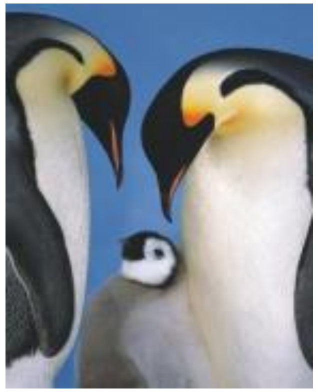
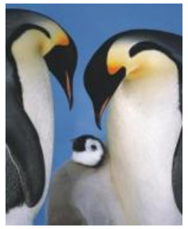
Standard I/O Library

Alain NINANE – RSSI UCL

23 Février 2016

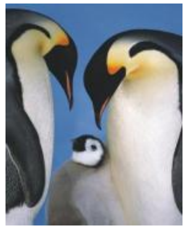


The Standard I/O Library



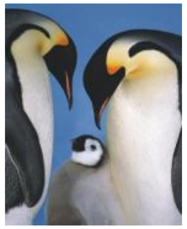
- Uniform interface for performing I/O
  - efficient user-level programming interface
  - work with “streams”
  - provides user level buffering
    - **syscalls are expensive** …
      - ***Just ask web search engine*** ☺
- Simplicity
  - only one include file
    - #include <stdio.h>

Streams



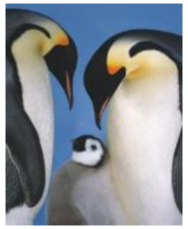
- Similar to file descriptors
- Designates devices (keyboard, files, ...)
- Standard predefined streams
  - .....
- Dynamically allocated streams
  - .....

Streams



- Predefined
  - stdin
  - stdout
  - stderr
- Dynamically allocated
  - from fopen -> man fopen
  - FILE *ioptr = fopen(“myFile”,”r”);
  - returns NULL in case of failure

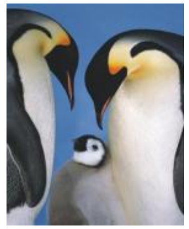
File's vs file descriptors



- File contains ...
  - a file descriptor (fileno)
  - user level buffers
  - See /usr/include/stdio.h

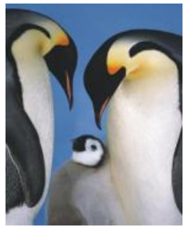
- fd -> ioptr
  - Use fdopen()

User level buffering



- Output is non - synchronous
- Flushing can be used
  - fflush
  - use setbuf
    - setbuf(ioptr,NULL)
- Stream destruction
  - fclose(ioptr)

Basic input functions

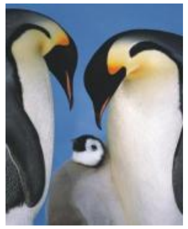


- #include <stdio.h>

```
int fgetc(FILE *stream);
char *fgets(char *s, int size, FILE *stream);
int getc(FILE *stream);
int getchar(void);
char *gets(char *s);
int ungetc(int c, FILE *stream);
```

- BEWARE: Buffer overflows !!!!!!!
  - allways check input length
  - avoid gets

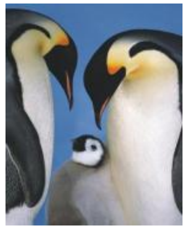
Basic output functions



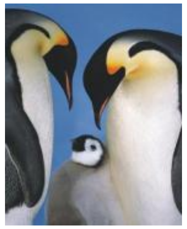
- #include <stdio.h>

```
int fputc(int c, FILE *stream);
int fputs(const char *s, FILE *stream);
int putc(int c, FILE *stream);
int putchar(int c);
int puts(const char *s);
```

Output Formatting Functions



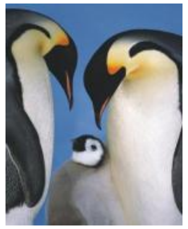
- printf()
  - printf(char* format, arg1, arg2, ...);
    - printf("\tval1 = %3d - val2 = 0x%04x\n,val1,val2);
- fprintf()
  - fprintf(FILE *ioptr, char *format, arg1, arg2...);
    - fprintf(ioptr,"Hello %s ! How are you ?\n",name);



Formatted Input Functions

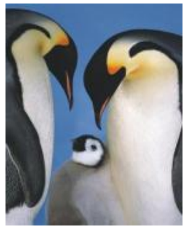
- scanf
  - scanf(char *format, ptr1, ptr2, ...);
    - int val;
    - char month[20];
    - int nv = scanf("%d %s",&int1, month);
- fscanf
  - fscanf(FILE *ioptr, char *format, ptr1, ptr2, ...);

String equivalents ...



- To better control formatting ... use
  - sprintf(char *outbuf, char *format, arg1, arg2, ...)
    - output is sent to outbuf
    - outbuf can be printed in one shot with fputs()
    - check output length
  - sscanf(char *inbuf, char *format, &arg1, &arg2, ...)
  - strtok

Good programming practices



- Use fgets to read input line
- Avoid buffer overflows
- Check for empty output
- Use sscanf to decode input
- Use strtok to get token from input

- Optional exercice for next week !
  - Write a simple command interpreter