
}
```
Package
Used to contain like classes. The import statement make visible a class or all classes in a package. Special rules about visibility of items between classes in the same package.

```java
import java.util.jar.Manifest
```

In the package `java` is the package `util` that contains the package `jar` in which is the class `manifest`.

```java
import java.util.jar.*
```

Make visible all classes in the package `java.util.jar`.

Data structures - yes
Arrays, container classes
Macro processor in C++

```
#include "filename" /* Taken from current working directory */
#include <filename> /* Taken from system directories */

#define MAX 120

#define larger(x,y)  ( (x)>(y) ? (x) : (y) )
```

Conditional compilation

```
#include <iostream>
#include "local.h"

int main()
{
    #ifdef DEBUG
        std::cerr << "Entering function main" << "\n";
    #endif
        std::cout << "Hello world" << "\n";
    return 0;
}
```
C++ Code

```cpp
#include <iostream>

int main()
{
    int countdown = 10;
    while ( countdown > 0 )
    {
        cout << countdown << endl;
        if ( countdown == 3 )
        {
            cout << "Ignition" << endl;
        }
        countdown = countdown - 1;
    }
    cout << "Blast off" << endl;
}
```
Over use of macro processors in C++

```c++
#define IF if(
#define THEN ){
#define ELSE } else {
#define ELIF } else if {
#define FI ;}
#define BEGIN {
#define END }
#define WHILE while(
#define DO ){
#define OD ;
#define REP do{
#define PER }while(
#define DONE );

#include <iostream>
int main()
BEGIN
  int countdown;
  countdown = 10;
  WHILE countdown > 0 DO
    cout << countdown << "\n";
    IF countdown == 3 THEN
      cout << "Ignition" << "\n";
    FI
    countdown = countdown - 1;
  OD
  cout << "Blast off" << "\n";
END;

#include <iostream>
int main()
{
  int countdown;
  countdown = 10;
  while( countdown > 0 ){
    cout << countdown << "\n";
    if( countdown == 3 ){
      cout << "Ignition" << "\n";
    };
    countdown = countdown - 1;
  };
  cout << "Blast off" << "\n";
};
```
ML/1 A general purpose Macro processor (1967)

```
MCSKIP MT,<WITH[ ]WITH>
MCINS [WITH= =WITH]
MCDEF BEGIN END AS <[
    [=A1=]
]> MCDEF WHILE DO OD AS <[while ( [=A1=] ) {
    [=A2=]
}> MCDEF IF THEN FI AS <[ if ( [=A1=] ) {
    [=A2=]
}>]

#include <iostream>
int main()
{
    int countdown;
    countdown = 10;
    while ( countdown > 0 )
    {
        cout << countdown << "\n";
        if ( countdown == 3 )
        {
            cout << "Ignition" << "\n";
        }
        countdown = countdown - 1;
    }
    cout << "Blast off" << "\n";
};
```
MCSKIP MT,<WITH[ ]WITH>
MCINS [WITH= =WITH]
MCDEF BEGIN END AS <[
{   
    [=A1=]
}]>
MCDEF WHILE DO OD AS <[while ( [=A1=] ) { 
    [=A2=]
}]>
MCDEF IF THEN FI AS <[ if ( [=A1=] ) { 
    [=A2=]
}]>
MCDEF cout N1 OPT <WITH< N1 OR ; ALL AS <[MCSET T3 = 2 [=L1=] <[cout]>>=[A [=T3=]=];MCGO L2 IF T3 EN T1 MCSET T3=T3+1 MCGO L1
[=L2=]]>

#include <iostream>
int main()
{
    int countdown;
    countdown = 10;
    while ( countdown > 0 )
    {
        cout << countdown;
        cout << "\n";
        if ( countdown == 3 )
        {
            cout << "Ignition";
            cout << "\n";
        }
        cout << "Blast off";
        cout << "\n";
        countdown = countdown - 1;
    }
    cout << "Blast off";
    cout << "\n";
};
Compiling and running an ada program

◆ Installing at home
  Download from:

  http://libre.adacore.com/libre/download/

  choose free software or academic development

  Select GNAT_GPL (~101MB)

  Download and install

◆ Running ada at the University / Home

  An example ada program is at:

  www.cem.brighton.ac.uk/staff/mas/ada95/programs/x70_cc4.ada

  gnatchop x70_cc4.ada  // File of ada program units
                  // Split into individual files

  gnatmake copy.adb    // Main procedure
                  // Run the copy program

  copy.exe  fromFile toFile
Transactional memory

- Sandy Bridge 32nm (2011-2013) [tock]

- Ivy Bridge 22nm (Q2 2012 - Q1 2014) [tick]
  2-15 cores, [15 cores $6.8k - $4.2k]
  22nm (Die shrink of Sandy bridge)

- Haswell 22nm (Q3 2013 - Q3 2014)
  2-18 cores [18 cores $expensive?] [tock]
  Introduced hardware support for Transactional memory

- Broadwell 14nm (Q1 2015 - ) [tick]
  2-4 cores

- Skylark 14nm (2015?) [tock]

  tick - A lithography shrink, tock - A new architecture

- What is Transactional memory
  Allows implementation of lock free data structures
Conventional data structure require the use of locks when the data structure can be accessed by more than one thread/process.

Problems using locks

- **Deadlock**
  A process (a) requires a resource that they can never obtain, as another process (b) holds the resource and requires a resource held by the process (a).

- **Convoying**
  A process (a) can not immediately proceed as it requires a resource that is held by another process. The process (a) relinquishes the CPU and causes a context switch.

When several equal priority processes compete for the same resource, the repeated context switches will degrade overall performance.

- **Priority inversion**
  A low priority process holds a resource that a high priority process needs.
Transactional memory

- Transaction memory
  Stores changes / read locations, if no conflict with what other cores have in their transactional memory data can be committed to ram memory.

  If not then has to be discarded and the transaction started again.

- Transaction memory stores all data read / to be written so has to be fully associative.
Transactional memory (Primitives)

- Overview
  Make tentative changes to 'shared memory' then at the end of the transaction will either

  - Comit
    Make changes permanent
  - Abort
    Discard changes (will need to repeat transaction)

- Primatives

  - Load transactional - LT
    Reads the value of a shared value into a private register

  - Load transactional exclusive - LTX
    Reads the value of a shared value into a private register, hinting that it is likely to be updated

  - Store transactional - ST
    Tentatively writes a value from a private register to a shared memory location.

    This value does not become visible until the transaction is successfully committed
Commit

Read set - is locations read by LT
Write set - is locations written by LTX & ST
Data set = Read set + Write set

Commit succeeds
If no other processes have updated locations in processes read set & no other transactions have read a location in its write set

Commit fails
All changes to its write set are discarded

Abort
Discards all changes to write set

Validate
true - All OK so far
false - Current transaction has aborted
Transactional memory (Example)

- Use LT or LTX to read shared locations
- Use Validate to check that the values read are consistent
- Use ST to modify locations
- Use Commit to make permanent
  If fail (Validate or Commit) repeat from 1st step
**Shared buffer**

```java
public synchronized Object get()
{
    while ( theNoInQueue <= 0 )
    {
        try {wait();} catch (InterruptedException e) {} // Suspend
    }
    Object res = theQueue[theHead]; // Remove
    theHead = theHead == LAST ? 0 : theHead+1; // Cycle
    theNoInQueue--; // 1 less
    notify(); // Wake up
    return res;
}

public synchronized void put( final Object value )
{
    while ( theNoInQueue >= QUEUESIZE )
    {
        try {wait();} catch (InterruptedException e) {} // Suspend
    }
    theQueue[theTail] = value; // Add
    theTail = theTail == LAST ? 0 : theTail+1; // Cycle
    theNoInQueue++; // 1 more
    notify(); // Wake up
}
```
public Object get()
{
    while ( true )
    {
        if ( LT(theNoInQueue) > 0 ) // not empty
        {
            Object res = LT(theQueue[LT(theHead)]); // Remove
            if (LTX(theHead) == LAST )
                ST( theHead, 0 )
            else
                ST( theHead, LTX(theHead)+1 );
            ST( theNoInQueue, LTX(theNoInQueue) - 1 ); // 1 less Q
            if (Commit() ) return res;
            for ( int i=0; i<WAIT; i++ ); // wait backoff
        } else {
            Abort();
            // wait sleep
        }
    }
}

public void put( final Object value )
{
    while (true)
    {
        if ( LT(theNoInQueue) < QUEUESIZE ) // not empty
        {
            ST(theQueue[LT(theTail)], value); // Add
            if (LTX(theTail) == LAST )
                ST( theTail, 0 )
            else
                ST( theTail, LTX(theTail)+1 );
            ST( theNoInQueue, LTX(theNoInQueue) + 1 ); // 1 more Q
            if (Commit() ) return res;
            for ( int i=0; i<WAIT; i++ ); // wait backoff
        } else {
            Abort();
            // wait sleep
        }
    }
}
When to use Transactional memory

- Short code sequences that would need to hold a lock to prevent other threads (usually running on other cores) accessing shared values.
SQL Injection (Simple example)

- In a computer system that uses a web page to validate a user's password the SQL statement used on the server might be:

  ```sql
  select * from passwords
  where USER = 'user' and PASS = 'password'
  ```

  Note **password** would have been passed through a 1 way algorithm. Hence the stored password in the database table is not the plain text version of the password.

  This SQL statement would be dynamically generated from the user's input. **user** is replaced by the input from the web form and so is **password** (after being passed through a 1 way function).

  Then a test is made to see if the resultset returned contains the user's credentials/ information, if it does then the user's credentials are accepted.

  Imagine though if the user supplied as the username **mas' or USER = 'x** and the password as **pass -> encrypted to dRfw**

  Then the SQL executed would be:

  ```sql
  select * from users
  where USER = 'mas' or USER = 'x' and PASS = 'dRfw'
  ```

  Which would return mas users record, which supposedly had been validated using mas's password.
How to defeat this
Check the users password does not contain a ' or SQL comment characters.